

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)» (СГАУ)

**ТЕХНОЛОГИИ И ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА
РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ
МОБИЛЬНЫХ УСТРОЙСТВ**

Электронный учебно-методический комплекс
по дисциплине в LMSMoodle

Работа выполнена по мероприятию блока 1 «Совершенствование
образовательной деятельности» Программы развития СГАУ
на 2009 – 2018 годы по проекту «Разработка магистерской программы
«Программное обеспечение мобильных устройств» по направлению
230100.68 Информатика и вычислительная техника»
Соглашение № 1/12 от 3.06.2013 г.

УДК 004.432+004.4`2
Т 384

Автор-составитель: **Солдатова Ольга Петровна**

Технологии и инструментальные средства разработки программного обеспечения для мобильных устройств [Электронный ресурс] : электрон. учеб.- метод. комплекс по дисциплине в LMS Moodle/ Минобрнауки России, Самар.гос. аэрокосм. ун-т им. С. П. Королева (нац. исслед. ун-т); авт.-сост..О.П. Солдатова. - Электрон. текстовые и граф. дан.- Самара, 2013. —1 эл. опт.диск (CDROM).

В состав учебно-методического комплекса входят:

1. Курс лекций семестр 9.
2. Курс лекций семестр А.
3. Задания для лабораторных работ семестр 9.
4. Задания для лабораторных работ семестр А.
5. Вопросы к экзамену семестр 9.
6. Вопросы к экзамену семестр А.
7. Рабочая программа курса.

УМКД «Технологии и инструментальные средства разработки программного обеспечения для мобильных устройств» предназначен для студентов факультета информатики, обучающихся по направлению подготовки магистров 230100.68 «Информатика и вычислительная техника» в 9 и А семестрах.

УМКД разработан на кафедре информационных систем и технологий.

© Самарский государственный
аэрокосмический университет, 2013

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»
(СГАУ)

Технологии и инструментальные средства разработки программного обеспечения для мобильных устройств

Курс лекций

Семестр 9

Солдатова О. П.

Самара

2013

Содержание

1 ВВЕДЕНИЕ В РАЗРАБОТКУ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ	6
1.1 Разработка приложений для мобильных устройств	8
1.2 Платформы для разработки.....	9
1.3 Тестирование приложений	25
2 MAC OS X И IOS	27
2.1 Общие сведения о семействе ОС Mac OS.....	27
2.2 История ОС Mac OS	27
2.3 Операционная система Mac OS X.....	28
2.4 Эволюция ОС Mac OS X.....	30
2.5 Совместимость Mac OS X.....	32
2.6 Плюсы и минусы ОС Mac OS.....	34
2.7 История iOS.....	35
2.8 Принципы пользовательского интерфейса iOS.....	37
2.9 Характеристики платформы	38
3 РАЗРАБОТКА ПРИЛОЖЕНИЙ ПОД MAC OS X И IOS	47
3.1 Переменные	47
3.2 Целые и дробные числа	48
3.3 Типы данных.....	50
3.4 Математические операции.....	50
3.5 Логические операнды.....	51
3.6 Остаток от деления	51
3.7 Комментарии.....	52
3.8 Функции.....	53
3.9 Запуск приложения	57
3.10 Вывод на экран	58
3.11 Печать значений переменных	59
3.12 Использование фреймворка Foundation.....	61
3.13 Компиляция и запуск программ	62
3.14 Обзор Xcode	63
3.15 Наше первое приложение	67
4 ВВЕДЕНИЕ В ЯЗЫК ПРОГРАММИРОВАНИЯ OBJECTIVE-C	70
4.1 Основные понятия	70
4.2 Синтаксис языка	71
4.3 Создание новых классов	72
4.4 Механизм сообщений.....	74
4.5 Протоколы	78
4.6 Обработка исключений.....	79
4.7 Синхронизация	79
4.8 Создание и уничтожение объектов.....	80
4.9 Категории.....	81
4.10 Объекты Class objects и Objective-C runtime	82

4.11 Язык Objective-C 2.0	83
5 СТРУКТУРА ПРОГРАММЫ И ОСНОВНЫЕ ТИПЫ ДАННЫХ ЯЗЫКА OBJECTIVE-C	87
5.1 Структура программы	87
5.2 Типы данных.....	87
5.3 Переменные и константы.....	89
5.4 Массивы (Arrays)	91
5.5 Структуры	91
5.6 Указатели	91
6 ОСНОВНЫЕ ОПЕРАТОРЫ ЯЗЫКА OBJECTIVE-C	93
6.1 Операторы	93
6.2 Условия.....	93
6.2.1 Инструкция if.....	93
6.2.2 Конструкция if-else	94
6.2.3 Конструкция switch/case/default	96
6.3 Циклы.....	96
6.3.1 Цикл for	97
6.3.2 Вложенные циклы for	98
6.3.3 Варианты цикла for	99
6.3.4 Цикл while	100
6.4 Операторы Break и Continue.....	101
6.4.1 Оператор Break	101
6.4.2 Оператор Continue	102
7 ОСНОВЫ ООП.....	103
7.1 Жизненный цикл объекта	103
7.1.1 Основные понятия.....	103
7.1.2 Создание и использование объектов	103
7.1.3 Анатомия сообщений	104
7.1.4 Объекты в памяти	105
7.2 Создание классов.....	106
7.3 Определение методов и свойств	110
7.4 Указатели	114
7.5 Сообщения и методы	115
7.6 Создание объекта.....	119
7.7 Протоколы	125
7.8 Категории.....	126
7.9 Потоки	126
7.10 Исключения.....	127
7.11 Блоки.....	127
7.12 Наследование, инкапсуляция, полиморфизм	130
8 УПРАВЛЕНИЕ ПАМЯТЬЮ.....	134
8.1 Управление памятью Cocoa	134
8.2 Управление политикой памяти Cocoa	135
8.3 Практическое управление памятью Cocoa.....	138
8.4 Использование Autorelease пулов Cocoa.....	143

8.5 Автоматический подсчет ссылок (ARC) Сосоа	146
9 РАБОТА СО СТРОКАМИ	148
9.1 Характеристики строковых объектов в Сосоа.....	148
9.2 Создание и конвертация строковых объектов	148
9.3 Форматирование строковых объектов.....	151
9.4 Чтение и запись строк в файлы и URL	156
9.5 Поиск, сравнение и сортировка строк	159
9.6 Параграфы (абзацы) и разрывы строк.....	162
9.7 Символы и кластеры.....	163
9.8 Наборы символов	165
9.9 Сканеры	167
9.10 Строковые представления файловых путей	169
9.11 Отрисовка строк.....	171
9.12 Класс NSString	171
9.13 Класс NSMutableString.....	177
10 КОЛЛЕКЦИИ	178
10.1 Характеристики коллекций в Сосоа.....	178
10.2 Массивы: упорядоченные коллекции	179
10.3 Словари: коллекции ключ-значение.....	185
10.4 Наборы: неупорядоченные коллекции объектов.....	189
10.5 Индекс наборы: Хранение индексов в массиве.....	193
10.6 Индекс пути: хранение пути через вложенные массивы	194
10.7 Копирование коллекций	195
10.8 Перечисление: обход элементов коллекции	196
10.9 Параметры указателя на функцию	198
11 ПАТТЕРНЫ IOS	200
11.1 Паттерн Prototype.....	200
11.2 Паттерн Factory Method	201
11.3 Паттерн Abstract Factory	203
11.4 Паттерн Builder.....	206
11.5 Паттерн Singleton.....	209
11.6 Паттерн Adapter	211
11.7 Паттерн Bridge	215
11.8 Паттерн Facade.....	217
11.9 Паттерн Mediator.....	219
11.10 Паттерн Observer.....	221
11.11 Паттерн Composite.....	225
11.12 Паттерн Iterator	226
11.13 Паттерн Visitor	230
11. 14 Паттерн Decorator	234
11.15 Паттерн Chain of responsibility.....	235
11.16 Паттерн Template Method	237
11.17 Паттерн Strategy	239

11.18 Паттерн Command.....	241
11.19 Паттерн Flyweight.....	245
11.20 Паттерн Proxy.....	247
11.21 Паттерн Memento.....	249
11.22 Архитектурный шаблон проектирования MVC	251
12 РАЗРАБОТКА ПРИЛОЖЕНИЙ С ГРАФИЧЕСКИМ	
ИНТЕРФЕЙСОМ (GUI).....	255
12.4 Обзор Interface Builder	259
12.5 Создание связей	262
12.6 Создание кода	264
12.7 Поиск методов	267
12.8 awakeFromNib	270
12.9 Указатели	271
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ.....	274

1 Введение в разработку мобильных приложений

Компания VisionMobile провела глобальный опрос 6000 разработчиков под различные портативные платформы с целью выяснить современное состояние мобильных платформ с точки зрения разработчика, а также выявить тенденции, которые могут повлиять на будущее индустрии, которая закрепляется в форме дуополии Android и iOS. Само исследование охватывает период с начала первого квартала по начало третьего квартала этого года.

Первым пунктом исследования для VisionMobile стало выяснение, какие же платформы являются для разработчиков приоритетными. Согласно полученным данным, 71% разрабатывают свои приложения для Android, что на 1% меньше, чем в начале года, тогда как платформа iOS, наоборот, прибавила 1% до 57%. Третье место занимает даже не платформа как таковая, а вообще программирование под HTML5. Под этим пунктом понимается в первую очередь разработка веб-приложений для мобильных браузеров, а не упаковка HTML5/JavaScript в PhoneGap или аналогичные фреймворки для публикации в Google Play или Apple AppStore. Порядка 52% разработчиков работают над HTML5-приложениями, причём их количество увеличилось на 2% по сравнению с первым кварталом. А вот четвёртое место с большим отставанием занимает Windows Phone, под которую разрабатывают 21% опрошенных, притом их количество не изменилось за прошедшее время. Наибольшее удивление вызывает платформа Blackberry 10, которой лишь за полгода удалось убедить 15% разработчиков готовить приложения и для смартфонов канадской компании. На вопрос о приоритетности 34.4% разработчиков указали на Android, 32.7% выбрали iOS, 17.3% в первую очередь ставят на веб-приложения, тогда как на Windows Phone и Blackberry в первую очередь ориентируются 4.5% и 4.2% разработчиков соответственно.

Подводя итог, специалисты VisionMobile отмечают, что на рынке сформировалась дуополия Android и iOS, и они не видят в обозримом будущем предпосылок к смене данного явления. По их мнению в действие вступает сетевой эффект, при котором подавляющее превосходство Android и iOS как в относительной рыночной доле, так и в абсолютных числах поставок смартфонов приводит к тому, что ценность этих платформ растёт по закону Меткалфа. Аналитики VisionMobile отдельно отмечают, что Microsoft, вложившая в разработку и продвижение Windows Phone 7 и 8 не менее \$5 млрд, так и не смогла набрать более 3% рыночной доли за 2.5 года существования платформы. Аналитики считают, что на данный момент попросту невозможно победить Apple и Google, которые крепко окопались на своих позициях, а другие сегменты рынка настолько мизерны, что даже доминирование на отдельных участках не принесёт общего успеха. Единственным удачным приёмом борьбы против, например, Android в VisionMobile рассматривают политику перехвата опорных точек управления экосистемой, которую проповедует Amazon с Kindle и с недавнего времени Facebook (Facebook Home и прочее). По мнению аналитиков такое решение менее затратно, чем попытки вести войну на истощение, в которой у Android наиболее обеспеченный тыл.

В настоящее время существуют следующие технологии по разработке мобильных приложений:

1. Java. Приложения разрабатываемые на Java для устройств с ограниченными ресурсами (например, мобильные телефоны) называются мидлетами.

Технология Java состоит из двух элементов: языка программирования и операционной среды, в которой могут запускаться программы, написанные на этом языке. Синтаксис языка программирования Java похож на синтаксис C++ - оба языка объектно-ориентированы. Основное отличие между C++ и Java заключается в том, что разработчику приложений на C++ необходимо скомпилировать исходный код специально для конкретного устройства, для которого предназначено приложение.

Java-код интерпретируется непосредственно самим устройством при помощи так называемой Java Virtual Machine. Этот механизм делает возможным свободное

распространение Java-приложений, так как они работают на всех устройствах с аналогичной Java-платформой.

Программирование Java-приложений и на сегодняшний день имеет большую нишу, так как большинство телефонов в мире имеют установленную Java-машину.

2. Qt. Среда разработки Qt была приобретена Nokia в 2008 году у норвежской Trolltech за 150 миллионов долларов. Qt в основном используется в качестве кросс-платформенной среды, которая позволяет использовать написанные с ее помощью приложения на различных устройствах и операционных системах, в том числе Windows, Mac OS X, Linux, Symbian, Android и других. Начиная с версии Qt 4.0 появилась возможность программировать для мобильных устройств. С растущей пользовательской базой Qt, растёт потребность во встроенных, мобильных приложениях и UI-разработчиках.

Qt является одной из самых удачных библиотек для C++. Отладка под мобильные устройства происходит с помощью эмулятора, который содержится в среде разработки. Таким образом, мы можем писать сложные приложения для мобильных устройств с использованием библиотек C++ и поддержкой кроссплатформенности.

В настоящее время последняя версия - Qt 5 бета. Финальный релиз планируется на 2012 год. Для работы Qt на мобильных устройствах необходима установка соответствующего Фреймворка.

3. Windows Phone SDK. На момент написания этой статьи, последняя версия инструментария доступна в версии Windows Phone SDK 7.1 Release Candidate в лицензии «Go Live» с возможностью разрабатывать свои приложения и публиковать их в Windows Phone Marketplace. Windows Phone SDK 7.1 Release Candidate содержит следующие компоненты:

- Windows Phone SDK 7.1;
- Windows Phone Emulator;
- Windows Phone SDK 7.1 Assemblies;
- Silverlight 4 SDK and DRT;
- Windows Phone SDK 7.1 Extensions for XNA Game Studio 4.0;
- Expression Blend SDK for Windows Phone 7;
- Expression Blend SDK for Windows Phone OS 7.1;
- WCF Data Services Client for Windows Phone;
- Microsoft Advertising SDK for Windows Phone.

Код разрабатываемого приложения описывается на языке XAML. На самом деле - это просто XML файлы с языком разметки XAML.

Платформа Windows Phone не просто очередная платформа для мобильных устройств. Она содержит в себе не только технологическую составляющую, но и полностью проработанную концепцию дизайна интерфейса и взаимодействия с пользователем под названием Metro-дизайн или стиль Metro.

Вся разработка под Windows Phone ведется в среде Visual Studio. Среда является очень удобной для разработки и отладки приложений. Для мобильных приложений под Windows Phone отладка происходит с помощью эмулятора Windows Phone с помощью среды разработки Windows Phone.

4. iPhone SDK. Разработка под iPhone под операционную систему iOS возможна только под Mac OS X. Но в интернете можно найти статьи, как можно программировать и на Macintosh и даже на VM. Стоит заметить, что инструменты Apple предоставляет бесплатно, платить придется за подписку разработчика.

Для написания программ под iPhone предлагается использовать Objective-C. При этом есть возможность писать так же и на C и на C++ (для этого необходимо изменять расширения файлов с .m на .mm). Правда при этом полностью уйти от Obj-C не удастся, почти весь API рассчитан именно на Obj-C, исключения составляют например OpenGL (хотя для его инициализации придется использовать несколько строк кода на Obj-C), так

же полностью доступны стандартные библиотеки C/C++ (так, например, с файловой системой можно работать как средствами SDK на Obj-C, так и используя стандартную библиотеку C для ввода/вывода (fopen(), fgetc(), etc)).

Отладка приложения происходит с помощью среды XCode и эмулятора iPhone установленного в ней.

5. Android SDK. Для разработки под Android можно использовать среду Eclipse с установленным плагином ADT. Разработка ведется на языке программирования Java. Есть возможность отладки с использованием эмулятора встроенного в ADT или непосредственно на мобильном устройстве с ОС Android.

Существует различные версии SDK, которые используются для написания кода для различных версий Android. В настоящее время большое распространения получили версии 2.2 и 2.3. Поддерживается почти полная обратная совместимость версий.

Кроме разработки на языке Java поддерживается возможность более низкоуровневая разработка с использованием Android NDK (Native Development Kit) на языке C/C++.

6. Symbian и C++. Для написания приложений под Symbian можно использовать язык программирования C++. В основном данный подход используется для Symbian OS v5.1, 7.0, 7.0s и 8.0.

Разработка для Symbian OS (если говорить о Си++) обычно ведется на ПК. Среда разработки - привычная многим программистам Visual Studio, это также могут быть IDE Metrowerks CodeWarrior Development Studio, Borland C++BuilderX Mobile Edition, Carbide.C++ (относительно новая IDE, созданная компанией Nokia на базе Eclipse), снабженная дополнительными инструментальными пакетами (SDK). Разработчику доступны практически все привычные возможности в отношении как создания ПО, так и отладки (трассировка, просмотр переменных, стека вызовов, структур классов и др.).

Отлаживаемая программа запускается в эмуляторе Symbian OS. Отметим, что эту подсистему правильнее было бы назвать симулятором, поскольку имитируются не аппаратные средства, а лишь программное окружение (соответствующие API операционной системы, реализованные поверх API Win32). При этом программные модули, которые загружаются в эмулятор, представляют собой исполнимые файлы для архитектуры x86 (не ARM, на базе которой построены смартфоны), соответствующее ПО для целевой платформы формируется после итоговой компиляции. Это предполагает определенную специфику (скажем, ранее была довольно распространена ситуация, когда программа, нормально функционировавшая в среде эмулятора, отказывалась работать на реальном устройстве), но сегодня эмулятор обеспечивает достаточно высокую степень сходства и проблемы возникают лишь при создании программ, нестандартно использующих API.

1.1 Разработка приложений для мобильных устройств

Разработка приложений для мобильных устройств — это процесс при котором приложения разрабатываются для небольших портативных устройств таких как КПК, смартфоны или сотовые телефоны. Эти приложения могут быть предустановлены на устройство в процессе производства, загружены пользователем с помощью различных платформ для распространения ПО или являться веб-приложениями, которые обрабатываются на стороне клиента или сервера (JavaScript).

Android, iOS, BlackBerry, Open webOS, Symbian OS, Bada от Samsung, и Windows Mobile поддерживают стандартные бинарные файлы приложений как на персональных компьютерах с кодом выполняющимся на процессоре определенного формата (в основном используется архитектура ARM). Windows Mobile может быть скомпилирована для архитектуры x86 для отладки на ПК без эмуляции процессора, а также поддерживает формат Portable Executable (PE) связанный с .NET Framework. Windows Mobile, Android,

HP webOS и iOS предоставляют бесплатные SDK и интегрированные среды разработки для разработчиков.

1.2 Платформы для разработки

Каждая из платформ для мобильных приложений имеет интегрированную среду разработки, предоставляющую инструменты, позволяющие разработчику программировать, тестировать и внедрять приложения на целевую платформу. В таблице 1.1 собраны сведения о разработке приложений для каждой среды.

Таблица 1.1

	<u>Язык программи рования</u>	<u>Доступн ый отладчи к</u>	<u>Доступн ый эмулятор</u>	<u>Доступна я интегриро ванная среда разработк и</u>	<u>Платфор мы для разработк и</u>	<u>Установ очный пакет</u>	<u>Стоимость инструмен тов для разработк и</u>
<u>Adobe AIR</u>	Action Script, HTML, CSS, JavaScript	Да	Да	Flash Builder, Flash Professional	iOS (iPhone, iPad, iPod touch), Android, BlackBerry	Собственн ый формат для каждой платформ ы	Flash Builder, Flash Professional — Доступные коммерческ ие лицензии, Adobe AIR SDK (command line tool) — бесплатно
<u>Airplay SDK</u> (сейчас Marmalade)	C, C++	Да	Да	Visual Studio, XCode	Android, BlackBerry, BREW, iOS (iPhone), Maemo, Palm/webO S, Samsung bada, Symbian, Windows Mobile 5.x, OSX	Собственн ый формат для каждой платформ ы	Доступны коммерческ ие лицензии
<u>alcheMo</u>	Java	отладчик встроен в Visual Studio, Eclipse и XCode	эмулятор доступен в соответств ующих IDE	Visual Studio, Eclipse, XCode	Android, BREW, iOS (iPhone), Windows Mobile	Собственн ый формат для каждой платформ ы	Доступна коммерческ ая лицензия
<u>Android</u>	Java, частично C, C++	встроенн ый отладчик Eclipse, доступна	Да	Eclipse, Проект Kenai — плагин Android для	Android	apk	бесплатно

	<u>Язык программи рования</u>	<u>Доступн ый отладчи к</u>	<u>Доступн ый эмулятор</u>	<u>Доступна я интегриро ванная среда разработк и</u>	<u>Платфор мы для разработк и</u>	<u>Установ очный пакет</u>	<u>Стоимость инструмен тов для разработк и</u>
		автономная отладка		NetBeans, IntelliJ IDEA			
<u>AppFurnace</u>	JavaScript	Да	Да, эмуляция в сетевой IDE, тестирование на устройствах.	Облачная IDE	Android и iOS	Собственный формат для каждой платформы	бесплатно для разработки, тестирования и демонстрации. Платная публикация приложения.
<u>Application Craft</u>	JavaScript, HTML5, CSS	Да	Да, эмулятор используется в Ripple	Облачная IDE	Android, iOS, Blackberry, Windows Mobile, Bada, WebOS, Symbian.	основано на облаке/браузере	свободный и открытый источник
<u>Appcelerator</u>	JavaScript	Да, в Titanium Studio, в наст. время beta.	Эмулятор доступен в инструментах сторонних производителей	внутренний SDK	Android, iPhone, BlackBerry	Собственный формат для каждой платформы	лицензия Apache 2.0, доступна коммерческая лицензия
<u>Appceptio n</u>	HTML5, CSS, JavaScript	Да	Да	Облачная IDE	Android	Собственный формат для каждой платформы	доступны бесплатная и коммерческая лицензии
<u>appMobi</u>	JavaScript, CSS3, HTML5	Да, XDK (Google Chrome App Store)	Да, в XDK	интегрировано в предпочитаемую среду разработки (VS, Notepad ++,	iOS, Android, HTML5 Web Apps, HTML5 Hybrid Apps	Собственный формат для каждой платформы	бесплатно для разработки, платные облачные сервисы

	<u>Язык программи рования</u>	<u>Доступн ый отладчи к</u>	<u>Доступн ый эмулятор</u>	<u>Доступна я интегриро ванная среда разработк и</u>	<u>Платфор мы для разработк и</u>	<u>Установ очный пакет</u>	<u>Стоимость инструмен тов для разработк и</u>
				VI)			
<u>Aqua</u>	C, C++, JavaScript	Да	Да	Visual Studio, XCode, Eclipse (среда разработк)	Android, BlackBerry Playbook, iOS, Palm/webO S, Samsung bada, Windows Mobile 5.x, Windows Desktop	Собственн ый формат для каждой платформ ы	доступны бесплатная и коммерческ ая лицензии
<u>Basic4and roid</u>	Visual Basic	Да	Да	Да	Android	apk	доступна коммерческ ая лицензия
<u>BatteryTe ch</u>	C, C++	Да	собственн ый эмулятор для каждой платформ ы (iOS, Android...)	Eclipse, Visual Studio, XCode	Android, iOS (iPhone), Windows, OSX	Собственн ый формат для каждой платформ ы	доступна коммерческ ая лицензия
<u>Bedrock</u> (сейчас webMetho ds Mobile Designer)	Java	Да	Да	Eclipse	Java ME, Android, BREW, BlackBerry, Nintendo DS, iOS (iPhone/iPa d), Palm/webO S, Sony PSP, Samsung bada, Symbian, Windows Mobile, Windows Phone 7, Windows Desktop,	Собственн ый формат для каждой платформ ы	доступна коммерческ ая лицензия

	<u>Язык программи рования</u>	<u>Доступн ый отладчи к</u>	<u>Доступн ый эмулятор</u>	<u>Доступна я интегриро ванная среда разработк и</u>	<u>Платфор мы для разработк и</u>	<u>Установ очный пакет</u>	<u>Стоимость инструмен тов для разработк и</u>
					OSX		
BlackBerry у	Java	отладчик встроен в IDE	Да	Eclipse	BlackBerry	alx, cod	бесплатно
BREW	C; API предоставлен ы на C с интерфейсом в стиле C++.	Поддерж ка отладчик а для целевого ARM кода. Можно использо вать Visual Studio для отладки на x86.	Нет эмулятора для ARM кода. имеется симулятор для тестирова ния на x86.	Visual Studio 5.0, Visual Studio 2003 .NET, Visual Studio 2005	Компиляци я в специальну ю версию BREW доступна на КПК.	OTA	Ключи для разработки требуют Brew App Certification — ежегодная плата VeriSign для сертифицир ованного разработчик а. ARM компилятор BREW (доступен бесплатный GNU C/C++, но с ограниченн ой функционал ьностью и поддержкой). Требуется тестировочн ый ключ BREW для разработки приложений .
Canappi	mdsl	Собствен ные отладчик и для каждой платформ ы (iOS, Android ...)	Собственн ые эмуляторы для каждой платформ ы (iOS, Android...)	Eclipse, с плагином mdsl, Apple Interface Builder и Balsamiq для графическо го UI дизайна	iOS, Android.	собственн ое внедрение для каждой платформ ы.	доступны свободная и коммерческ ая лицензии

	<u>Язык программи рования</u>	<u>Доступн ый отладчи к</u>	<u>Доступн ый эмулятор</u>	<u>Доступна я интегриро ванная среда разработк и</u>	<u>Платфор мы для разработк и</u>	<u>Установ очный пакет</u>	<u>Стоимость инструмен тов для разработк и</u>
<u>CellSDK</u>	C#	Да	Да	Visual Studio and MonoDeveloper	Android, iOS and Windows Phone	Собственный формат для каждой платформы	доступны свободная и коммерческая лицензии
<u>Celsius</u>	Java	Да	Да	Eclipse	Java ME, Android, BlackBerry, iPhone, Symbian, Windows Mobile	Собственный формат для каждой платформы	доступна коммерческая лицензия
<u>CloudPact</u>	HTML, CSS, JavaScript	Да	Да	облачная IDE или загружаемая версия	Android, BlackBerry, iPhone, Windows Mobile	OTA или собственный формат для каждой платформы	бесплатная ограниченная trial-версия и планируется Enterprise-версия
<u>Codenam eOne</u>	Java	Да	Да	Netbeans, Eclipse (среда разработки)	iOS (iPhone, iPad, iPod touch), Android, BlackBerry, WIN7, J2ME	Собственный формат для каждой платформы	свободный и открытый источник + платные облачные сервисы
<u>CoStore</u>	сетевой drag-and-drop	N/A	Да	сетевой создатель приложений	iOS, Android, HTML5	Собственный формат внедрения или основанный на сети	бесплатная неограниченная trial-версия; доступны коммерческие лицензии для внедрения.
<u>Corona SDK</u>	Lua	Да	Да	Xcode	iOS, Android, NOOK	Собственный формат внедрения	бесплатная неограниченная trial-версия;

	<u>Язык программи рования</u>	<u>Доступн ый отладчи к</u>	<u>Доступн ый эмулятор</u>	<u>Доступна я интегриро ванная среда разработк и</u>	<u>Платфор мы для разработк и</u>	<u>Установ очный пакет</u>	<u>Стоимость инструмен тов для разработк и</u>
					Color	для каждой платформ ы	доступны коммерческ ие лицензии для внедрения.
<u>CrossMob</u>	C, C++	Да	собствен ный эмулятор или ОТА тестирова ние	Visual Studio, Xcode	iOS, Android	ОТА	свободная beta-версия
<u>DragonR AD</u>	визуальные drag-and-drop элементы	Да	использо ует сторонние эмуляторы	подходящая IDE	Android, BlackBerry, Windows Mobile	ОТА	доступны свободная и коммерческ ая лицензии
<u>FeedHenr y</u>	HTML, CSS, JavaScript	Да	Да	Studio включает полную IDE и плагин Eclipse.	Apple iPhone & iPad, Android, Windows Phone 7, Blackberry, Nokia WRT.	собствен ный формат для каждой платформ ы	свободная, профессион альная и планируется Enterprise версии
<u>Fivespark</u>	сетевой point-and- click редактор	N/A	Отобража ется в браузере	сетевой создатель приложени й	Android, BlackBerry, iOS и Windows Phone 7	HTML5 ОТА	бесплатно для разработки
<u>GameMa ker Studio</u>	Drag-and- Drop, GML (GameMaker Language)	Да	Да	встроенная IDE	Android, Apple iOS (iPhone, iPad), HTML5, Mac OS X, Windows	Собственн ый формат для каждой платформ ы	Коммерческ ая лицензии
<u>GeneXus for Mobile and Smart Devices</u>	Представлен ие знаний и декларативно е программиро вание для	Нет	Нет	подходящая IDE	Android, Apple iOS (iPhone, iPad), BlackBerry и HTML5	Собственн ый формат для каждой платформ	Бесплатно для пробы, доступны коммерческ ая и Enterprise

	<u>Язык программи рования</u>	<u>Доступн ый отладчи к</u>	<u>Доступн ый эмулятор</u>	<u>Доступна я интегриро ванная среда разработк и</u>	<u>Платфор мы для разработк и</u>	<u>Установ очный пакет</u>	<u>Стоимость инструмен тов для разработк и</u>
	простой разработки, код автоматическ и генерируется для каждой платформы				по желанию	и также основанн ый на облаке/бр аузере	лицензии
<u>Gideros Mobile</u>	Lua	No	Да	подходящая IDE	iPhone, iPad, iPod Touch, Android	Собственн ый формат для каждой платформ ы	доступны бесплатная и коммерческ ая лицензии
<u>Haxe NME</u>	Haxe	N/A	Собственн ый эмулятор для каждой платформ ы (iOS, Android...)	FlashDevelo p	iOS, Android, webOS, BlackBerry,	Собственн ый формат для каждой платформ ы	бесплатная и открытая
<u>IBM Worklight</u>	HTML5, CSS и JavaScript	Да	поставляет ся с IDE.	плагины на Eclipse	Android, BlackBerry, iOS и Windows Phone 7	Собственн ый формат для каждой платформ ы.	бесплатная ознакомител ьная версия, доступна коммерческ ая лицензия.
<u>iOS SDK</u>	Objective-C	отладчик встроен в Xcode IDE	в комплекте с iPhone SDK, интегриро ван с Xcode IDE	Xcode	iPhone, iPad, iPod Touch	только через App Store, требуется проверки и утвержден ия Apple Inc.	Инструмент ы бесплатны для основанных на Intel Mac. Тестирован ие на симуляторе бесплатно, но установка на устройство

	<u>Язык программи рования</u>	<u>Доступн ый отладчи к</u>	<u>Доступн ый эмулятор</u>	<u>Доступна я интегриро ванная среда разработк и</u>	<u>Платфор мы для разработк и</u>	<u>Установ очный пакет</u>	<u>Стоимость инструмен тов для разработк и</u>
							требуется платный ключ разработчик а.
<u>iOS SDK</u>	Object Pascal	отладчик встроен в Xcode IDE	включен в Delphi XE2 professional и выше	Embarcadero Delphi XE2	iPhone, iPad, iPod Touch	только через App Store, требуется обзор и одобрение Apple Inc.	Разработка требуется основанный на Intel Mac и IDE на Windows. Разработка на Windows. Компиляция и внедрение должны выполняться на Mac. Тестирован ие на симуляторе бесплатно, но установка на устройство требуется платный ключ разработчик а.
<u>IwGame Engine</u>	C, C++	отладчик и Visual Studio / XCode	Да, через Marmalade SDK эмулятор	Visual Studio / XCode	iPhone, iPod и iPad, Android, Bada, Blackberry BBX (Playbook), Symbian, WebOS, Windows Mobile, Mobile Linux, LG- TV, Windows	Собствен ный формат для каждой платформ ы.	Бесплатный и открытый исходник, использует Marmalade SDK -[1]

	<u>Язык программи рования</u>	<u>Доступн ый отладчи к</u>	<u>Доступн ый эмулятор</u>	<u>Доступна я интегриро ванная среда разработк и</u>	<u>Платфор мы для разработк и</u>	<u>Установ очный пакет</u>	<u>Стоимость инструмен тов для разработк и</u>
					Desktop, Mac OSX		
<u>Java ME</u>	Java	Да	доступен эмулятор, Sun Java Wireless Toolkit, mpowerpla yer	Eclipse, LMA NetBeans Mobility Pack	многие реализации VM имеют зависящие от устройства ошибки	Jad/Jar пакет; PRC файлы в PalmOS	бесплатно
<u>JMango</u>	JMango	N/A	N/A	JMango Flash IDE	Java ME, Android, Bada, BlackBerry, iPhone, Windows Mobile 6, Windows Phone 7	Собственн ый формат для каждой платформ ы	бесплатно
<u>July Systems Mi™ Platform</u>	Java, HTML, Groovy, FreeMarker, Java Script	Да	Да	IDE — Eclipse с GUI Editor	iOS, Android, Blackberry, WP7, Mobile Web, HTML5, QT	Собственн ый формат для каждой платформ ы	доступна коммерческ ая лицензия
Kony	Графический Drag-and- Drop	Да	использоу ет собственн ый эмулятор для каждой платформ ы(iOS, Android, BlackBerry , Windows Phone 7)	основанные на Eclipse	Android, BlackBerry, iOS, Java ME, Palm/webO S, Symbian, Windows Phone 7. Поддержка мобильного веб- браузера (WML через оптимизиро ванный для устройства	Собственн ый формат для каждой платформ ы	доступна коммерческ ая лицензия

	<u>Язык программи рования</u>	<u>Доступн ый отладчи к</u>	<u>Доступн ый эмулятор</u>	<u>Доступна я интегриро ванная среда разработк и</u>	<u>Платфор мы для разработк и</u>	<u>Установ очный пакет</u>	<u>Стоимость инструмен тов для разработк и</u>
					HTML5)		
<u>Lazarus</u>	Object Pascal	Да, можно отлажива ть в IDE через ActiveSyn с для Windows CE	Собственн ый эмулятор для каждой платформ ы	Lazarus IDE, включает интегриров анный GUI дизайнер и отладчик	Компилиру емый язык программи рования доступен для Windows CE, устройств на Linux, Symbian OS в разработке	Собственн ый формат для каждой платформ ы	бесплатно
<u>Macrome dia Flash Lite</u>	ActionScript	Да	связан с IDE	Macromedia Flash MX2004/8, Eclipse	Да	SIS / CAB внедрение или OTA/IR/B luetooth SWF files	Различно, бесплатно но ограничено с MTASC
основанны е на <u>Microbro wser</u>	XHTML (WAP 2.0), WML (WAP 1.2)	Да	много	много	Стандартн ый рендеринг страниц с постраничн ой настройкой для разных браузеров.	Нет данных	бесплатно
<u>Meme IDE</u>	MemeScript	Проверка предоста вляется в обзоре проблем.	Да, может быть интегриро ван эмулятор Android	Eclipse RCP	Android, Windows Mobile	Собственн ый формат для каждой платформ ы	бесплатно для разработки
<u>MobiFlex</u>	Графический drag & drop.	N/A	N/A	Web Portal	Android, iPhone	N/A	свободно только для разработки
<u>MobileNa tionHQ</u>	визуальный paradigm/java	N/A	N/A, моменталь ное	интегриров анная SaaS	Android, iPhone	N/A	свободная и коммерческ

	<u>Язык программи рования</u>	<u>Доступн ый отладчи к</u>	<u>Доступн ый эмулятор</u>	<u>Доступна я интегриро ванная среда разработк и</u>	<u>Платфор мы для разработк и</u>	<u>Установ очный пакет</u>	<u>Стоимость инструмен тов для разработк и</u>
	script		облачное внедрение.	среда			ая лицензии
<u>Moscrif</u>	JavaScript	N/A	поставляет ся с <u>Moscrif SDK</u>	Встроенны й SDK (Mono based)	Android, iOS (iPhone), Samsung bada, Symbian, Windows Mobile	Собственн ый формат для каждой платформ ы.	свободная и коммерческ ая лицензии
<u>Mono for Android</u>	C#	Да	Да	Visual Studio 2005 и MonoDevel op	Android	Собственн ый формат для каждой платформ ы	
<u>MonoTou ch</u>	C#	Да	Да	Visual Studio 2005 и MonoDevel op	iOS	Собственн ый формат платформ ы	
<u>MoSync</u>	C, C++, Lua, HTML5, CSS, JavaScript	Да	Да	Eclipse, Visual Studio 2005 и выше, MoBuild w/ текстовые редакторы	Android, Java ME, Moblin, iOS (iPhone), Smartphone 2003, Symbian, Windows Mobile (Pocket PC), Blackberry (экспериме нтально)	SIS, CAB, JAD, JAR, APK, OTA	бесплатно, GPL 2.0, бесплатная подписка Indie; доступна коммерческ ая подписка.
<u>Toura Mulberry</u>	HTML5, CSS, JavaScript	Да	Нет, инструмен ты сторонних производи телей	Нет, инструмент ы сторонних производит елей	Android, Apple iOS, Mobile Web	Собственн ый формат для каждой платформ ы, OTA.	бесплатно

	<u>Язык программи рования</u>	<u>Доступн ый отладчи к</u>	<u>Доступн ый эмулятор</u>	<u>Доступна я интегриро ванная среда разработк и</u>	<u>Платфор мы для разработк и</u>	<u>Установ очный пакет</u>	<u>Стоимость инструмен тов для разработк и</u>
<u>NeoMAD</u>	Java	Да	эмуляторы сторонних производи телей	плагины Eclipse и Netbeans	Java ME, Android, BlackBerry, Windows Phone 7, iOS (iPhone) и скоро Samsung bada	Собственн ый формат для каждой платформ ы	доступна коммерческ ая лицензия
<u>.NET Compact Framework</u>	C#, VB.NET, Basic4ppc	Да	бесплатны й эмулятор, доступен исходный код, также встроен в IDE	Visual Studio 2008, 2005, 2003, Basic4ppc IDE	Windows Mobile, Windows CE, устройства на Symbian с использова нием сторонних инструмент ов.	OTA файлы CAB, ActiveSyn с	большинств о инструмент ов бесплатно, но требуется коммерческ ие версии Visual Studio для графическог о дизайна.
<u>NS BASIC/A pp Studio</u>	Visual Basic	Да	Нет. Тестирова ние в браузерах Chrome или Safari	подходящая IDE	iPhone, iPad, iPad Touch, Android 2.1+	Java Web App или собственн ое внедрение с использов анием Phonegap	доступны ознакомител ьная версия и коммерческ ая лицензия.
<u>OpenPlug</u>	ActionScript, XML	Да	Да	плагин OpenPlug ELIPS для Adobe Flash Builder	Android, iOS (iPad, iPhone, iPod Touch), Symbian, Windows Mobile	Собственн ый формат для каждой платформ ы	свободная и коммерческ ая лицензии
<u>Palm OS</u>	C, C++, Pascal	Да	OS 1.0 — 4.1: бесплатны й эмулятор	Palm OS Developmen t System (Eclipse), CodeWarrio	Palm OS КПК, или Windows Mobile с эмулятором	файлы PRC, , PalmSourc е установи	бесплатно (POSE или GCC для Palm OS), или платно

	<u>Язык программи рования</u>	<u>Доступн ый отладчи к</u>	<u>Доступн ый эмулятор</u>	<u>Доступна я интегриро ванная среда разработк и</u>	<u>Платфор мы для разработк и</u>	<u>Установ очный пакет</u>	<u>Стоимость инструмен тов для разработк и</u>
			предостав ленный PalmSource (Access); OS 5.0: — 5.4 Симулято ры конкретны х устройств предостав лены Palm (palmOne)	г. PocketStudi o, HB++, Satellite Forms	StyleTap.	ик (.psi)	(CodeWarrio r), или различные платные фреймворки для быстрой разработки
<u>Particle SDK</u>	Java, ActionScript	Да	Да	плагин Eclipse с редактором GUI	Android, BlackBerry, iOS, webOS, и устройства на Windows Phone 7 + Flash, HTML5 веб- приложени я	Собствен ный формат для каждой платформ ы	свободная beta-версия. Будет доступна коммерческ ая лицензия.
<u>PhoneGap</u>	HTML, CSS, JavaScript	Да	Нет, сторонние инструмен ты.	Нет, сторонние инструмент ы.	iPhone, Android, Windows Phone, BlackBerry, Symbian, Palm	Собствен ный формат для каждой платформ ы	лицензия MIT
<u>Python</u>	Python	Да	Add-on для Nokia Emulator	Различные, включая плагины для Eclipse	Интерпрети руемый язык программи рования доступен только на Nokia Series60, но существую т порты на другие	Sis внедрение с py2sis или можно использо вать Python Runtime	свободно

	<u>Язык программи рования</u>	<u>Доступн ый отладчи к</u>	<u>Доступн ый эмулятор</u>	<u>Доступна я интегриро ванная среда разработк и</u>	<u>Платфор мы для разработк и</u>	<u>Установ очный пакет</u>	<u>Стоимость инструмен тов для разработк и</u>
					платформы, включая PalmOS		
<u>Qt SDK</u>	C++, QML	Да	Да	Qt Creator	Symbian, Maemo, MeeGo, Linux, Windows, Mac OS X	собствен ный формат для каждой платформ ы	свободная и коммерческ ая лицензии
<u>RareWire - App Creation Studio</u>	XML	Да	Да	облачная IDE	iOS (iPhone, iPad, iPod touch), Android(ско ро)	собствен ный формат для каждой платформ ы	свободные разработка, тестировани е и демонстрац ия. Платная публикация.
<u>Resco MobileFo rms Toolkit</u>	C#	Да	Да	Visual Studio	Windows Mobile (Pocket PC), Windows CE, Android, iOS, Smartphone	собствен ный формат для каждой платформ ы	доступны бесплатная и коммерческ ая лицензии
<u>Rhomobil е</u>	Ruby с особенностя ми интерфейса HTML, компилирую щийся в приложение для каждой платформы.	Да	N/A, приложен ия могут запускатьс я на Win32 runner, или на эмуляторе устройств а для поддержи ваемых платформ.	xCode или Eclipse, по требованию версия RhoHub включающа я полную IDE	iOS (вкл. 3.0)(iPhone, iPad), Windows Mobile 5.1 Professional , Mobile Windows 5.0 Standard, BlackBerry 4.6, 4.7, 5.0, 5.0 (BlackBerry 4.2 и 4.5 поддержив ается, но доступ к базе	OTA, iOS через App store, .SIS, .CAB, .APK, .COD	Rhodes — бесплатный и открытый источник с лицензией MIT, RhoSync — GPL или коммерческ и. Доступна коммерческ ая поддержка. Подписка на RhoHub.

	<u>Язык программи рования</u>	<u>Доступн ый отладчи к</u>	<u>Доступн ый эмулятор</u>	<u>Доступна я интегриро ванная среда разработк и</u>	<u>Платфор мы для разработк и</u>	<u>Установ очный пакет</u>	<u>Стоимость инструмен тов для разработк и</u>
					данных очень медленный на этих устройства х), Symbian и Android 1.6 и выше		
<u>Smartface Platform</u>	Drag-and- drop инструменты и редактирован ие действий	Нет, не нужен	Да	Smartface Designer	Android, BlackBerry, J2ME, Symbian S60	Собственн ый формат для каждой платформ ы.	лицензия Community
<u>Stencyl</u>	Drag-and- drop редактор основанный на MIT Scratch, Objective-C	Да	Да	Xcode	iOS (iPad, iPhone, iPod Touch)	собственн ый формат для каждой платформ ы	доступны бесплатная и коммерческ ая лицензии для разработки.
<u>Symbian</u>	C++	Да	Да	Различные	Symbian	SIS внедрение	доступны бесплатные и коммерческ ие инструмент ы
<u>Tiggzi Mobile App Builder</u>	сетевой визуальный редактор, HTML5, CSS, JavaScript, jQuery Mobile, PhoneGap	Да, в специаль ном окне отладки.	Не нужен, тестирова ние в браузере или на устройств е.	сетевая IDE, можно экспортиро вать в проект Eclipse или Maven.	Mobile Web (HTML/JS/ CSS), iOS, Android, BlackBerry, Windows Phone 7	собственн ый формат для каждой платформ ы или mobile Web (HTML/JS /CSS)	бесплатно и платная месячная подписка для разработки, доступны опции размещения.
<u>TotalCross</u>	Java	Да	Да	Eclipse, TKN Mobile Studio для	Android, BlackBerry, iOS	Собственн ый формат	SDK — открытый исходник и

	<u>Язык программи рования</u>	<u>Доступн ый отладчи к</u>	<u>Доступн ый эмулятор</u>	<u>Доступна я интегриро ванная среда разработк и</u>	<u>Платфор мы для разработк и</u>	<u>Установ очный пакет</u>	<u>Стоимость инструмен тов для разработк и</u>
				TotalCross	(iPhone, iPad), Palm OS, Windows Mobile, Windows Phone 7, планируются Symbian	для каждой платформы (cab, jad, apk, pdb/prc, deb)	бесплатно для разработки на desktop. VM для целевого устройства должна иметь лицензию
<u>Unity</u>	C#, JavaScript, Boo, другие языки основанные на .NET	Да	Управление используется для моделирования взаимодействия устройств перед загрузкой приложения на устройство.	Unity Editor, также работает с Visual Studio и MonoDevelop.	Android, iOS (iPhone/iPad), PC, Mac, desktop browser, XBOX360, PS3, Wii, BlackBerry Playbook, Nokia Symbian, Roku 2 и другие доступны через программу Union.	собственный формат для каждой платформы.	бесплатная и коммерческая лицензии для разработки.
<u>WebORB Integration n Server</u>	C#, VB.NET, Java, PHP, ActionScript, JavaScript, Objective-C, XML	Да	использует эмулятор для соответствующих SDK.	работает с Eclipse, Visual Studio, IntelliJ IDEA и Amethyst IDE	Android, iOS (iPhone/iPad), BlackBerry Playbook, Windows Phone7	собственный формат для каждой платформы.	бесплатные лицензии для разработки; Free and Commercial deployment licenses
<u>webOS</u>	JavaScript, CSS, HTML, C and C++ through the PDK	Да	Да	Eclipse	webOS, только Palm	OTA, webOS через App store, Web URL, Precentral, .ipk	бесплатно

	<u>Язык программи рования</u>	<u>Доступн ый отладчи к</u>	<u>Доступн ый эмулятор</u>	<u>Доступна я интегриро ванная среда разработк и</u>	<u>Платфор мы для разработк и</u>	<u>Установ очный пакет</u>	<u>Стоимость инструмен тов для разработк и</u>
<u>WinDev Mobile</u>	WLanguage	Да	Да	WinDev Mobile, Android DSK	Android, Windows Mobile	OTA, apk, файлы CAB, ActiveSyn с	доступны коммерческ ие лицензии
<u>Windows Mobile</u>	C, C++	Да	эмулятор (доступны й исходник), также встроен в IDE	Visual Studio 2010, 2008, 2005, eMbedded VC++ (бесплатно) , Satellite Forms	Windows Mobile, Windows FU, Windows CE	OTA, файлы CAB, ActiveSyn с	доступные инструмент ы командной строки или eMbedded VC++, или Visual Studio (Стандартна я версия или лучше)
<u>Windows Phone</u>	C#	Да	Доступны й эмулятор, поставляет ся с IDE	Visual Studio 2010	Windows Phone	OTA, файлы XAP	
<u>mobileFX Studio 7</u>	J2me, Java, HTML5, JavaScript	Да, WTK для J2me, Интеграц ия с отладчик ом Chrome V8 для JavaScript	Да, WTK для BlackBerry SDK для J2me, браузеры WebKit, iPhone, iPad и устройств а на Android для HTML5/Ja vaScript	mobileFX Studio 7 (Windows XP, Windows 7)	платформы PhoneGap и собственны е компилятор ы для J2me (BlackBerry , NOKIA S40/S60, SE JP7/8, Motorola, Samsung, LG, и т. д.)	OTA (SMS), Bluetooth (OBEX), OTA (размещен ие), QRCode- to-mobile	бесплатно

1.3 Тестирование приложений

Сначала приложение тестируется в среде разработки с использованием эмулятора. После этого приложение тестируется на устройстве. Эмуляторы являются простым способом тестировать приложение на мобильном телефоне, не используя его физически. Ниже представлен список доступных инструментов для тестирования приложений среди самых популярных мобильных операционных систем:

1. Google Android Emulator.

Android Эмулятор запускается на Windows как отдельное приложение без необходимости полностью загружать и устанавливать Android SDK.

2. Официальный Android SDK Emulator.

Включает в себя эмулятор мобильного устройства, который реализует все аппаратные и программные особенности типичного устройства.

3. MobiOne.

MobiOne Developer это mobile Web IDE для Windows помогающее разработчику программировать, тестировать, отлаживать упаковывать и внедрять мобильные веб-приложения на устройства, такие как iPhone, BlackBerry, устройства на Android и Palm Pre.

4. TestiPhone.

Основанный на веб-браузере симулятор для быстрого тестирования веб-приложений для iPhone. Работает с использованием Internet Explorer 7, Firefox 2 и Safari 3.

5. iPhoneu.

Предоставляет точную среду веб-браузера, разработана Safari. Может быть использована для разработки веб-сайтов для iPhone. Не является эмулятором iPhone. iPhoneu запускается только на Mac OS X 10.4.7 и выше.

6. BlackBerry Simulator.

Существует множество официальных эмуляторов BlackBerry. С любым из них возможна проверка того, как ПО, экран, клавиатура устройства будут работать с приложением.

iOS (до 24 июня 2010 года — iPhone OS) — мобильная операционная система, разрабатываемая и выпускаемая американской компанией Apple. Была выпущена в 2007 году; первоначально — для iPhone и iPod touch, позже — для таких устройств, как iPad и Apple TV. В отличие от Windows Phone и Google Android, выпускается только для устройств, производимых фирмой Apple.

Пользовательский интерфейс iOS основан на концепции прямого манипулирования с использованием жестов мультитач. Элементы управления интерфейсом состоят из ползунков, переключателей и кнопок.

iOS разработана на основе OS X и использует тот же набор основных компонентов Darwin, совместимый со стандартом POSIX.

В iOS есть четыре слоя абстрагирования: слой Core OS, слой Core Services, слой Media Layer, и слой Cocoa Touch.

Для текущей версии операционной системы (iOS 7.0.2) выделяется 1,4—2 Гб флеш-памяти устройства для системного раздела и примерно 800 Мб свободного места (варьируется в зависимости от модели).

По состоянию на 19 мая 2013 года магазин приложений App Store содержит более 900 тыс. приложений для iOS, которые все вместе были загружены более 50 миллиардов раз [1].

2 Mac OS X и iOS

2.1 Общие сведения о семействе ОС Mac OS

Mac OS (Macintosh Operating System) — семейство проприетарных операционных систем с графическим интерфейсом. Вместе с Mac OS X вторая по популярности в мире операционная система (рыночная доля в июле 2009 года — ▼4,86 %).¹ Разработана корпорацией Apple (ранее — Apple Computer) для своей линейки компьютеров Macintosh. Популяризация графического интерфейса пользователя в современных операционных системах часто считается заслугой Mac OS. Она была впервые представлена в 1984 году вместе с оригинальным Macintosh 128K. Apple хотела, чтобы Макинтош представлялся как компьютер «для всех остальных» («for the rest of us»).

Сам термин «Mac OS» в действительности не существовал до тех пор, пока не был официально использован в середине 1990-х годов. С тех пор термин применяется ко всем версиям операционных систем Макинтоша как удобный способ выделения их в контексте других операционных систем. Ранние версии Mac OS были совместимы только с Макинтошами, основанными на процессорах Motorola 68k, следующие версии были совместимы с архитектурой PowerPC (PPC). С недавних пор Mac OS X стала совместима с архитектурой Intel x86. Но политика фирмы Apple такова, что она разрешает устанавливать систему Mac OS только на компьютеры Apple.

2.2 История ОС Mac OS

Mac OS вышла в свет в 1984 году вместе с первым персональным компьютером Macintosh от компании Apple. Идеи, воплощенные в первой версии системы Mac OS, ее авторы почерпнули у фирмы Xerox. В исследовательском центре Xerox PARC в то время уже существовал компьютер с графической операционной системой, что тогда было настоящим прорывом в эволюционном развитии операционных систем. Но они использовали ее только для собственных нужд и не планировали коммерческого применения. Соединив уже имеющиеся наработки и собственные идеи, программисты компании Apple создали Mac OS, первую доступную для всех графическую операционную систему. В ней уже тогда был использован всем нам привычный оконный интерфейс, папки с файлами, и впервые был применен манипулятор, названный компьютерной мышью, способный передвигать курсор по всей области экрана. Такая концепция вполне соответствовала главной идее самой компании Apple, предлагавшей создать компьютер доступный для всех, как по цене, так и в техническом плане.

Уже тогда авторы Mac OS задались целью создать такую операционную систему, которая будет удобна и интуитивно понятна любому пользователю, даже не изучавшему компьютер ранее. До этого времени компьютерными программами и процессами управляли с помощью командной строки, поэтому появление наглядных значков обозначавших папки, корзину, компьютер и другие элементы стало настоящей сенсацией того времени. Эти значки авторы стали называть иконками. Разработчики Mac OS определили основы дальнейшего развития всех операционных систем, многое из того, что они придумали в далеком 1984 году, сейчас является эталоном для всех разработчиков программного обеспечения. Первая версия Mac OS занимала всего 216 кб дискового пространства и работала даже при обычном копировании с одного компьютера на другой. Но такой продукт был совершенно не защищен от подделки, поэтому для того, чтоб сохранить свои доходы разработчики все дальнейшее время посвятили не только ее техническому усовершенствованию, расширению функциональности и стабильности, но и защите. После выхода первой версии Mac OS вышло еще девять ее модификаций, в которых были введены такие улучшения как:

- использование мульти файндера, позволяющего работать сразу нескольким приложениям одновременно;

- усовершенствование мультимедийных функций и возможностей работы в интернете;
- смена черно-белых иконок на цветные, а позже появление нового «платинового» интерфейса и возможности выбирать тему на свой вкус;
- появление новой файловой системы HFS+;
- улучшение функций инсталлеров;
- появление программы Sherlock, предназначенной для поиска файлов на локальных дисках и в сети интернет;
- появление многопользовательского режима и разграничение прав доступа.

Пиком работы над образом, усовершенствованием возможностей и обеспечением стабильности работы операционной системы Mac OS того времени стала ее девятая версия, которую сейчас еще можно встретить на некоторых компьютерах Macintosh. Ее называют классикой Mac OS. Macintosh завоевывал популярность, а Mac OS принесла компании огромный коммерческий успех. Но шло время и руководство компании Apple приняло решение о глобальных переменах.

В марте 2000 года свет увидела абсолютно новая десятая версия Mac OS X, которая не поддерживает программное обеспечение, предназначенное для Mac OS 9. Для ее создания полностью переписали начальный программный код, а за основу взяли операционную систему NeXTSTEP, разработчиком которой был один из тех, кто когда-то основал компанию Apple, но по стечению обстоятельств покинул ее. Разработки фирмы NEXT вместе с ее руководителем были куплены компанией Apple, благодаря чему во главе нее снова встает ее основатель Стив Джобс.

Теперь в Mac OS X используется ядро Mach, стандартные сервисы BSD и все основные возможности операционной системы Unix. Это дало возможность в много раз повысить ее функциональность, защищенность и стабильность. Вытесняющая многозадачность, которая используется в Mac OS X, позволяет работать нескольким процессам сразу, но при этом не мешать друг другу, а при сбое в работе одного из них не допускать сбоя всей системы и прерывания работы других процессов.

На данный момент Mac OS X имеет собственный красивый, не перегруженный спецэффектами и приятный для глаз интерфейс Aqua. Она проста в использовании и дружелюбна. В ней используется среда программирования Core Foundation, включающая в себя такие компоненты как Carbon API, Cocoa API и Java API. Графическая среда представлена использованием таких технологий как QuickTime, Quartz Extreme и OpenGL. К тому же Mac OS X позволяет использовать программное обеспечение, написанное на таких языках программирования, как Си, C++, Objective-C, Ruby и Java. Немаловажным достоинством Mac OS X является ее безопасность при работе в интернете, она неплохо защищена от интернет-атак, да и количество вирусов способных ее поразить на сегодняшний день ничтожно мало.

Если учитывать все достоинства Mac OS, то сразу встает вопрос, почему она до сих пор не так широко распространена, как ее основной конкурент всем известная ОС Windows. Ответ на него очень прост, ОС Mac OS устанавливается только на компьютеры Macintosh компании Apple, что является принципиальным решением ее руководства. Это не значит, что Mac OS не совместима с компьютерами других производителей и не может быть на них установлена. Установить систему конечно можно, но на данный момент это считается не законным. Компьютеры Macintosh имеют небольшой модельный ряд и не могут похвастаться разнообразием, к тому же они стоят в несколько раз дороже обычных персональных ЭВМ, поэтому число пользователей Macintosh сравнительно мало относительно общего числа компьютерных пользователей во всем мире.

2.3 Операционная система Mac OS X

Mac OS X абсолютно новая версия операционной системы Mac OS, основой которой является свободное программное обеспечение, называемое Darwin. Darwin

включает в себя ядро XNU, созданное с использованием технологии Mach и стандартных сервисов BSD. Главным преимуществом данной технологии является то, что ее фундамент основан на всем известной системе UNIX, что обеспечивает новой Mac OS повышенную стабильность, многофункциональность, скорость работы, безопасность и надежность.

Все модификации операционной системы Mac OS X названы различными видами животных из семейства кошачьих. Начиная с версии 10.0 до версии 10.6, операционные системы носят имена: Cheetah (в переводе Гепард), Puma, Jaguar, Panther, Tiger, Leopard и самая последняя версия Snow Leopard (в переводе Снежный Леопард). Каждая из версий имеет свои улучшения и доработки, но всем им присущи общие нововведения, которые делают Mac OS X абсолютно не похожей на Mac OS 9, это привело к тому, что приложения, написанные для предыдущих версий Mac OS, не будут работать на Mac OS X. Изначально работа приложений, написанных под Mac OS 9, осуществлялась посредством виртуальной эмуляции, но со временем от этого отказались совсем.

Неоспоримым преимуществом визуализации графической оболочки Mac OS X является новый интерфейс пользователя, который носит название Aqua. Графический интерфейс Aqua создает иллюзию водной среды, с ощущением прозрачности, глубины и движения. Он остается практически неизменным во всех модификациях Mac OS X, но все это время не теряет своей актуальности и свежести решения. Aqua создан не только для красоты, но и для комфорта и удобства пользователей. Одним из нововведений является возможность складывать окна активных приложений в отдельные секции, которые при желании можно скрыть из зоны видимости, не давая им без необходимости заполнять полезное пространство рабочего стола.

В новой версии Mac OS X реализованы такие функции как, защита памяти, благодаря которой исключается зависание всей системы при сбое в работе одного из приложений, а также вытесняющая многозадачность и симметричная многопроцессорность. В Mac OS X реализовано динамическое распределение памяти, что позволяет выделять и освобождать память непосредственно в процессе работы программ. Это исключает появление сообщений «out of memory» и не требует корректировки количества доступной памяти для каждого приложения.

Лишним подтверждением того, что Mac OS разрабатывается в первую очередь для удобства пользователя, является автоматическая интеграция в сеть. Использование этой функции позволяет подключаться к сети интернет любым доступным методом соединения без дополнительных настроек со стороны пользователя. Новая технология энергосбережения позволяет компьютеру выходить из состояния сна практически мгновенно.

В Mac OS X реализована возможность многопользовательского доступа к данным, разграничены права доступа, повышена степень защиты информации пользователей, администрирование системы стало еще более простым, а интерфейс управления шрифтами и шрифтовыми наборами стал более удобным и гибким, к тому же Mac OS X теперь напрямую поддерживает такие форматы шрифтов как: TrueType, Type1 и OpenType. В системе Mac OS X теперь есть встроенная поддержка принтеров фирм Epson, Hewlett-Packard и Cannon.

Двухмерная визуализация и печать документов в Mac OS X организована с помощью технологии Quartz, которая использует уже ставший стандартным графический формат PDF, что обеспечивает расширенную поддержку шрифтов и другие графические возможности системы. Для работы с аудиоданными и видеоданными в Mac OS X используется запатентованная технология QuickTime, которая теперь интегрирована в систему, а 3D графика и игры реализованы с использованием технологии OpenGL.

Mac OS X значительно отличается от предыдущих версий Mac OS. Основу системы составила POSIX-совместимая операционная система Darwin, которая является свободным программным обеспечением. Ее ядром является XNU (рекурсивный акроним

от «Xnu is Not Unix» — «Xnu — не Юникс»), в котором используется ядро Mach и стандартные сервисы BSD. Все возможности Unix доступны через консоль.

Поверх этой основы, в Apple разработано много проприетарных компонентов, таких как API Cocoa и Carbon, Quartz.

Mac OS X включает множество возможностей, делающих её более стабильной, чем предыдущая версия — Mac OS 9.

В Mac OS X используется вытесняющая многозадачность и защита памяти, позволяющие запускать несколько процессов, которые не могут прервать или повредить друг друга. На архитектуру Mac OS X повлияла OpenSTEP, которая была задумана как портируемая операционная система. К примеру, NeXTSTEP была портирована с оригинальной платформы 68k компьютера NeXT, до того как NeXTSTEP была куплена Apple. Так и OpenSTEP была портирована на PowerPC в рамках проекта Rhapsody.

Наиболее заметным изменением стал графический интерфейс Aqua. Использование закруглённых углов, полупрозрачных элементов и светлых полосок также повлияло и на внешний вид аппаратного обеспечения первых iMac. Некоторым пользователям это не понравилось, они считали это непрофессиональным. Другие были довольны и считали это шагом вперёд. После выхода первой версии Mac OS X другие разработчики тоже стали использовать дизайн Aqua. Для предотвращения использования своего дизайна на других платформах Apple воспользовалась услугами юристов.

Mac OS X включает среду разработки программного обеспечения Xcode, которая позволяет разрабатывать программы на нескольких языках, включая Си, C++, Objective-C, Ruby и Java. Она поддерживает компиляцию в так называемые «универсальные программы» (Universal Binary), которые могут запускаться на нескольких платформах (x86, PowerPC), так же, как «fat binaries» использовались для запуска одного приложения на 68k и PowerPC платформах.

Основами Mac OS X являются:

- Подсистема с открытым кодом — Darwin (ядро Mach, набор утилит BSD).
- Среда программирования Core Foundation (Carbon API, Cocoa API и Java API).
- Графическая среда Aqua (QuickTime, Quartz Extreme и OpenGL).
- Технологии CoreImage, CoreAudio и CoreData.

2.4 Эволюция ОС Mac OS X

Операционная система компании Apple Mac OS X на данный момент представлена в двух вариантах: версия для рабочих станций и ноутбуков, а также версия для сервера, необходимая для организации работы локальной сети предприятия. После выхода версии Mac OS X 10.0 было выпущено ещё шесть её модификаций, каждая из которых носит название животного из семейства кошачьих.

Mac OS X 10.0 Cheetah (Гепард): стала первой обновленной версией операционной системы Mac OS. Она появилась в марте 2001 года и была создана на основе ОС UNIX и сервисов FreeBSD. Новая технология позволила повысить функциональность, надежность и безопасность новой операционной системы. Но версия 10.0 Cheetah была нестабильна в работе, в ней отсутствовал ряд функций, которые есть в последних версиях системы Mac OS X, поэтому компания Apple выпустила бесплатное обновление.

Mac OS X 10.1 Puma: бесплатное обновление разработанное компанией Apple, позволяло обновить операционную систему Mac OS X 10.0 Cheetah до версии 10.1 (Puma), что дало возможность исправить ряд недоработок, существовавших в первой версии и повысить стабильность работы. В ней появились такие функции как: персонификация, высокое качество графики, быстрый доступ к системным параметрам, расширение спектра сетевых возможностей и периферийных устройств, воспроизведение и запись DVD дисков.

Mac OS X 10.2 Jaguar: в новую версию операционной системы Mac OS X было внесено более 150 видов различных изменений и дополнений, главным из которых стала поддержка аппаратного ускорения пользовательского интерфейса с применением технологии Quartz Extreme, в результате этого вся обработка изображений осуществляется процессором видеокарты. В этой версии появился мультимедийный центр QuickTime 6, интернет программа обмена мгновенными сообщениями iChat, единая адресная книга, система рукописного ввода Inkwell. Улучшены программы mail, где появилась фильтрация спама, поиск файлов на диске Finder и поиск информации в интернете Sherlock 3.

Mac OS X 10.3 Panther: появилась на рынке 24 октября 2003 года, в нее была внедрена технология шифрования данных FileVault. Последующие обновления системы привнесли еще ряд изменений таких как: исправлены некоторые проблемы безопасности, повышение надежности в работе с удаленными файлами в смешанных сетях, улучшена система файлового обмена и поддержка беспроводных сетей, обновлены драйвера к видеокартам, система каталогов и поддержка OpenGL. Внесены изменения и доработки в браузер Safari, почтовый клиент mail, адресную книгу, Stickies и QuickTime, а также доработана поддержка программного обеспечения сторонних разработчиков.

Mac OS X 10.4 Tiger: вышла в апреле 2005 года и включила в себя более 200 изменений и дополнений, которые необходимы для более быстрой и стабильной работы компьютера. Главными новинками этой версии операционной системы Mac OS X стали уникальная функция быстрого поиска информации Spotlight и интерфейс для виджетов Dashboard, приложение Automator, предназначенное для автоматизации ряда постоянно повторяющихся действий, что позволило выполнять рутинную работу с помощью одного нажатия.

Mac OS X 10.5 Leopard: появилась на рынке 26 октября 2007 года, она включает в себя более 300 улучшений и дополнений. Изменения в первую очередь коснулись ядра операционной системы, в него теперь интегрирован фреймворк динамической трассировки DTrace, а также имеется среда Cocoa Bridges для разработки приложений на Ruby и Python языках программирования, Scripting Bridge позволяет автоматизировать Mac-приложения посредством программ на Objective-C, Ruby и Python. В систему были также включены Rails, Mongrel и Capistrano. В данный момент Mac OS X 10.5 Leopard признана полноценной UNIX системой. Немаловажным событием стало появление в данной версии программы Boot Camp, с помощью которой на компьютеры Macintosh теперь можно устанавливать ОС Windows, в качестве второй рабочей операционной системы. В новой версии Mac OS X был полностью обновлен Finder и другие полезные приложения, добавлены такие функции как Stacks, Quick Look, Spaces и программа Time Machine для создания резервных копий системы и данных.

Mac OS X 10.6 Snow Leopard: была представлена в июне 2008 года и включает в себя ряд кардинальных изменений. Теперь она работает только на процессорах Intel, использует в работе 64-разрядные технологии, одинаково работает с 64-разрядными и 32-разрядными приложениями, занимает почти в два раза меньше места на диске, быстрее устанавливается, грузиться и работает. Все стандартные приложения Mac OS X (Finder, Mail, Safari, iCal и iChat и др.) в новой версии были переведены на 64-разрядный код, что значительно повысило их производительность и сделало их более устойчивыми к атакам хакеров. Диспетчер Grand Central Dispatch позволяет на полную использовать возможности многоядерных систем и увеличить производительность компьютера. Добавлено масштабирование интерфейса. ОС Snow Leopard использует технологию Bonjour для обмена файлами, стеки для работы с окнами программ, а также в ней обновлена функция Expose, которая стала еще удобней и доступней. Увеличена скорость резервного копирования, для передачи данных с целью предотвращения сбоев соединения в iChat теперь может использоваться сервер ретрансляции AIM.

28 августа 2009 вышел последний на сегодняшний день релиз 10.6 “**Snow Leopard**” (Снежный Барс). Система стоит 129\$ для новых пользователей и 29\$ для владельцев «Леопарда». Прекратив поддержку процессоров Power PC, Apple сосредоточилась на внутренних улучшениях системы вместо расширения функционала, который изменился намного меньше, чем в «Panther» и «Leopard». Пользователи в очередной раз получили ощутимое увеличение производительности. Особенно это заметно на скорости работы программы **Time Machine**. Удивительно, но новая версия ОС занимает места на диске заметно меньше, чем 10.5 (примерно на 7 гигабайт).

Серьезное внимание разработчики уделили многоядерности: система **GDC (Grand Central Dispatch)** позволяет эффективно распределять вычислительные мощности на системах с несколькими процессорами, а фреймворк **OpenCL** позволяет подключить к процессу распределения еще и графические процессоры. С гордостью подчеркивается, что система стала полностью 64-битной, в частности, под 64 бит Cocoa переписан **Finder**. Максимальное количество используемой оперативной памяти теперь равняется 16 терабайтам.

В «Snow Leopard» появились новые версии программ: **Quick Time X** (полностью переписанный, что символизирует скачок версий от седьмой сразу до десятой), **Safari 4** (который уже не является последней версией браузера после выхода 10 июня **Safari 5**) и **VoiceOver**, серьезно улучшенный по сравнению с предыдущей версией. Усовершенствованная функция извлечения дисков теперь сообщает, какое приложение или процесс использует внешний носитель, в случае отказа от его извлечения.

Система также получила полную поддержку Microsoft Exchange Server в Mail, iCal и Address Book.

2.5 Совместимость Mac OS X

Для обеспечения гладкого перехода с Mac OS 9 на Mac OS X был создан Carbon. Приложение, написанное с помощью Carbon, может запускаться на любой из этих ОС. С другой стороны, Mac OS X наследует многое из OpenSTEP, которая не является обратно-совместимой с другими версиями Mac OS. В данный момент Apple рекомендует API, именуемый Cocoa, и там наследие OpenSTEP весьма заметно — имена многих классов начинаются с «NS» (NSObject, NSArray), что является аббревиатурой от NeXTSTEP.

Также Mac OS X поддерживает Java. Это означает, что приложения, написанные на Java и использующие Swing, выглядят так же, как и приложения, использующие Cocoa. Традиционно приложения под Cocoa разрабатываются на Objective C, альтернативе Java. Однако 25 июля 2007 года Apple заявила, что дальнейшие расширения в Cocoa не будут портированы на Java. В составе Mac OS X, начиная с версии 10.5 Leopard, поставляется интерпретатор Ruby с поддержкой Cocoa.

В отличие от предшественниц, Mac OS X является полноценной, сертифицированной UNIX'03 операционной системой. Это означает, что большинство программ, написанных для BSD, Linux и других UNIX-подобных систем, скомпилируются и будут работать на Mac OS X почти или же вовсе без дополнительных изменений в коде. Для удобной установки таких программ разработаны менеджеры пакетов, такие как Fink или MacPorts (ранее — DarwinPorts). Они подобны apt в Debian или портам во FreeBSD.

Начиная с версии 10.3, Mac OS X по умолчанию также включает в себя X11.app — адаптированную версию X-сервера. Это позволяет запускать на Mac OS X приложения, разработанные для X11 — с использованием gtk, Qt для X11 (Qt4 поддерживает Mac OS X как в режиме X11, так и в обычном режиме Aqua) и прочих. Для вывода на экран X11.app использует Quartz. Однако X11.app имеет ряд проблем, таких как отсутствие Aqua-стиля в оформлении приложений и неполную поддержку Unicode.

Более ранние версии ОС могут запускать X-сервер через XDarwin (англ.).

Mac OS X основана на ядре XNU, созданном на основе микроядра Mach 3.0.

Ранние версии Mac OS X поддерживали все компьютеры Macintosh (лаптопы, десктопы или серверы) на процессорах PowerPC G3, G4 и G5. Более поздние версии перестают поддерживать старое оборудование: например, версия 10.3 Panther не поддерживает старые G3, 10.4 Tiger не поддерживает системы без портов FireWire, бета-версия 10.5 Leopard, представленная на WWDC, не поддерживала G3 вообще. Однако существуют утилиты, такие как XPostFacto, и патчи к установочному диску, сделанные сторонними разработчиками, для установки новых версий Mac OS X на оборудовании, официально не поддерживаемым Apple, включая некоторые до-G3 системы. Исключая некоторые возможности, требуемые оборудованием (такие, как графическое ускорение, запись DVD), операционная система предлагает одинаковую функциональность на всём поддерживаемом оборудовании.

Версия Mac OS X для PowerPC остаётся совместимой со старыми Mac OS приложениями через эмуляцию так называемой Classic, которая позволяет пользователям запускать Mac OS 9 как процесс в Mac OS X, поэтому многие старые приложения запускаются так, как будто под старой операционной системой. Classic не поддерживает компьютеры на процессорах Intel.

В апреле 2002 года eWeek сообщил слух о том, что у Apple есть версия Mac OS X с кодовым названием Marklar, которая запускается на процессорах Intel x86. Идея Marklar была в переходе Mac OS X на альтернативную платформу, которая позволит Apple преодолеть проблемы развития платформы PowerPC. Слухи оставались не подтверждёнными до мая 2005 года, когда в файлообменных сетях появилась версия Mac OS X для процессоров Intel.

6 июня 2005 года Стив Джобс подтвердил слухи и сообщил на WWDC, что в течение двух лет Apple перейдёт с PowerPC на процессоры Intel. До этого Apple меняла платформу с Motorola 68K на IBM/Motorola PowerPC — Apple включила эмулятор Motorola 68K в новую ОС, которая позволяла запускать большинство 68K-приложений. Apple поддерживала эмулятор в течение 11 лет; однако, при переходе на Intel, он был удалён. В новую ОС был включён эмулятор PowerPC, который называется Rosetta. Также, новая версия XCode и соответствующие консольные утилиты позволяют разрабатывать universal binaries (fat binaries) — исполняемые файлы, которые могут содержать в себе поддержку нескольких платформ (ppc, ppc64, i386, ...).

Сейчас большинство приложений, которые доступны только для PowerPC, поддерживается при помощи эмулятора Rosetta. Однако Apple просит разработчиков создавать универсальные приложения для обеих платформ. Универсальные приложения запускаются быстрее на компьютерах с процессором Intel, чем приложения для PowerPC, запускаемые с помощью эмулятора Rosetta. Некоторое программное обеспечение для PowerPC, такое, как расширения ядра и плагины системных настроек (System Preferences), не поддерживаются на компьютерах с процессорами Intel.

В то время, когда MacIntel могут запускать приложения для PowerPC, x86 и универсальные, PowerPC Маки могут запускать только универсальные и PowerPC приложения. Поддержка платформы PowerPC осталась в Mac OS X 10.5. Джобс также подтвердил слух, что новая версия Mac OS X запускалась на Intel процессорах большую часть своего существования, так как кросс-платформенность уже существовала в Mac OS X — OpenSTEP был портирован на множество платформ, включая x86, и Darwin включил поддержку для обоих (PowerPC и x86).

Mac OS X была портирована на iPhone и iPod touch. Несмотря на серьезные изменения — например, закрытость платформы и отсутствие рабочего стола (его заменяет SpringBoard) — в порте сохранились такие особенности, как Darwin, ядро XNU. В некоторых сторонних программах сохранились эффекты из настольной версии — например, в программе Converter.

Ядро Mac OS X — это компиляция POSIX, построенного на ядре XNU, со стандартными утилитами, доступными из командной строки. Это дает основание отнести

Mac OS X к Unix-системам. Ядро доступно в качестве операционной системы с открытым кодом **Darwin**. Поверх Darwin установлено множество компонентов, в том числе графический интерфейс Aqua и Finder, что и создает графическую операционную систему Mac OS X.

10-я система стала надежнее предшественницы и получила множество новых возможностей. Новая архитектура была изначально рассчитана на многозадачность и независимую обработку процессов в памяти, без прерывания или взаимного повреждения.

Архитектура Mac OS X применяет многоуровневую структуру. Подсистема Darwin отвечает за управление файловой системой и Unix-слоем прав доступа. Многие критикуют права доступа, так как возможные в них неполадки выводят из строя работу программ.

Программный интерфейс (API) OS X – прямое наследие системы OPENSTEP, разработанный совместно компанией NeXT и Sun Microsystems. Сегодня он называется Cocoa.

Для разработки приложений создана специальная среда – Xcode. Она является интерфейсом к компиляторам и нескольким языкам программирования: C, C++, Objective-C, и Java.

В системе так же есть API под названием Carbon – он был создан для плавного перехода от OS 9. Приложения, написанные в Carbon, могли работать как в 9й, так и в 10й версии системы. Ввиду своего происхождения от POSIX, в системе можно загружать многие UNIX-приложения, для чего сегодня Apple предлагает опциональный компонент X11.

В 2005 был объявлен переход от процессоров Power PC на Intel x86. Вместе с этим прекратилась поддержка эмуляции процессоров Motorola 68k. Для работы приложений написанных под Power PC в новых компьютерах была создана среда **Rosetta**. Новая версия Xcode смогла компилировать программы в формате Universal Binari для работы с обоими процессорами.

Технология Grand Central Dispatch предназначена для упрощения разработки приложений, поддерживающих использование многоядерных процессоров. Open CL позволяет использовать вычислительные мощности видеокарт для обработки приложениями других типов данных. Для удобства разработчиков в системе есть отдельные слои для упрощения написания программ: Core Audio, Core Video, Core Image, Core Animation.

2.6 Плюсы и минусы ОС Mac OS

Главной причиной небольшого процента людей, которые выбрали для работы компьютеры от компании Apple, является в первую очередь цена, а во вторую очередь закрытость Mac OS. Политика руководства компании направлена на то, чтоб операционная система Mac OS могла устанавливаться только на компьютеры их собственного производства, поэтому те, кто решил насладиться всеми преимуществами Mac OS, просто обязаны купить себе Macintosh.

Обсуждать достоинства и недостатки системы Mac OS необходимо вместе с обсуждением плюсов и минусов самих компьютеров Macintosh. Но каждое достоинство и каждый недостаток в этом случае дело абсолютно условное, поскольку тут надо учитывать с какой стороны посмотреть на ту или иную проблему. Цена главный минус компьютеров от компании Apple, поскольку они недоступны широким массам населения, и совсем не тянут на бюджетную версию.

Модельный ряд компьютеров Macintosh предложенный в магазинах довольно бедный, каждое направление представлено лишь двумя тремя экземплярами. При том все компьютеры выпускаются только в готовом виде, и тем, кто привык сам собирать себе компьютер этот вариант совсем не подойдет. Но с другой стороны придя в магазин, вам не придется долго думать какой из Macintosh стоит выбрать, при этом качество каждого из них будет на самом высоком уровне.

Да, Macintosh удовольствие дорогое, но за эти деньги вы получаете красивый, со своей особой индивидуальностью компьютер отличного качества и современную операционную систему, разработанные с учетом всех новейших технологий и достижений науки. При этом ОС Mac OS была создана именно для компьютеров Macintosh, что позволяет вам использовать возможности железа на 100 процентов, а не переплачивать деньги за новинки, которые не известно, когда и с помощью чего можно будет оценить.

К тому же в комплект с Mac OS входит набор действительно полезных программ, позволяющих организовать весь рабочий процесс современного человека. Прибавьте к этому бесплатное сервисное обслуживание в любом фирменном магазине компании Apple, и уже встает другой вопрос, на сколько этот компьютер дорогой, и стоит ли он этих денег.

Еще одной неприятной проблемой является закрытость ОС Mac OS, что в первую очередь сказывается на недостатке программного обеспечения для нее от сторонних разработчиков. До сих пор еще не существует некоторых важных программных продуктов написанных под Macintosh, да и игроманам разгуляться не получится, поскольку игры разрабатываются в первую очередь для Windows, а потом уже для Mac OS, к тому некоторых игрушек вы вообще не найдете.

Но время не стоит на месте, и появляются организации, которые занимаются разработкой программных продуктов под Mac OS, а известные разработчики программного обеспечения заинтересованы в том, чтоб их продукт работал на компьютерах Macintosh. Но самое главное компания Apple в последнюю версию ОС Mac OS включила приложение BootCamp, позволяющее с легкостью устанавливать на компьютеры Macintosh операционную систему Windows и использовать на них любое программное обеспечение.

Споры, что лучше можно продолжать до бесконечности, но если вы спросите у тех, кто решился и приобрел себе компьютер Macintosh, согласен ли он его поменять на другой, скорее всего вы получите отрицательный ответ. Те, кто работает на Macintosh – любят свои компьютеры. Объяснить это можно тем, что руководство компании Apple создает свои продукты в первую очередь для людей. Главной их стратегией является красота и удобство. К тому же все их разработки идут в ногу со временем, и даже немного его опережают. Покупая компьютер Macintosh с ОС Mac OS можно быть уверенным, что он не устареет через полгода, а будет актуален еще долгое время.

<http://ru.wikipedia.org/wiki/OSx86>.

2.7 История iOS

История мобильной операционной системы Apple началась 29 июня 2007 года. Путь ее развития был ничуть не менее простым, чем у Android. Это первые смартфоны конкурентов были действительно похожи на смартфоны, а первый iPhone скорее напоминал плеер с браузером. В самом начале не было App Store, 3G, MMC, функции копирования и вставки, многозадачности. Да много чего не было, но шесть крупных релизов и около пятидесяти небольших обновлений сделали iOS такой, как мы знаем ее сейчас.

Летом на WWDC 2013 состоится презентация iOS 7, от которой все ждут очень существенных изменений в функциональности и внешнем виде. Так что сейчас, вместо слухов о «яблочных» новинках, самое время вспомнить, какой путь прошла мобильная операционная система Apple.iPhone OS 1 — 29 июня 2007.

Первая версия мобильной операционной системы была построена на том же ядре Unix, что и Mac OS X. Но с первых минут презентации стало ясно, что отличия будут колоссальными. Каким бы инновационным ни был iPhone на момент презентации, его функциональность крайне ограничена. Мало какие знакомые сейчас функции были реализованы в первой версии операционной системы:

- основной интерфейс,

- мультитач жесты,
- iPod,
- Safari,
- карты,
- синхронизация с iTunes.

В обновлениях **iPhone OS 2 11 июля 2008** появились:

- веб–приложения на домашнем экране,
- изменение расположения иконок,
- клавиатура с поддержкой мультитач,
- iTunes Music Store.

Во второй версии операционной системы Стив Джобс признал идею использовать только веб–приложения провальной. Появление App Store стало одним из самых важных изменений за время существования мобильной операционной системы:

- — App Store,
- — Mobile Me,
- — нативные приложения от сторонних разработчиков,
- — поддержка Exchange,
- — поиск в приложении Контакты.

В обновлениях **iPhone OS 3 — 17 июня 2009** появились:

- — плейлисты iTunes Genius,
- — Google Street View,
- — скачивание подкастов.

В третью версию операционной системы было добавлено много функций, которых так сильно не хватало пользователям, например поиск по смартфону и копирование/вставка. Кроме того в версии 3.2 появилась поддержка планшетов:

- вырезать, копировать, вставить,
- голосовое управление,
- MMC,
- съемка видео,
- автофокус,
- поиск Spotlight,
- Push уведомления,
- ландшафтная ориентация клавиатуры,
- Find my phone,
- Компас,
- возможность войти в аккаунт YouTube,
- Bluetooth A2DP,
- покупки внутри приложений,
- родительский контроль,
- прямые покупки в iTunes
- HTML5 в Safari.

В обновлениях **iOS 4 — 21 июня 2010** появились:

- Genius,
- загрузка рингтонов,
- поддержка голосового управления по Bluetooth,
- поддержка разрешения экрана iPad,
- приложения для iPad,
- поддержка Bluetooth клавиатур,
- iBooks.

Летом 2010 iPhone OS была переименована в хорошо всем знакомую iOS. Наконец—то появилась многозадачность, пусть урезанная, но это стало очередным скачком в развитии:

- многозадачность,
- папки на домашнем экране,
- FaceTime,
- группировка сообщений в Mail,
- iAd,
- установка любых картинок в качестве заставки,
- возможность дарить приложения,
- поиск в приложении Сообщения.

В обновлениях **iOS 5 — 12 октября 2011** появились:

- Game Center,
- iTunes Ping,
- многозадачность на iPad,
- папки на iPad,
- AirPlay,
- AirPrint,
- iPhone в качестве точки доступа,
- iTunes Home Sharing,
- AirPlay для сторонних приложений.

Пятая версия мобильной операционки, зато смогла похвастаться виртуальным ассистентом Siri, фирменным сервисом сообщений и облаком:

- Siri,
- центр уведомлений,
- iMessage,
- iCloud
- синхронизация с iTunes по Wi-Fi
- включение новых устройств без подключения к компьютеру,
- Киоск,
- Напоминания,
- интеграция с Twitter,
- пробки и альтернативные маршруты на картах,
- поддержка RTF в почте,
- базовое редактирование снимков,
- функция Reader в Safari.

В обновлениях появились:

- измененное приложение Камера для iPad
- увеличение лимита скачивания приложений по 3G до 50 МБ,
- удаление фотографий из Фото потока.

2.8 Принципы пользовательского интерфейса iOS

Первоклассные iOS-приложения включают в себя платформу и принципы проектирования пользовательского интерфейса.

Наиболее ценными считаются iOS приложения, которые выглядят будто их создавали именно под определенное устройство. Например, если приложение четко подходит по размеру экрана устройства и отвечает на жесты, которые пользователь знает, это как раз тот опыт, который нужен пользователю. Даже если человек не знаком с принципами проектирования пользовательского интерфейса, например, такими как непосредственное управление или согласованность, он может определить, когда

приложение его «слушается», а когда нет. Когда вы начнете проектировать приложения под iOS, удостоверьтесь, что вы понимаете, что именно делают устройства с данной операционной системой особенными и понимаете, как использовать принципы высококачественного дизайна, чтобы предоставить конечному пользователю именно то, что он оценит по достоинству.

Отличный дизайн приложения начинается с четких определений.

Когда у вас появляется идея для создания приложения, важно четко определить, какие ключевые характеристики вы собираетесь представить и кому. После того, как вы с этим определитесь, вы должны продумать тот вид и впечатление от вашего приложения на том устройстве, для которого данное приложение разрабатывается и для тех задач, которое оно будет выполнять.

Если вы переносите уже существующее программное обеспечение на iOS, то вы столкнетесь с теми же сложностями. Если вы переделываете вашу уже существующую программу под iOS платформу, в данном случае вам может помочь изучение дизайнерских решений для других программ, которые были успешно переконструированы для устройств с iOS платформой, таких, например, как Mail или Keynote.

Внимание к деталям — основа успешного взаимодействия с пользователем

Очень важно максимально ориентироваться на опыт и восприятие пользователя при разработке любого аспекта вашего приложения, начиная от выполняемых задач и заканчивая, как ваше приложение начинает и завершает работу, и как используются кнопки. Определите принципы, исходя из которых вы будете создавать внешний вид и модель поведения вашего приложения в общем и частном случае.

Люди ожидают увидеть iOS технологии в приложениях, которые они используют.

iOS предоставляет массу технологий, которые добавляют больше возможностей, например, такие как многозадачность, печать и VoiceOver. Не смотря на то, что пользователи будут рассматривать данные возможности, как автоматически доступные, когда бы они не использовали устройство с iOS, это требует вашей работы над объединением данных возможностей в вашем приложении. Если iOS технологии уместны в вашем приложении, следуйте инструкциям, в которых указаны возможности их использования.

Любое приложение нуждается в индивидуальном дизайне.

Даже если ваше приложение предоставляет возможность решать серьезные вопросы и использует только стандартные элементы интерфейса, все равно вам следует предоставить красивую, кастомную иконку для приложения, радующую глаз людей, которые увидят ее в App Store и на своем Рабочем столе. Содержит ли Ваше приложение значительное количество кастомной графики или ее там немного, Вы должны понимать, какие иконки и картинки необходимы и какой вид они должны иметь. Кроме этого, создавая графику под Ретину, изучите методы, которые могли бы упростить данный процесс.

2.9 Характеристики платформы

Устройства с iOS платформой имеют некоторые общие специфические характеристики, которые влияют на опыт взаимодействия пользователя со всеми приложениями, которые работают на данных устройствах. Самые успешные приложения включают в себя все эти характеристики и предоставляют возможность пользователю задействовать свой опыт использования данных устройств.

Экран - самая важная часть, вне зависимости от его размера.

Экран на устройствах с iOS платформой - основная часть пользовательского опыта. Люди не только видят на нем красивый текст, графику и медиа, но и физически воздействуют на экран, который обладает функцией Multi-Touch, чтобы управлять интерфейсом (даже, когда они не видят экран).

Не смотря на то, что экраны с разными размерами и разрешениями могут по-разному влиять на использование пользователем приложения, тем не менее, некоторые возможности применимы ко всем устройствам с iOS платформой:

Минимальный удобный размер элементов, которые можно нажать - 44 x 44 точки

Качество графики в приложении играет значительную роль

Внимание пользователя фокусируется на содержании.

Устройства с iOS платформой имеют следующие размеры экранов:

Таблица 1-1. Размеры экранов для устройств с iOS платформой

Примечание: Пиксель - единица измерения, используемая при обсуждении размера экрана устройства или размера иконки, которую Вы создаете в редакторе изображений. Точка - единица измерения, используемая при обсуждении размера площади экрана, занимаемой графикой

На экранах со стандартным разрешением 1 точка равняется 1 пикселю, но при других разрешениях могут возникать другие соотношения. На экране с Ретиной, например, 1 точка равняется 2 пикселям.

Ориентация устройства может меняться.

Можно вращать устройство в любое время по многим причинам. Например, некоторые задачи, выполняемые пользователем, выглядят более естественно в портретном режиме, но иногда люди считают, что смогут увидеть больше в ландшафтном режиме. Какие бы не были причины, чтобы вращать устройство, люди ожидают от приложения фокусировки внимания на основном функционале.

Достаточно часто приложение открывают с «Рабочего стола», потому ожидается, что все приложения будут запускаться в одинаковой ориентации. По причине того, что iPhone и iPad по-разному могут показывать «Рабочий стол», эти ожидания могут по-разному влиять на приложения:

На iPhone и iPod touch «Рабочий стол» показывается только в одной ориентации - портретной, с кнопкой Домой внизу. Это заставляет пользователя ожидать, что приложения на iPhone будут открываться в этой ориентации по умолчанию.

На iPad «Рабочий стол» может отображаться в любой ориентации, потому пользователь ожидает, что приложение на iPad откроется в той ориентации, в которой устройство находится в данный момент.

Приложения отвечают на жесты, а не на клики.

Можно делать специфические движения пальцами, называемые жестами, чтобы управлять Multi-Touch интерфейсом устройств с iOS платформой. Например, люди нажимают кнопку, активируя его, листают или тянут длинный список либо "раздвигают пальцами", чтобы увеличить изображение.

Удобно пользоваться типовыми жестами, потому что они используются во всех встроенных приложениях. Опыт использования встроенных приложений дает людям стандартный набор жестов, которыми пользователи рассчитывают пользоваться в большинстве других приложений.

Таблица 1-2. Жесты, используемые для взаимодействия пользователя с устройствами с iOS платформой.

Название жеста	Производимое действие
Тап	Нажатие или выбор контроллера или элемента (аналог клика мышкой)
Драг	Прокручивание или сдвигание (движение со стороны в сторону)
Флик	Быстрое прокручивание или сдвигание
Свайп	В табличном представлении ряда, появляется кнопка «Удалить»
Двойной тап	Увеличивает и центрирует тот участок содержания или изображения, на которое

	"тапнули". Имеет также функцию уменьшить (если до этого было увеличено)
Движение двумя пальцами в разные стороны	Увеличение
Сведение двух пальцев	Уменьшение
Нажать и держать	В редактируемом тексте показывается увеличенное изображение для расположения курсора
Тряхнуть	Вызывает функцию отмены или восстановления

Как iPhone так и iPad, поддерживают многопальцевые жесты. Но то, что больший по размерам экран имеет больше места для дополнительных касаний, не означает, что многопальцевый метод управления всегда лучше.

Можно взаимодействовать только с одним приложением за раз.

Только одно приложение присутствует на переднем плане за один раз. Когда пользователь переключается с одного приложения на другое, предыдущее приложение исчезает вместе со своим интерфейсом.

До iOS 4 это означало, что выключаясь, приложение также удалялось из памяти. В iOS 4 и последующих версиях закрытое приложение переходило в фон, где оно продолжало или не продолжало работу. Такая возможность называется мультизадачность, она позволяет приложениям находится в фоне, пока их не запустят опять или пока не завершат работу с ними.

Примечание: мультизадачность доступна на определенных устройствах с версией iOS 4 и позднее.

Большинство приложений, находясь в фоновом режиме, входят в состояние ожидания. Приостановленные приложения отображаются в мультизадачном пользовательском интерфейсе, который предоставляет возможность быстро найти часто используемые приложения. Мультизадачный пользовательский интерфейс появляется внизу экрана под интерфейсом приложения, которое работает в настоящее время, либо же внизу «Рабочего стола».

Когда приостановленное приложение перезагружается, оно может немедленно возобновить работу с того места, где оно было остановлено.

Некоторые приложения могут продолжать работать в фоновом режиме, пока пользователь работает с другим приложением. Например, пользователь может захотеть, чтобы приложение, которое проигрывает аудио, продолжало это делать, пока он использует другое приложение, чтобы открыть календарь или проверить почту.

Параметры, которые можно настроить.

Можно настроить определенные параметры для встроенных iOS приложений. Они должны выйти из используемого приложения, чтобы поменять его настройки.

Обычно параметры в приложениях настраиваются один раз и редко меняются. Хотя некоторые встроенные приложения имеют в себе возможность менять настройки, но большинству приложений не нужна функция изменения настроек, поэтому в них и нет этой возможности.

Минимальная справочная информация на экране.

Мобильные пользователи не имеют ни времени, ни желания читать справочную информацию перед тем, как начать пользоваться приложением. Вдобавок справочная информация занимает место там, где она хранится и место на экране.

Устройства с iOS платформой и их встроенные приложения понятны и просты в использовании, поэтому не нужна справочная информация, которая бы поясняла, как пользоваться устройством или приложением.

У приложения одно окно.

iOS приложения, вне зависимости от типа, имеют одно активное окно. Программным путём окно обеспечивает основу, на которую выкладывается содержание и функциональность приложения. Пользователь ожидает от этого окна не тех же возможностей, которые он ждет от окон в компьютерных приложениях. Пользователь устройства с iOS платформой видит приложение, как коллекцию экранов, большинство которых появляются последовательно.

Можно рассматривать экраны приложения, как целостные визуальные структуры или формы. У приложения может быть несколько экранов и каждый из них может содержать различные комбинации изображений и контроллеров.

Пользователи воспринимают экран приложения и экран устройства как тождественные, но содержимое экрана приложения может простирается далеко за границы экрана устройства. Например, в приложении «Контакты» на iPhone, список контактов показывается на одном экране, хотя этот список возможно содержит достаточно имен, чтобы заполнить экран устройства несколько раз полностью.

Два типа программ для iOS.

Существует два типа программного обеспечения, которые Вы можете разрабатывать под устройства с iOS платформой:

- iOS приложение;
- Веб-содержимое.

iOS приложение - это приложение, которое Вы разрабатываете, используя iOS SDK, чтобы оно могло работать в исходном формате на iOS устройстве. iOS приложения сходны с встроенными приложениями iOS устройства, в том смысле, что они находятся в устройстве и используют ключевые характеристики iOS среды. Можно устанавливать iOS приложения на свои устройства и использовать их так же, как и встроенные приложения, такие как Photos, Calendar или Mail.

Веб-содержимое размещается на вебсайте, который посещают посредством iOS устройства. Веб-содержимое, которое появляется на iOS устройстве, делится на три категории:

- Веб-страницы, которые обеспечивают целенаправленное решение задачи и соответствуют определенному виду вывода данных на экран, так же известны, как веб-приложения, потому что они ведут себя сходно с iOS приложениями. Веб-приложение часто прячет интерфейс Safari на iOS, потому что выглядит похожим на родное приложение. Веб-приложение также предоставляет иконку для «Рабочего стола». Это позволяет открывать веб-приложение так же, как и iOS приложения.
- Оптимизированные веб-страницы. Веб-страницы, которые оптимизированы под Safari для iOS, отображаются и работают так, как спроектированы (за исключением любых элементов, которые основываются на технологиях, которые не поддерживаются iOS, такие как плагины, Flash и Java). Можно еще добавить, что оптимизированные веб-страницы масштабируют контент под размеры экрана и задуманы так, что могут определить, когда этот контент просматривается на экране iOS устройства, чтобы выровнять его соответствующим образом.
- Совместимые веб-страницы. Веб-страницы совместимые с Safari под iOS отображаются и работают так, как спроектированы (за исключением любых элементов, которые основываются на технологиях, которые не поддерживаются iOS, такие как плагины, Flash и Java). Совместимые веб-страницы не оптимизируют контент под iOS устройства, но чаще всего устройство выводит просматриваемую страницу без ошибок.

iOS приложение может совмещать в себе "родные" элементы интерфейса со взятыми из веба. Такое приложение может выглядеть и вести себя как "родное" iOS приложение, без привлечения внимания к тому факту, что оно зависит от веб-ресурсов.

Safari на iOS поддерживает интерфейс для загрузки веб-контента на iOS устройства. Хотя Safari под iOS во многом похож на Safari для компьютера, тем не менее, это не одно и то же.

По большей части, пользователь не может изменить размер окна просмотра (площадь, на которой размещается контент). На экране компьютера пользователь может изменять размер окна, изменяя размер окна браузера. На iOS устройствах размер окна просмотра не изменяется, кроме варианта изменения ориентации устройства. iOS пользователь может менять масштаб окна, приближая или отдаляя веб-контент, или может двигать веб-страницу.

Safari на iOS поддерживает cookie-файлы. Использование cookie-файлов ускоряет работу пользователя с веб-контентом, сохраняя контекст, настройки и прежде введенные данные.

Safari под iOS не поддерживает Flash, Java (включая Java приложения) или подключаемые программы сторонних поставщиков в веб-контенте. Взамен этого Safari под iOS поддерживает <audio> и <video> теги HTML5, обеспечивая работу аудио и видео потоков, так же JavaScript и CSS3 трансформацию, переходы и анимирование для отображение анимированного контента.

Safari под iOS интерпретирует большинство жестов как способ устройства продемонстрировать контент, но не контент как таковой. С помощью тапа, аналога клика мышкой, Safari под iOS отправляет событие на веб-страницу. Больше аналогов жестов, например, таких, как наведение курсора, схожих с поведением мышки нет.

Safari под iOS позволяет приложениям работать в полноэкранном режиме. Веб-приложения, которые запускаются из ярлыка, находящегося на «Рабочем столе», могут скрывать пользовательский интерфейс Safari, так что они выглядят больше как родные приложения.

Принципы разработки пользовательского интерфейса.

Пользовательский интерфейс получится качественным, если следовать рекомендациям по разработке пользовательского интерфейса, которые базируются на изучении работы и мышления человека-пользователя, но не на возможностях самого устройства. Использование даже самого отличного приложения будет казаться каторгой, если у него будет непривлекательный, запутанный или нелогичный интерфейс. А красивый, интуитивно понятный и привлекательный интерфейс может улучшить функциональность приложения и вызвать позитивный эмоциональный настрой у пользователей.

Эстетическая целостность - не мера того, насколько красиво выглядит приложение. Это показатель того, насколько внешний вид приложения соотносится с его функциональностью. Например, приложение, позволяющее продуктивно решать определенные задачи, содержит декоративные элементы, которые незаметны и находятся в фоне, пока решаемые задачи находятся на переднем плане, обеспечивая работу с ними стандартными контроллерами и функционалом. Такие приложения дают пользователю понятную и целостную информацию о своем назначении и сущности. Но, если же приложение позволяет продуктивно решать задачи, но вместе с тем пользовательский интерфейс выглядит эксцентрично или непродуманно, люди могут не понять, как интерпретировать данные противоречивые послания.

Сходная ситуация и с приложениями с эффектом присутствия таких, например, как игры - пользователь ожидает увидеть красивый внешний вид приложения, который обещает удовольствие от использования и поощряет открытия. Хотя пользователи и не планируют выполнять какие-то серьезные или полезные задачи в игре, тем не менее, они ждут, что во внешнем интерфейсе будет задействован тот опыт управления, к которому они привыкли.

Непосредственное управление.

Когда пользователи непосредственно управляют элементами на экране, вместо того, чтобы использовать отдельные контроллеры, то им удобнее выполнять нужную задачу и понимать, что будет результатом их действий. iOS пользователь может насладиться повышенным ощущением непосредственного управления благодаря Multi-Touch интерфейсу. Использование жестов позволяет людям ощутить более непосредственную связь, чувство контроля над объектами, которыми они управляют на экране, потому что люди могут дотронуться до них, не используя посредника, например, такого, как мышка.

Например, вместо нажатия контроллера отвечающего за увеличение, человек может использовать жесты щипка, чтобы целенаправленно увеличить или уменьшить определенную область контента. В играх игрок двигается и взаимодействует непосредственно с объектами на экране. Например, в игре может присутствовать замок с цифровым кодом, который пользователь может покрутить, чтобы открыть.

В iOS приложениях люди могут использовать непосредственное управление, в следующих случаях:

- Вращая или по-другому манипулировать устройством, чтобы повлиять на объекты на экране.
- Используя жесты, чтобы манипулировать объектами на экране.
- Видят, что их действия имеют мгновенный и заметный результат.

Обратная связь.

Обратная связь подтверждает действия пользователя и подтверждает, что процесс идет. Пользователи ожидают немедленной ответной реакции, когда они выполняют ряд действий, и они ценят присутствие видимости обновлений во время выполнения соответствующих действий.

Встроенные iOS приложения отвечают на любое действие пользователя какими-то заметными изменениями. Например, список объектов, когда пользователь нажимает на него, слегка подсвечивается. Во время операций, которые длятся более нескольких секунд, контроллер показывает течение процесса, и, если уместно, приложение может показать пояснительное сообщение.

Лёгкая анимация дает людям понятную обратную связь, что делает ясными результаты их действий. Например, в списках, добавление еще одной строки может показываться в виде анимации, что более наглядно показывает внесенные изменения.

Звуковые сигналы так же могут давать обратную связь. Но он не должен быть основным или единственным механизмом, потому что люди могут пользоваться своими устройствами в тех местах, где он не будет слышен или же отключен по необходимости.

iOS предоставляет широкие возможности для использования метафор благодаря тому, что поддерживает разнообразные графические элементы и жесты. Люди физически взаимодействуют с реалистичными объектами на экране, в большинстве случаев поступая с ними так, как если бы они были объектами реального мира. Аналогии в iOS в себя включают:

- Нажатие кнопки воспроизведения на iPod.
- Перетаскивание, листание и свайпинг объектов в игре.
- Перемещение бегунка для действия включить/выключить.
- Листание страниц с фотографиями.
- Вращение сортировочного колеса, чтобы сделать выбор.

т.е., что есть возможность корректно остановить операцию на стадии выполнения.

Стратегии разработки дизайна приложения.

Все великие приложения начинаются с отличной идеи, но это не значит, что путь от идеи до успешного приложения iOS будет легким. Опишем некоторые стратегии, используемые для уточнения вашей идеи, пересмотра ваших вариантов дизайна, и ведут к приложению, которое люди будут ценить.

Основная задача приложения - это краткое и конкретное объяснение основной цели приложения и его целевой аудитории.

Сформулируйте задачи приложения в ранних стадиях разработки, чтобы воплотить идею и список функций в цельный, гармоничный продукт, который люди захотят использовать. В процессе разработки используйте эту формулировку, чтобы решить, нужны ли вообще запланированные функции. Для того, чтобы создать вразумительную формулировку, произведите следующие шаги.

- Перечислите все функции, которые, могут понравиться пользователям.
- Определите кто будет пользователем.
- Отфильтруйте список функций, держа в уме целевую аудиторию.
- Не останавливайтесь на достигнутом.
- Разрабатывайте под конкретные устройства.
- Принимайте к сведению парадигмы интерфейса iOS.

Пользователи iOS привыкли к виду и поведению включённых в оболочку приложений, поэтому ожидают подобного и от сторонних программ. Не нужно слепо копировать каждую деталь стандартных программ, но понимать логику их поведения очень желательно. Попробуйте разобраться, что же выделяет устройства на iOS и программы для них. Так же стоит помнить:

- Элементы управления должны выглядеть нажимаемыми. В iOS у кнопок, инструментов выбора и ползунков контуры как бы приглашают нажать на них.
- Структура программы должна быть чёткая и перемещения по ней должны быть легки. iOS предоставляет навигационную полосу для прохождения через упорядоченный контент и панель вкладок для отображения блоков элементов, подобных по назначению или содержанию.
- Обратная связь с пользователем должна быть ненавязчивой, но ясной. Программы iOS часто делают точные, притом плавные анимации для демонстрации пользователям их действий. Так же они могут использовать индикаторы активности и прогресса для отображения статуса и предупреждения для оповещения о наиболее важной информации.

Убедитесь, что универсальная программа одинаково хорошо работает и на iPhone и на iPad. Если вы планируете разработать программу, которая будет работать на iPhone и на iPad, вам нужно адаптировать дизайн программы для обоих устройств.

Вот некоторые рекомендации, которые помогут вам сделать это:

- Формируйте пользовательский интерфейс программы для каждого устройства, на котором она сможет работать.
- Адаптируйте оформление программы к размеру экрана.
- Сохраните основную функциональность вашей программы независимо от того, на каком устройстве она работает.
- Превысите ожидания. Немодифицированные программы для iPhone по умолчанию работают на iPad в режиме совместимости.
- Пересмотрите дизайн веб-приложений.

Вот методика, способная помочь веб-разработчикам создавать программы для iOS:

- Сосредоточьтесь на своей программе. Пользователи iOS ожидают, что программа сделает, то что было обещано, и они хотят увидеть полезную информацию немедленно.
- Убедитесь, что программа позволяет пользователю что-то сделать. Люди могут получать удовольствие от просмотра содержимого веб-сайта, но они хотят что-то сделать в программе.
- Проектируйте использование сенсорного управления. Не пытайтесь повторить основы пользовательского интерфейса веб-дизайна в ваших

программах для iOS. Вместо этого, познакомьтесь с элементами пользовательского интерфейса и моделями iOS и используйте их для демонстрации вашей информации.

- Позвольте пользователям листать информацию с помощью прокрутки. На iOS-устройствах, использование прокрутки - легкая, ожидаемая часть взаимодействия.
- Переместите иконку главной страницы. iOS программы позволяют пользователю нажать строку состояния, чтобы быстро пролистать информацию назад в начало.

Настройка специального интерфейса задания.

В лучших iOS программах специальный интерфейс находится в равновесии между ясностью назначения и простотой использования. Для достижения этого баланса в вашем приложении, убедитесь, что вы обдумали особенность интерфейса в начале процесса проектирования. Потому что беспокойство по поводу узнаваемости, оригинальности и пригодности для продаж часто влияет на решения по поводу специальности интерфейса, это может заставить сосредоточиться на том, как специальный интерфейс повлияет на взаимодействие с пользователем.

Используя iOS SDK, вы можете настроить специальный интерфейс вашей программы, как захотите, сильно или слегка. Нет никаких практических ограничений на глубину изменений интерфейса, которые вы можете сделать, вам нужно лишь определить, как специальный интерфейс может повлиять на задачи, выполняемые вашей программой. Когда вы думаете об этих задачах, подумайте о том, как часто пользователи будут выполнять их и при каких обстоятельствах.

Всегда есть повод для использования специального интерфейса. В идеале, специальный интерфейс упрощает выполнение задачи, которую хочет выполнить пользователь и повышает его опыт. Вам нужно, насколько это возможно, позволить вашей программе управлять вашими решениями по поводу создания специальных интерфейсов.

Например:

- Если ваша программа позволяет выполнять задачу, которая включает в себя манипулирование большим объемом подробной информации, пользователи, скорее всего, оценят простой, в основном, стандартный интерфейс и рациональную навигацию.
- Если ваша программа позволяет пользователям просматривать какую-то информацию, они, как правило, не воспринимают пользовательский интерфейс, который отличается от стандартного.
- Если ваша программа - игра или предлагает пользователю захватывающее, с насыщенным сюжетом взаимодействие, пользователи ожидают, что попадут в уникальный мир, наполненный богатой, красивой графикой и инновационным взаимодействием.

Насколько возможно, избегайте увеличения когнитивной нагрузки (повышения умственного восприятия и переработки внешней информации) на пользователя. Пользователи знакомы с внешним видом и поведением стандартных элементов пользовательского интерфейса, поэтому они не должны останавливаться и думать о том, как использовать эти элементы. Когда они сталкиваются с элементами, которые выглядят или ведут себя не так как стандартные элементы, то пользователи теряют преимущество своего предыдущего опыта. Если ваши уникальные элементы не делают выполнение задачи проще, пользователям может не понравиться освоение новых элементов интерфейса, и склонить их к переходу на другое программное обеспечение.

Будьте внутренне непротиворечивы. Чем больше специфичен ваш пользовательский интерфейс, тем более важна для внешнего вида и поведения ваших специальных элементов интерфейса их последовательность и единообразие в вашей программе. Если пользователи тратят время на то, чтобы узнать, как использовать

незнакомые им элементы управления, которые вы создали, они хотят опираться на эти знания на протяжении всей вашей программы.

Всегда считайтесь с информационным содержанием. Поскольку стандартные элементы хорошо знакомы, они не конкурируют с содержанием за внимание человека. Если вы используете специальный пользовательский интерфейс, позаботиться о том, чтобы он не заслонял информационное содержание людям. Например, если ваше приложение позволяет пользователям смотреть видео, вы можете использовать специальные элементы управления воспроизведением. Но используете ли вы специальные или стандартные элементы управления воспроизведением, это менее важно, чем то, что элементы управления будут исчезать после того, как пользователь начинает просмотр видео и появляться снова поле касания экрана.

3 Разработка приложений под Mac OS X и iOS

3.1 Переменные

Учась водить машину, вам приходится держать в голове сразу несколько вещей. Нужно одновременно помнить о педалях сцепления, газа и тормоза. Программирование тоже требует от вас внимания ко многим деталям, чтобы программа работала без сбоев. Когда вы начинаете учиться водить машину, вы уже так или иначе знакомы с ней. Когда же вы начинаете учиться программировать на Xcode, этого преимущества вы лишены. Поэтому, мы отложим разбор среды программирования до следующих разделов. Сначала мы постараемся познакомить вас с кодом Objective-C, начиная со очень знакомых вам некоторых основ математики [7].

В Objective-C тоже используются переменные. Переменные являются не более чем удобными именами для обозначения конкретных фрагментов данных, таких как числа. Ниже написано высказывание на Objective-C, а именно строка кода, где переменной присваивается определённое значение.

```
//(1)  
x=4;
```

Переменная *x* принимает значение 4. Следует заменить точку с запятой в конце строки. Точка с запятой нужна в конце каждой команды. Почему? Ну, код, приведенный в примере (1), может выглядеть понятным вам, но компьютер не знает что с ним делать. Специальная программа «компилятор» призвана перевести текст, который вы пишете, в единички и нолики, которые понимает ваш Mac. Компьютеру очень сложно понять введенный текст, например, где кончается команда. Для этого и используется точка с запятой.

Если вы забудете поставить точку с запятой в вашем коде, код не скомпилируется, т.е. не переведется в код, который сможет выполнить Mac. Не сильно беспокойтесь об этом, компилятор пожалуется, если не сможет скомпилировать код. Как мы увидим в следующей главе, он будет стараться помочь вам понять, что случилось.

Хотя имена переменных и не имеют особого значения для компилятора, описательные имена переменных могут сделать программу гораздо проще для чтения и следовательно понимания людьми. Это большой плюс при отслеживании ошибок в коде.

Ошибки в программах по традиции называют «багами» (англ. bugs). Их поиск и исправление называют дебагингом.

Таким образом, в реальном коде следует избегать использования не описательных имен переменных вроде *x*. Например, переменную для ширины изображения можно было бы назвать `pictureWidth` (2).

```
//(2) pictureWidth = 8;
```

Компиляторы обычно прощают отсутствие точки с запятой, но вы поймете, что программирование – это в основном детали. Например, нужно обращать внимание на заглавные и прописные буквы в названиях функций и переменных (case-sensitive). Например: `pictureWidth`, `pictureWIDTH` и `PictureWidth` – это три разные переменные. Согласно общим соглашениям, мы делаем переменные, соединяя несколько слов, первое слово без заглавных и все остальные начинаются с заглавных, как на примере (2). Этот стиль часто называют `camelCase`. Выбрав этот метод, вы существенно сократите количество ошибок программирования, связанных с регистром.

Помните, что имя переменной обычно состоит из одного слова (или одного символа в крайнем случае).

Несмотря на то, что вы можете использовать любое имя для своих переменных,

существуют определенные правила, которые необходимо соблюдать. Не буду пока перечислять их всех сразу, скажу лишь про одно, оно основное — переменные не должны совпадать с зарезервированными словами Objective-C. Используя переменные, состоящие из нескольких слов, например `pictureWidth`, вы никогда не нарушите это правило. И еще, старайтесь использовать строчные буквы в именах переменных, это обеспечит их читабельность и облегчит вам жизнь в дальнейшем. Если вам интересно узнать несколько правил, дочитайте это параграф. Кроме букв, можно пользоваться числами, но переменные не должны начинаться с них. Кроме этого допустим символ подчеркивания «_».

Приведём несколько примеров названий переменных.

Правильные имена переменных:

`doork`

`door8`

`door`

Недопустимые:

`door 8` (содежит пробел) `8door` (начинается с цифры)

Не рекомендуется:

`Door8` (начинается с заглавной буквы)

Теперь, когда мы знаем, как присваивать переменным значения, мы можем осуществлять расчёты. Давайте посмотрим на код (3), осуществляющий расчет площади поверхности картинки.

```
//(3) pictureWidth=8; pictureHeight=6;
pictureSurfaceArea=pictureWidth*pictureHeight;
```

Удивительно, компилятор не обращает внимание на пробелы (за исключением имен переменных, команд и т.д.) Для большей читаемости кода вы можете использовать пробелы.

```
//(4) pictureWidth = 8;
pictureHeight = 6;
pictureSurfaceArea = pictureWidth * pictureHeight;
```

3.2 Целые и дробные числа

Теперь взглянем на пример (5), в частности на первые две строки (7).

```
//(5) pictureWidth = 8;
pictureHeight = 4.5;
pictureSurfaceArea = pictureWidth * pictureHeight;
```

Числа можно разделить на два вида: целые и дробные. В выражениях (6.2) и (6.3) представлены оба вида. Целые используются для подсчета, например, когда вам нужно сделать несколько повторений одной и той же инструкции. Дробные или числа с плавающей точкой используются, например, в вычислениях среднего числа попаданий в бейсболе.

Код из примера (5) не будет работать. Проблема в том, что компилятору нужно

знать заранее, какие именно переменные вы будете использовать в своей программе и на какие типы данных они ссылаются, то есть целые числа или дробные с плавающей запятой. Это называется «объявить переменные».

```
//(6)
int pictureWidth;
float pictureHeight, pictureSurfaceArea; pictureWidth = 8;
pictureHeight = 4.5;
pictureSurfaceArea = pictureWidth * pictureHeight;
```

В строке (6.2) `int` означает, что переменная `pictureWidth` целое число (integer). В следующей строке, мы объявляем сразу две переменные, это можно сделать при помощи запятой. Как вы можете видеть, обе переменные типа `float`, то есть могут хранить в них дробные числа. На самом деле, немного глупо то, что переменная `pictureWidth` отличается от других двух, потому что если вы умножите переменную типа `int` на переменную типа `float` результат расчетов будет `float`. Вот почему вам стоит объявить `pictureSurfaceArea` как `float` (6.3).

Почему компилятор должен знать, представляет ли переменная собой целое число или число с дробной частью?

Хорошо, компьютерным программам нужна часть компьютерной памяти. Компилятор резервирует место в памяти (байты) для каждой переменной в программе. Так как различные типы данных, в данном случае целые `int` и дробные `float`, требуют разные объемы памяти и имеют различные представления, компилятор должен зарезервировать правильный объем памяти и использовать верные представления о типе данных.

Что, если мы работаем с очень большими числами или десятичными числами очень высокой точности? И они не будут помещаться в несколько байтов, которые выделит для них компилятор, что с ними станет? Есть два ответа на этот вопрос: во-первых, числа как целые так и с плавающей точкой, имеют аналоги, которые могут хранить большие числа (или числа более высокой точности). В большинстве систем это `long` и `double`, соответственно. Но даже они могут переполняться, что приводит нас ко второму ответу: это будет ваша (как программиста) работа — отслеживать ошибки. В любом случае, это не та проблема, которая будет обсуждаться в первой главе.

Кстати, целые десятичные числа могут быть отрицательным, как вы знаете, например ваш банковский счет. Если вы знаете, что значение переменной никогда не бывает отрицательным, можно увеличить диапазон значений, которые вписываются в число доступных байтов.

```
//(7)
unsigned int chocolateBarsInStock;
```

Увы, нет такого понятия, как отрицательное число шоколадок на складе, так что беззнаковые целые `unsigned int` могли бы быть использованы здесь. Тип `unsigned int` представляет собой целые числа большие или равные нулю.

Есть возможность объявить и инициализировать переменную за один шаг.

```
//(8)
int x = 10;
float y = 3.5, z = 42;
```

Это позволит вам несколько сэкономить на написании кода.

3.3 Типы данных

Как мы только что увидели, данные, хранящиеся в переменной, могут быть одним из нескольких конкретных типов, для примера целые и дробные числа.

В Objective-C, такие простые типы данных как эти, также известны как скалярные типы данных. В таблице 3.1 приведен список общих скалярных типов данных доступных в Objective-C:

Таблица 3.1 Список общих скалярных типов

Имя	Тип	Пример
<code>void</code>	пусто	ничего
<code>int</code>	целое число	-1, 0, 1, 2
<code>unsigned</code>	беззнаковое целое число	0, 1, 2
<code>float</code>	числа с плавающей точкой	-0.333, 0.5, 1.23
<code>double</code>	числа с плавающей точкой удвоенной точности	0.5252525233323409
<code>char</code>	символ	a, b, c
<code>BOOL</code>	логический	0, 1, TRUE, FALSE, YES, NO

3.4 Математические операции

В предыдущем примере, мы выполнили операцию умножения. Используйте следующие символы, называемые операторами, для выполнения простых математических расчетов.

+ для сложения
- для вычитания
/ для деления
* для умножения

Используя операторы, мы можем выполнять широкий спектр расчетов. Если вы посмотрите на код профессиональных Objective-C программистов, вы найдете пару особенностей, возможно, потому что они ленивы.

Вместо записи вида `x = x + 1`; программисты часто прибегают к чему-то другому, как (9) или (10).

```
//(9) x++;  
//(10)  
++x;
```

В любом случае это означает: увеличение `x` на одну единицу. В некоторых случаях важно, используется `++` до или после имени переменной. Обратите внимание на примеры (11) и (12).

```
//(11) x = 10;  
y = 2 * (x++);  
//(12) x = 10;  
y = 2 * (++x);
```

В примере (11), после его выполнения, `y = 20`, а `x = 11`. В отличие от этого, в

примере (12.2), увеличение x на единицу происходит до умножения на 2. Так, в конце концов получается, что $x = 11$, а $y = 22$. Код примера (12) эквивалентен примеру (13).

```
//(13) x = 10; x++;  
y = 2 * x;
```

Итак, программист фактически объединил два выражения в одно. Лично я считаю, что это делает программу трудно читаемой. Если вы экономите время и силы, то это прекрасно, но имейте ввиду, то что там может скрываться ошибка.

Это будет банально, если вам удалось пройти среднюю школу, но скобки могут использоваться для определения порядка, выполнения операций. Обычно $*$ и $/$ имеют больший приоритет над $+$ и $-$. Поэтому $2 * 3 + 4 = 10$. А с помощью скобок, вы можете заставить это скромное сложение выполняться первым: $2 * (3 + 4) = 14$.

Оператор деления заслуживает особого внимания, потому что он выполняет совершенно разные действия в зависимости от того, работает ли он с целыми числами или вещественными. Взгляните на следующие примеры (14, 15).

```
//(14)  
int x = 5, y = 12, ratio; ratio = y / x;  
//(15)  
float x = 5, y = 12, ratio; ratio = y / x;
```

В первом случае (14), результатом будет 2. И только во втором случае (15), результатом будет именно то, чего вы, вероятно, ожидали: 2.4.

3.5 Логические операнды

Булевый тип является простым логическим представлением значений true или false. 1 (YES) и 0 (NO) отождествляемые с true и false значениями часто используются как взаимозаменяемые, и их можно считать эквивалентными:

Таблица 3.2 Булевый тип

T	FA
1	0
Y	NO

Их часто используют, когда нужно принять решение о выполнении некоторого действия в зависимости от логического значения переменной или результата работы функции.

3.6 Остаток от деления

Оператор, с которым вы, вероятно, знакомы $\%$ (остаток от деления). Он работает не так, как вы могли бы ожидать: оператор остатка от деления не используется для вычислений с процентами. Результатом оператора $\%$ будет остаток от целочисленного деления первого операнда на второй (если значение второго операнда равно нулю, то поведение оператора $\%$ не определено).

```
//(16)  
int x = 13, y = 5, remainder; remainder = x % y;
```

Теперь значение remainder равно 3, потому что x равняется $2*y + 3$. Вот несколько примеров на остаток от деления:

```
% 7 равно 0
% 7 равно 1
% 7 равно 2
% 7 равно 3
27 % 7 равно 6
% 2 равно 0
% 2 равно 1
% 2 равно 0
% 2 равно 1
% 2 равно 0
50 % 9 равно 5
60 % 29 равно 2
```

Иногда он действительно может очень пригодиться, но учтите, что работает он только с целочисленными переменными.

Часто этот оператор применяют, чтобы выяснить является ли число четным. Если оно четное, остаток от деления на два будет нулевой. Или не нулевой, в противном случае. Вот пример:

```
/(17) int anInt;
//Некий код, меняющий значение переменной anInt if ((anInt % 2) == 0)
{
    NSLog(@"Чётное");
}
else
{
    NSLog(@"Нечётное");
}
```

3.7 Комментарии

Используя «говорящие» имена переменных, мы можем сделать наш код более читабельным и понятным (18).

```
/(18)
float pictureWidth, pictureHeight, pictureSurfaceArea; pictureWidth = 8.0;
pictureHeight = 4.5;
pictureSurfaceArea = pictureWidth * pictureHeight;
```

До сих пор наши примеры кода содержали лишь несколько выражений, но даже очень простая программа может быстро вырасти до сотен или тысяч строк. Пересматривая свой код спустя несколько недель или месяцев, вам, возможно, будет трудно вспомнить, почему вы поступили именно так. Здесь на помощь приходят комментарии. Комментарии помогут вам быстро понять свой код, какие части что делают, почему и в каком порядке. Некоторые программисты идут так далеко, что пишут комментарии к классам до начала кодирования. Это помогает им организовать собственное мышление и избегать ненужных действий.

Рекомендуется тратить какое-то время на комментирование своего кода. Мы можем вас уверить, вы сэкономите значительно больше времени из-за этого в будущем. Также, если вы отдадите ваш код кому-нибудь, ваши комментарии помогут ему быстрее адаптировать его к собственным нуждам.

Чтобы создать комментарий, просто начните строку с двух символов косой черты.

```
// Это комментарий
```

В Xcode комментарии отображаются зеленым цветом. Если комментарий длинный и многострочный, поместите его между `/*` и `*/`.

```
/* Это комментарий из двух строк */
```

Мы обсудим отладку программ в ближайшее время, Xcode имеет большие возможности для этого. Один из старинных способов отладки называется *outcommenting*. Разместив часть кода между `/*` `*/`, вы можете временно отключать часть кода, чтобы проверить, работает ли остальная часть кода как и ожидалось. Это позволяет отловить ошибку. Если *outcommented* часть должна присвоить, например, значение конкретной переменной, вы можете включить временное выражение, где вы установите значение переменной пригодное для тестирования оставшейся части вашего кода.

Для чего нужны комментарии? Важность комментариев трудно переоценить. Часто бывает полезно добавить пояснение (на простом человеческом языке) о том, что происходит в длинной последовательности выражений. Это позволит не ковыряться в коде, пытаясь понять что он делает, а сразу понять — является ли он частью проблемы, которую вы сейчас устраняете. Вы должны также использовать комментарии, чтобы выразить вещи, которые трудно или невозможно понять из кода. Например, если программа выполняет математические функции с помощью конкретной модели, подробно описанной в книге, вы должны поставить библиографические ссылки в комментарии, связанные с вашим кодом.

Иногда бывает полезно записать несколько комментариев перед написанием кода. Это поможет вам структурировать свои мысли и, как результат, программировать будет проще.

Код примеров в этой книге не содержит так много комментариев, как мы обычно используем, потому что они уже содержатся в пояснениях.

3.8 Функции

Самый длинный кусок кода, который мы видели до сих пор, включал всего 5 операторов. Программы из многих тысяч строк могут казаться излишне многословными, поэтому мы должны обсудить принципы и алгоритмы работы Objective-C на самых ранних стадиях.

Если бы программа состояла из длинной, непрерывной последовательности утверждений, то было бы трудно найти и исправить ошибки.

Кроме того, некоторые последовательности операторов могут появляться в вашей программе в нескольких местах. Если есть ошибка в последовательности операторов, то ее нужно исправить в нескольких местах. Ужас: очень легко забыть одну (или две)! Итак, люди придумали способ писать код, который упрощает исправление ошибок.

Решение этой проблемы заключается в том, что последовательность операторов составляет функцию. Например, вы можете написать функцию, которая позволяет рассчитать площадь круга. После того как вы проверили, что этот код работает надежно, вам никогда не придется проверять, есть ли в нем ошибки. Набор операторов, называемый функцией, имеет название, и вы можете вызывать его для выполнения кода. Это понятие использования функций настолько глобально, что всегда есть по крайней мере одна функция в программе: `main()`. Благодаря ей компилятор знает, где должно начаться выполнение кода после запуска.

Давайте взглянем на функцию `main()` подробнее (19).

```

//(19)
main()
{
// Тело функции main(). Добавьте свой код сюда.
}

```

Инструкция (19.2) показывает название функции, т.е. `main`, завершенное открытием и закрытием скобок. В то время как `main` является зарезервированным словом, и присутствие функции `main()` является обязательным, если вы напишете свои собственные функции, то можете назвать их так, как вам нравится. Постановка там скобок — хорошая вещь, но мы не будем обсуждать это до конца главы. На строках (19.3, 19.5) есть фигурные скобки. Мы должны писать наш код между этими фигурными скобками `{}`. Все, что находится между ними, называется телом функции. Я взял код из первой главы и вставил его здесь(20).

```

//(20)
main()
{
// Ниже объявляются переменные
float pictureWidth, pictureHeight, pictureSurfaceArea;
// Мы инициализируем переменные (устанавливаем их значения) pictureWidth =
8.0;
pictureHeight = 4.5;
// Здесь производится расчет
pictureSurfaceArea = pictureWidth * pictureHeight;
}

```

Если и дальше добавлять код в тело функции `main()`, то в итоге программу будет сложно отлаживать. Хотелось бы избежать неструктурированного кода. Давайте напишем другую программу с иной структурой. Помимо обязательной функции `main()` мы создадим функцию `circleArea()` (21).

```

//(21)
main()
{
float pictureWidth, pictureHeight, pictureSurfaceArea; pictureWidth = 8.0;
pictureHeight = 4.5;
pictureSurfaceArea = pictureWidth * pictureHeight;
}

circleArea() // (21.10)
{

}

```

Это просто, но наша пользовательская функция (21.10) пока ничего не делает. Заметим, что описание функции находится вне тела функции `main()`. Иначе — функции не вложенные.

Наша новая функция `circleArea()` должна вызываться из функции `main()`. Посмотрим, как это сделано (22).

```

//(22)
main()
{
float pictureWidth, pictureHeight, pictureSurfaceArea, circleRadius, circleSurfaceArea; //
(22.5)
pictureWidth = 8.0;
pictureHeight = 4.5;
circleRadius = 6.0; // (22.8)
pictureSurfaceArea = pictureWidth * pictureHeight;
// Здесь мы вызываем нашу функцию!
circleSurfaceArea = circleArea(circleRadius); // (22.11)
}

```

Примечание: остальная часть программы не показана (см. (21)).

Мы добавили пару вещественных переменных (22.5) и инициализировали переменную `circleRadius`, т.е. установили ее значение (22.8). Наиболее интересна строка (22.11), где вызывается функция `circleArea()`. Вы можете видеть, что название переменной `circleRadius` было помещено между круглыми скобками. Это аргумент функции `circleArea()`. Значение переменной `circleRadius` будет передано функции `circleArea()`. Когда функция `circleArea()` закончит вычисления, она должна вернуть результат. Давайте изменим функцию `circleArea()` (21), чтобы видеть это (23).

Примечание: показана только функция `circleArea()`.

```

//(23)
circleArea(float theRadius) // (23.2)
{
float theArea;
theArea = 3.14159 * theRadius * theRadius; //  $S = \pi R^2$  (23.5)
return theArea;
}

```

В (23.2) мы определяем, что для функции `circleArea()` требуется входное значение вещественного типа. После получения оно хранится в переменной с именем `theRadius`. Мы используем вторую переменную `theArea` для хранения результатов расчетов (23.5), поэтому мы должны объявить ее (23.4) также, как объявили переменные в функции `main()` (22.4). Следует заметить, что объявление переменной `theRadius` делается в скобках (23.2). Строка (23.6) возвращает результат в программу, из которого функция была вызвана. В результате в строке (22.11) переменная `circleSurfaceArea` установлена в это значение.

Функция в примере (23) завершена за одним исключением. Мы не указали тип результата, который возвращается функцией. Компилятор требует от нас сделать это, поэтому у нас нет иного выбора, кроме как подчиниться и указать ему вещественный тип (24.2).

```

//(24)
float circleArea(float theRadius) //(24.2)
{
float theArea;
theArea = 3.14159 * theRadius * theRadius;
return theArea;
}

```

Первое слово в строке (24.2) показывает, что данные, возвращаемые этой функцией (т.е. значение переменной `theArea`), — вещественного типа. Как программист, вы должны сделать переменную `circleSurfaceArea` в функции `main()` (22.5) такого же типа, чтобы

компилятор не пилил нас за это.

Не все функции требуют параметров. Если их нет, скобки () требуются, но остаются пустыми.

```
//(25)
int throwDice()
{
    int noOfEyes;
    // Код для генерации случайного значения от 1 до 6 return noOfEyes;
}
```

Не все функции возвращают значение. Если функция ничего не возвращает, она имеет тип void. Оператор return является опциональным. Если вы используете тип void, то ключевое слово return не должно сопровождаться значением/названием переменной.

```
//(26)
void beepXTimes(int x)
{
    // Код для мычания x раз
    return;
}
```

Если функция принимает более одного аргумента как, например, функция pictureSurfaceArea() ниже, эти аргументы разделяются запятыми.

```
//(27)
float pictureSurfaceArea(float theWidth, float theHeight)
{
    // Код для расчета площади поверхности
}
```

Функция main(), в соответствии с соглашением, должна вернуть целое число, и так и есть — у нее тоже есть оператор return. Он должен вернуть 0 (28.9), чтобы показать, что функция выполнялась без проблем. Т.к. функция main() возвращает целое число, мы должны написать int перед main() (28.1). Теперь давайте напишем весь код, который мы имеем в одном списке.

```
//(28) int main()
{
    float pictureWidth, pictureHeight, pictureSurfaceArea, circleRadius, circleSurfaceArea;
    pictureWidth = 8;
    pictureHeight = 4.5;
    circleRadius = 6.0;
    pictureSurfaceArea = pictureWidth * pictureHeight;    circleSurfaceArea =
    circleArea(circleRadius); // (28.10) return 0; // (28.11)
}

float circleArea(float theRadius) // (28.14)
{
    float theArea;
    theArea = 3.14159 * theRadius * theRadius; return theArea;
}
```

3.9 Запуск приложения

Как вы можете видеть (28), у нас есть функция `main()` (28.2) и другая функция, описанная нами (28.14). Если мы попытаемся скомпилировать этот код, компилятор продолжит ругаться. В строке (28.10) он не видит, что существует функция `circleArea()`. Почему? Видимо, компилятор начинает чтение функции `main()` и вдруг сталкивается с тем, чего не знает. Он останавливается и дает вам предупреждение. Чтобы ублажить компилятор, нужно просто добавить описание этой функции перед функцией `main()` (29.2). Ничего сложного в этом нет, поскольку оно такое, как и в строке (28.14), только заканчивается точкой с запятой. Теперь компилятор не будет удивлен, когда столкнется с вызовом этой функции.

```
//(29)
float circleArea(float theRadius); // объявление функции

int main()
{
// Код функции main здесь...
}
```

Примечание: остальная часть программы не показана (см. (28)).

Скоро мы сможем скомпилировать эту программу. Сначала несколько ненужных деталей. Во время написания программ рекомендуется иметь в виду возможность повторного

использования кода в будущем. Наша программа может иметь функцию `rectangleArea()`, как показано ниже (30), и функция может быть вызвана в функции `main()`. Это полезно даже в том случае, если код функции используется лишь один раз. Функцию `main()` теперь легче читать. Если понадобится отладить код, вам будет легче найти ошибку в вашей программе. Вы можете обнаружить, что она в функции.

Вместо того, чтобы долго искать в коде, вам надо просто проверить вычисления в функции, которые легко найти, благодаря открытию и закрытию фигурных скобок.

```
//(30)
float rectangleArea(float length, float width)
{
return (length * width);
}
```

Как вы видите, в простых случаях, вроде этого, можно совместить вычисления и возврат результата в одном операторе (30.3). На самом деле, лишняя переменная `theArea` в (28.16) была нужна только как пример локального определения переменной в функции.

Хотя функции, которые мы создали в этой главе, достаточно тривиальные, важно понимать главное — теперь вы можете свободно изменять тело функции не влияя на остальной код, который эту функцию вызывает. Конечно, это все верно только пока вы не измените заголовок функции (т.е. ее имя, список параметров и возвращаемое значение, указанные в первой строчке).

Например, вы можете изменить названия переменных внутри функции, и функция будет работать как и раньше (и на остальной код это не повлияет). Кто-то другой тоже может написать функцию, и вы сможете её использовать не зная, что происходит у неё внутри. вам надо только знать как эту функцию использовать, а для этого понадобятся:

- Название функции;
- Число, порядок и типы аргументов функции;
- Что функция возвращает (значение площади поверхности прямоугольника), и тип возвращаемого значения.

В примере (30), эти значения соответственно будут:

`rectangleArea;`

- 2 аргумента, оба вещественного типа, где первый представляет длину, а второй ширину;
- Функция что-то возвращает и тип возвращаемого значения `float` (это видно по первому слову строки (30.2)).

Код внутри функции защищен от воздействия извне (от основной программы и кода других функций).

Что это означает, что значение переменной внутри функции по умолчанию не влияет на любую другую переменную в любой другой функции, даже если он имеет такое же название. Это является наиболее важной чертой Objective-C. В главе 5, мы будем обсуждать данную проблему еще раз. Но сперва, мы разберемся с Xcode, запустим код из примера (29).

3.10 Вывод на экран

Мы добились хорошего прогресса в нашей программе, но мы еще не знаем как показать результаты вычислений. Язык Objective-C сам не знает как это сделать, но к счастью люди написали функции вывода на экран, которые мы и будем использовать. Есть много методов вывода на экран. В этой книге мы будем использовать функцию окружения Apple Cocoa — функцию `NSLog()`. Это хорошо, потому что теперь вам не придется беспокоиться о выводе чего-либо на экран.

Функция `NSLog()` изначально была разработана для отображения ошибок, а не для вывода результатов работы программы. Но из-за простоты использования, мы адаптировали ее в этой книге для отображения результатов. Когда вы немного изучите Cocoa, вы сможете использовать более изощренные методы.

Давайте взглянем на то, как используется функция `NSLog()`.

```
//(31) int main()
{
    NSLog(@"Юлия — моя любимая актриса."); return 0;
}
```

При исполнении инструкций из примера (31) текст: «Юлия — моя любимая актриса.» будет выведен на экран.

Текст между `@` и `"` называется строкой.

В дополнение к самой строке, функция `NSLog()` выводит различную дополнительную информацию, текущую дату и имя приложения. Например на моей системе программа (31) выводит:

```
2005-12-22 17:39:23.084 test[399] Юлия - моя любимая актриса.
```

Строка может иметь длину ноль и более символов.

Примечание: В следующих примерах показаны только те строки, которые представляют для нас интерес.

```
//(32) NSLog(@"");
NSLog(@" ");
```

Выражение (32.2) содержит 0 символов, это называется пустой строкой (например, строка, длина которой равна 0). Выражение (32.3) не является пустой строкой, несмотря на то, что она выглядит очень похоже. Она содержит один пробел, а значит ее длина равна 1.

Некоторые специальные последовательности символов в строках имеют особый

смысл. Такие символы называются эскейп-символами.

Например, для того, чтобы последнее слово в нашем предложении напечаталось с начала новой строки, необходимо включить в выражение (33.2) специальный код. Таким кодом является `\n`, называемый символом новой строки.

```
//(33)
NSLog(@"Юлия — моя любимая \nактриса.");
```

Теперь вывод строки на экран будет выглядеть следующим образом (только относительный вывод будет показан):

Юлия — моя любимая актриса.

Знак `\` (обратный слеш) в (33.2) называется эскейп-символом, так как он служит сигналом для функции `NSLog()` о том, что следующий символ не обычная буква, а символ, имеющий особое значение: в нашем примере `n` означает «начать новую строку».

В редком случае, когда вы хотите напечатать обратный слеш `\`, может показаться, что у вас есть проблема. Если у обратного слеша есть специальное значение, то как можно его напечатать? Хорошо, мы помещаем еще один обратный слеш перед или после первого. Это сообщает функции `NSLog()`, что второй обратный слеш должен быть напечатан и что любое специальное значение должно быть проигнорировано. Например:

```
//(34)
NSLog(@"Юлия — моя любимая актриса.\n");
Инструкция (34.2) выдаст
Юлия — моя любимая актриса.\n
```

3.11 Печать значений переменных

Пока мы показывали только статические строки. Давайте напечатаем вычисленное значение на экран.

```
//(35)
int x, integerToDisplay; x = 1;
integerToDisplay = 5 + x;
NSLog(@"Целое значение — %d.", integerToDisplay);
```

Имейте в виду, что в скобках у нас стоят строка, запятая и имя переменной. Строка содержит нечто странное: `%d`. Как и обратный слеш `\`, символ процента `%` имеет особое значение. Если после него следует `d` (малое десятичное число), то после выполнения кода вместо `%d` будет показано то, что стоит после запятой, т.е. текущее значение переменной `integerToDisplay`. Запуск примера (35) приводит к

Целое значение — 6

Для печати вещественных значений вам следует использовать `%f` вместо `%d`

```
//(36)
float x, floatToDisplay; x = 12345.09876;
floatToDisplay = x/3.1416;
NSLog(@"Вещественное значение — %f.", floatToDisplay);
```

Вы сами можете указать, сколько оставить цифр после точки. Чтобы показать 2 значимые цифры, поставьте .2 между % и f, вот так:

```
/(37)
float x, floatToDisplay; x = 12345.09876;
floatToDisplay = x/3.1416;
NSLog(@"Вещественное значение с 2 значимыми цифрами — %.2f.",
floatToDisplay);
```

Позже, когда Вы узнаете, как повторять вычисления, Вы можете захотеть создать таблицу. Представьте, например, таблицу для перевода из градусов Фаренгейта в градусы Цельсия. Если вы хотите чтобы таблица выглядела опрятно, значения в столбцах должны быть одинаковой ширины. Вы можете указать нужную ширину (количество символов) непосредственно между % и f (или между % и d).

Однако, если Вы укажете ширину меньше чем реальная ширина числа, ваше указание будет проигнорировано.

```
/(38)
int x = 123456; NSLog(@"%2d", x);
NSLog(@"%4d", x);
NSLog(@"%6d", x);
NSLog(@"%8d", x);
```

Пример (38) имеет следующий вывод:

```
123456
123456
123456
123456
```

В первые двух выражениях, (38.3, 38.4) мы запрашиваем слишком мало пространства для количества символов, которое будет отображаться, но выражение все равно выводится целиком. Только выражение, (38,6) задает ширину большую, чем значение, поэтому сейчас мы видим появление дополнительного пространства, свидетельствует о размере места для выводимого числа.

Кроме того, можно объединить параметры ширины и количества десятичных чисел, который должны отображаться.

```
/(39)
float x=1234.5678;
NSLog(@"Зарезервировать 10 символов, и отобразить 2 значащих цифры.");
NSLog(@"%10.2d", x);
```

Конечно, можно отображать более чем одно значение, или любое сочетание значений (40.4).

Вы должны убедиться, что вы правильно указываете тип данных (int, float), используя %d и

%f соответственно.

```
/[40] int x = 8;
float pi = 3.14159;
NSLog(@"Целое значение — %d, вещественное значение — %f.", x, pi);
```

Одна из самых распространённых ошибок начинающих программистов это неправильный выбор типов данных для NSLog() и других функций. Если результаты ваших программ кажутся вам странными или если программа завершает работу нестандартно, без причины. Проверьте типы данных!

Если у вас получилось в первый раз, это не значит, что во второй получится отобразить нормально.

```
//(40b) int x = 8;
float pi = 3.1416;
NSLog(@"Целое значение — %f, вещественное значение — %f.", x, pi);
// На самом деле должно быть: NSLog(@"Целое значение — %d, вещественное значение — %f.", x, pi);
```

Получим следующий выход:

Целое значение — 0.000000, вещественное значение — 0.000000.

3. 12 Использование фреймворка Foundation

От первого запуска программы нас отделяет только один вопрос.

Как наша программа узнает об этой полезной функции NSLog()? Ну, она и не узнает, пока мы ей не скажем. Чтобы это сделать, наша программа должна сообщить компилятору, чтобы тот импортировал библиотеку «сладостей», включая и функцию NSLog(). Используйте следующую конструкцию:

```
#import <Foundation/Foundation.h>
```

Эта команда должна быть первой в вашей программе. Собрав вместе все, чему мы научились в этой главе, мы получим код, который мы запустим в следующей главе:

```
//(41)
#import <foundation/foundation.h>

float circleArea(float theRadius);
float rectangleArea(float width, float height);

int main()
{
    float pictureWidth, pictureHeight, pictureSurfaceArea, circleRadius, circleSurfaceArea;
    pictureWidth = 8.0;
    pictureHeight = 4.5;
    circleRadius = 5.0;
    pictureSurfaceArea = rectangleArea(pictureWidth, pictureHeight); circleSurfaceArea
= circleArea(circleRadius);
    NSLog(@"Площадь окружности: %10.2f.", circleSurfaceArea); NSLog(@"Площадь
картинки: %f. ", pictureSurfaceArea); return 0;
}

float circleArea(float theRadius) // первая пользовательская функция
```

```
{
float theArea;
theArea = 3.14159 * theRadius * theRadius; return theArea;
}
```

```
float rectangleArea(float width, float height)
{
return width*height;
}
```

3.13 Компиляция и запуск программ

Полученный нами код, пока что, не более чем читаемый текст. И хотя для нас это - не литературное произведение, Mac'у он ещё более непонятен. С ним он не сможет сделать ровным счётом ничего! Для преобразования вашего программного кода в исполняемый Mac'ом необходима специальная программа, называемая компилятором. Она является частью программной среды Xcode, которую вам нужно установить с диска, идущего совместно с копией Mac OS X. В любом случае, проверьте, что у вас установлена последняя версия, которую можно скачать в разделе для разработчиков <http://developer.apple.com> (необходима бесплатная регистрация).

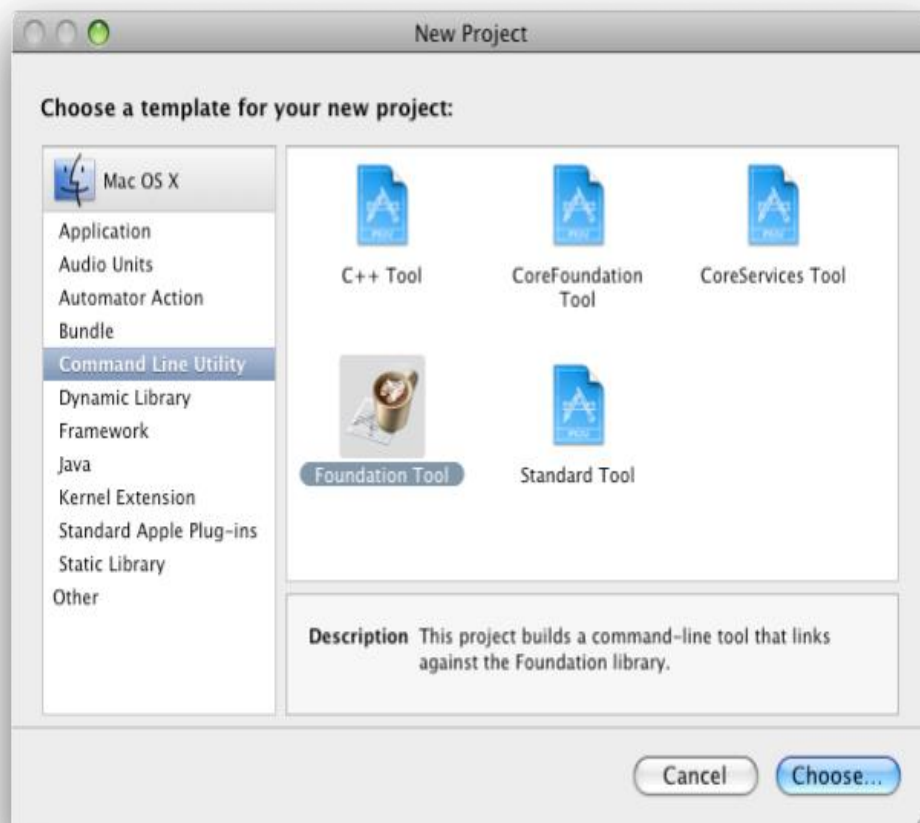
Запустим, теперь, Xcode, который располагается в папке /Development/Application/ на вашем диске или же просто введите в Spotlight «xcode» и нажмите клавишу ← на клавиатуре. При первом запуске будет задано несколько вопросов. На все из них можно ответить

утвердительно и в случае чего никогда не поздно изменить их в настройках. Для того чтобы начать работу, нужно создать новый проект (File → New Project ↑%N). При этом появится диалоговое окно с возможными типами проектов.

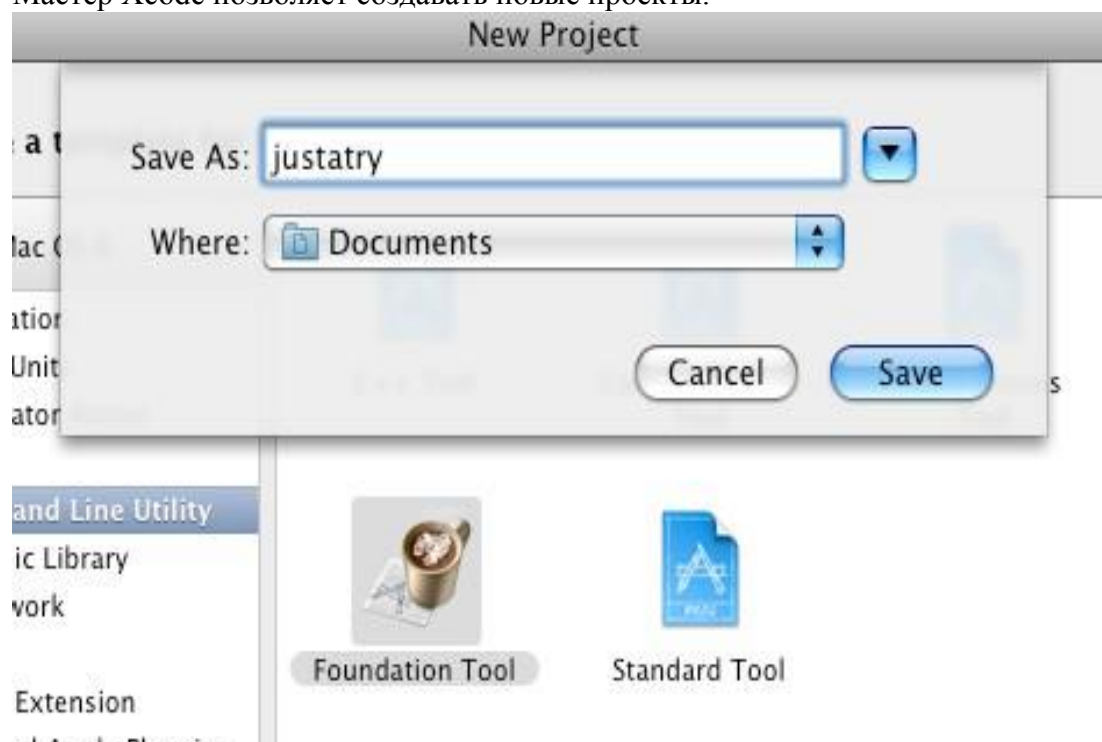
Мы хотим создать очень простую программу на Objective-C, без графического пользовательского интерфейса (GUI). Промотайте вниз и выберите Foundation Tool под секцией Command Line Utility.

Введите имя вашего приложения, например «justatry». Выберите место на диске, куда вы сохраните ваш проект и сохраните (нажмите кнопку Save).

Проект, который мы собираемся создать, можно запустить из Терминала (Terminal). Если вы хотите сделать это, и избежать некоторых трудностей, убедитесь, что название вашего проекта состоит из одного слова. Кроме того, обычно названия программ, запускающихся из Терминала, начинаются с маленькой буквы. С другой стороны, название программ с графическим интерфейсом пользователя должно начинаться с заглавной.



Мастер Xcode позволяет создавать новые проекты.



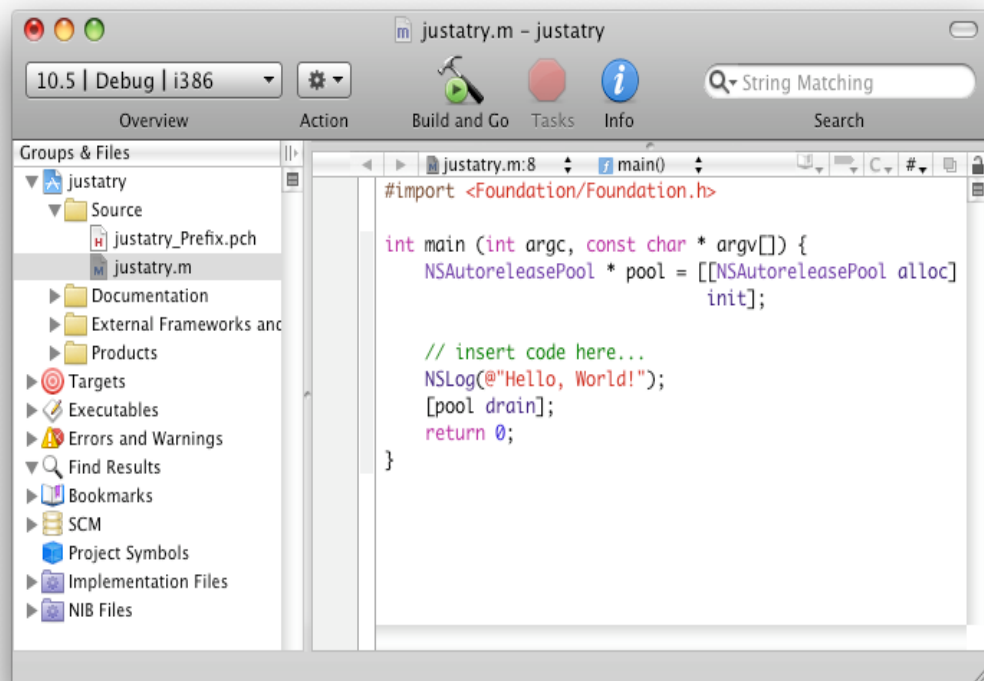
Выбор имени и месторасположения нового проекта.

3.14 Обзор Xcode

Теперь перед вами окно, которое вы, как программист, будете видеть очень часто. Окно состоит из двух фреймов. Основной фрейм располагается справа и в основном в нём происходит работа с исходными кодами. «Groups & Files» — располагается слева и служит для доступа ко всем файлам, из которых состоит ваша программа. Сейчас их не слишком много, но позже, когда вы будете работать над многоязычными графическими программами, все файлы для описания интерфейса и различных языков будут именно тут. Файлы сгруппированы и хранятся в папках, но не стоит их искать на вашем жестком диске.

Xcode предоставляет эти виртуальные папки (Groups) для организации вашего проекта.

В поле ввода «Groups & Files», попробуйте открыть группу justatray, для того чтобы начать работать с исходным кодом в файле с именем justatry.m [42]. Вы помните, что каждая программа должна содержать функцию с именем main()? Ну вот, этот файл, содержит эту main() функцию. Позже в этой главе мы собираемся изменить его, чтобы включить его в код нашей программы. Если вы открываете justatry.m, дважды щелкнув его значок, вас ждет приятный сюрприз. Apple уже создала main() функцию для вас.



Xcode, отображающий main() функцию.

//(42)

```
#import <Foundation/Foundation.h>
```

```
int main (int argc, const char * argv[]) // (42.3)
```

```
{
```

```
NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init]; // (42.5)
```

```
// insert code here... NSLog(@"Hello, World!"); [pool drain]; //(42.9) return 0;
```

```
}
```

Давайте взглянем на эту программу и поищем то, что вы уже знаете:

- Директива import, начинающаяся символом #, нужна для использования таких

функций как NSLog;

- Функция main();
- Фигурные скобки, которые должны содержать тело нашей программы;
- Комментарий, приглашающий писать код здесь;
- Команда NSLog() для вывода строки на экран;
- Команда return 0.

Есть здесь и то, что вы пока не знаете:

- Забавный список аргументов в скобках функции main() (42.3);
- Оператор, начинающийся на NSAutoreleasePool (42.5);
- Еще один оператор со словами pool и drain (42.9).

Лично я не очень-то счастлив, когда авторы книг предлагают мне, как читателю, ознакомиться с кодом, полным непонятных операторов, обещая при этом все разъяснить позже. Ага, как же. Именно поэтому я слегка отклонился от темы чтобы познакомить вас с концепцией «функций», так что вам не придется сталкиваться с большим количеством новых понятий.

Вы уже знаете, что функции это способ организовать программу, что в каждой программе есть функция main(), и как функции выглядят. Тем не менее, должен признаться, я не могу сейчас полностью объяснить все, что вы видите в примере (42). Я очень сожалею, но пока просто не обращайте внимания на эти строки ((42.3, 42.5 и 42.9)).

Есть еще несколько моментов в Objective-C, с которыми стоит познакомиться в самом начале и которые позволят создавать простые программы. Две последних главы были довольно сложные, но в следующих трёх главах все будет просто (нам нужно экономить силы для штурма следующей вершины).

Если вы действительно не хотите быть оставленными без каких-либо объяснений, вот вам резюме:

- Аргументы в main() (42.3) функции необходимы для запуска программ из Терминала.

При помощи этих аргументов передаются параметры в программу (прим. редактора);

- Ваша программа резервирует память (42.5). Память, которая нужна другим программам, когда ваша программа закончила работу. Ваша задача резервировать память, которая нужна вашей программе;

- И не менее важно: освобождение памяти, когда она не нужна (42.9).

Давайте запустим программу (42). Нажмите вторую иконку «с молотком» помеченную как

«Build and Go» для постройки (сборки) и запуска приложения.



Кнопка Build and Go

Программа выполняется и результаты выводятся в окно «Console» (сделать видимым его можно, находясь в Xcode, выбрав пункты меню Run → Console ↑%R), вместе с дополнительной информацией. Последняя строка сообщает нам, что программа завершилась с кодом 0. Этот

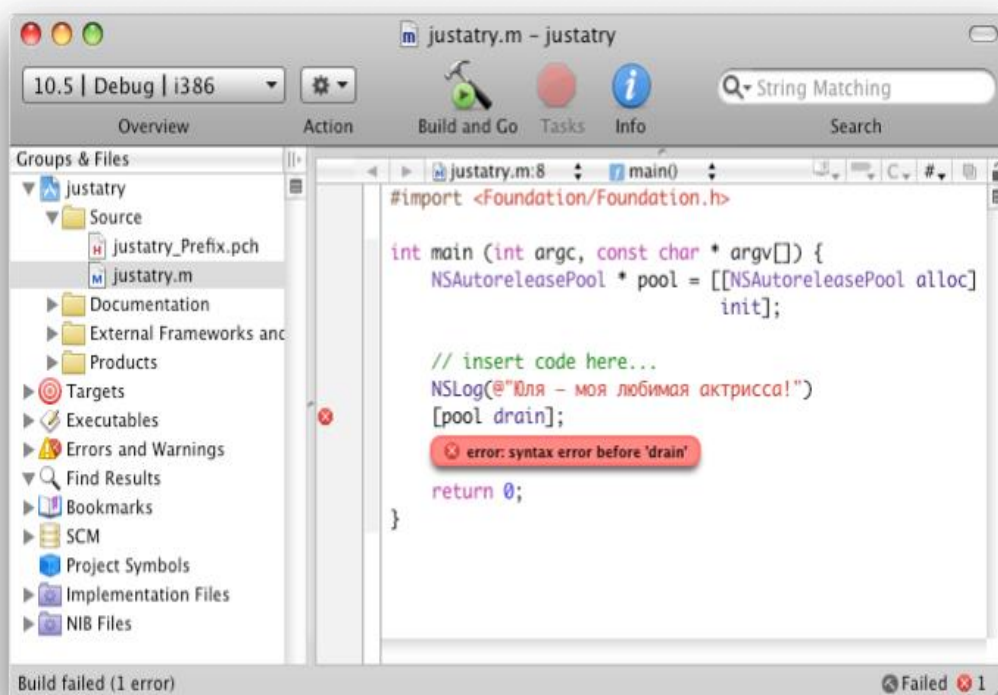
как раз тот 0, который вернула функция main(), которую мы обсуждали ранее (28.11). Значит, наша программа выполнилась до последней строки и нигде не прервалась раньше времени. Уже хорошо!

Вернемся к примеру (42) и посмотрим, что произойдет, если имеется ошибка в программе. Например, я поменял функцию NSLog(), но забыл поставить точку с запятой в конце команды.

```
 //(43)
#import <Foundation/Foundation.h>

int main (int argc, const char * argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    // insert code here...
    NSLog(@"Юля — моя любимая актрисса") // Упс, забыл поставить точку с ←
    запятой!
    [pool drain]; //[43.9] return 0;
}
```

Чтобы построить приложение, нажмите иконку «Build and Go» на панели. Перед командой (43.9) появится красный круг.



Xcode указывает на ошибку компиляции.

Если вы кликните его, строка под тулбаром покажет короткое пояснение ошибки:

error: syntax error before "drain"

Синтаксический анализ, или парсинг — один из первых этапов работы компилятора: он проходит по каждой строчке кода, чтобы проверить может ли он её понять. И вы должны помочь ему в этом, размещая подсказки. Например, для директивы import надо использовать символ #, а конец оператора надо отмечать с помощью точки с

запятой (;). В нашем случае, когда компилятор доходит до строчки [43.9], он замечает, что что-то не так. Однако, он не понимает, что проблема не в этой строке, а в предыдущей, где не хватает точки с запятой (компилятор считает данную запись из двух операторов одним оператором, которую не может разобрать — прим. редактора). Важный вывод для нас состоит в том, что хотя компилятор пытается давать разумные пояснения, не стоит рассчитывать, что его сообщения обязательно укажут точную причину и место ошибки (хотя скорее всего он будет очень близок к этому).

Исправьте код добавив точку с запятой, и запустите программу еще раз, чтобы убедиться что теперь все в порядке.

3.15 Наше первое приложение

А сейчас давайте возьмем код из предыдущей главы и объединим его с кодом, который подготовил для нас Apple ((42)). В результате получим пример (44).

```
//(44)
#import <Foundation/Foundation.h>

float circleArea(float theRadius); // (44.4)
float rectangleArea(float width, float height); // (44.5)

int main (int argc, const char * argv[]) // (44.7)
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init]; int pictureWidth;
    float pictureHeight, pictureSurfaceArea, circleRadius, circleSurfaceArea;
    pictureWidth = 8;
    pictureHeight = 4.5;
    circleRadius = 5.0;
    pictureSurfaceArea = pictureWidth * pictureHeight; circleSurfaceArea =
    circleArea(circleRadius); NSLog(@"Площадь картинки: %f. Площадь окружности: %10.2f.",
    pictureSurfaceArea, circleSurfaceArea); [pool drain];
    return 0;
}

float circleArea(float theRadius) // (44.24)
{
    float theArea;
    theArea = 3.14159 * theRadius * theRadius; return theArea;
}

float rectangleArea(float width, float height) // (44.31)
{
    return width *height;
}
```

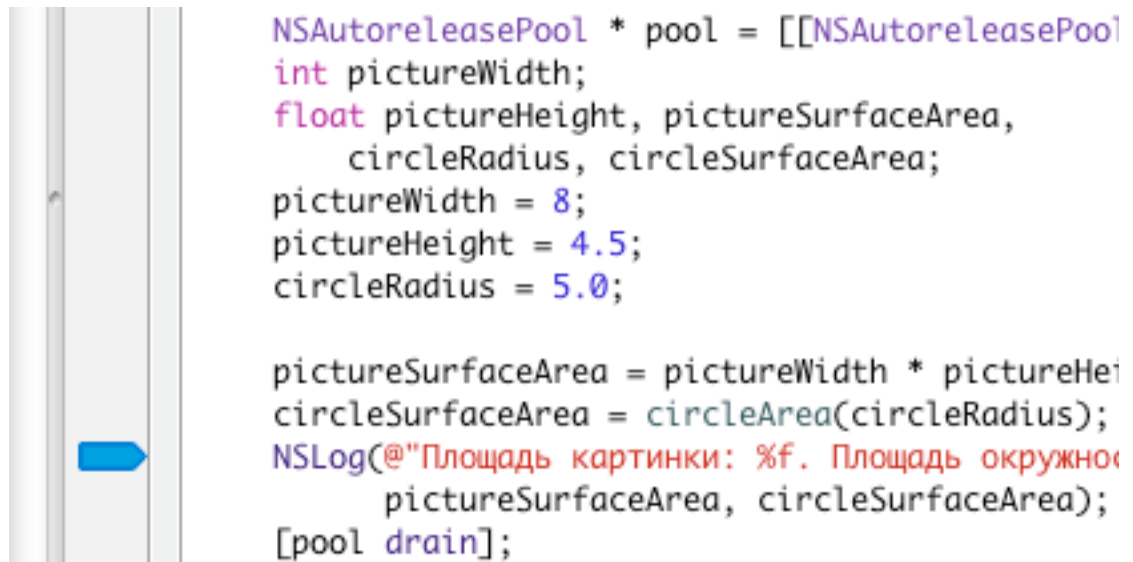
Убедитесь, что вы понимаете структуру программы. У нас имеются заголовки (44.4, 44.5) созданных нами функций circleArea() (44.24) и rectangleArea() (44.31), находящиеся выше функции main() (44.7), где они и должны быть. Наши функции находятся вне фигурных скобок функции main() [44.8, 44,22]. Мы набираем текст программы внутри тела функции main(), где это и советует делать Apple.

После исполнения кода, мы увидим:

Площадь картинки: 35.000000. Площадь окружности: 78.54.

The Debugger has exited with status 0.

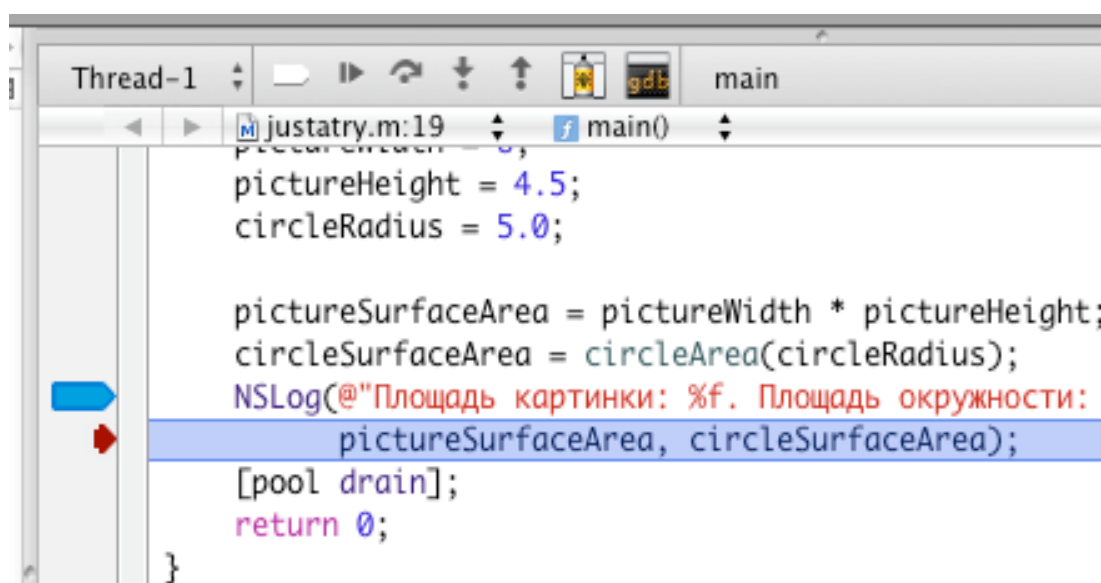
Когда программа усложняется, отладка выполняется труднее. Поэтому периодически возникает необходимость узнать, что происходит внутри программы в момент её выполнения. С этой задачей также с успехом справляется Xcode, необходимо лишь нажать левой кнопкой мыши на серой границе перед выражениями, для которых вы хотите узнать значения переменных и Xcode добавит в этом месте точку остановки — breakpoint, отмеченную голубой стрелкой.



Установка точки остановки в коде.

Запомните, вы увидите значения переменных до выполнения строки с точкой остановки, так что ставьте точку остановки в следующей строке после изменения нужного значения.

Нажав «Build and Go (Debug)» (Run → Debug ↑%Y), перед вами будет оно Xcode в режиме отладки.



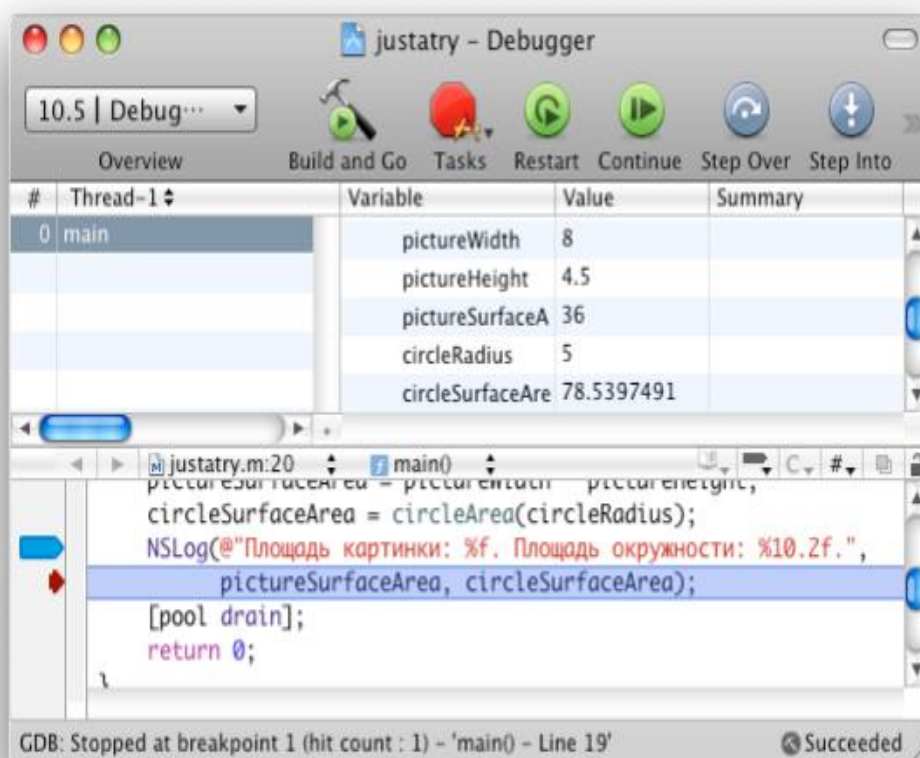
Xcode в режиме отладки позволяет выполнять программу шаг за шагом и проверять переменные.

Программа будет выполняться до того момента как достигнет первой точки остановки. Если выбрать верхнюю правую панель в отладчике (Debugger. Для открытия отладчика

Run → Debugger ↑%Y), то вы сможете увидеть значения различных переменных. Любые значения, которые изменились с момента отображения последней точки остановки выделены красным цветом. Для продолжения выполнения нажмите кнопку «Continue». Отладчик — мощный инструмент, который поможет разобраться со многими ошибками, возникающими при написании программ. Поиграйтесь с ним некоторое время, чтобы поплотнее с ним ознакомиться.

Теперь у нас есть все, чтобы писать, отлаживать и запускать простые программы для Mac OS X.

Если вы не хотите делать программы с графическим пользовательским интерфейсом (GUI), единственное что вам нужно сейчас — увеличить ваши знания Objective-C, что позволит разрабатывать более сложные программы. В течение следующих нескольких глав мы будем делать именно это. После чего мы будем погружаться приложения с GUI.



Окно отладчика в процессе работы с программой.

4 Введение в язык программирования Objective-C

4.1 Основные понятия

Помимо широкого известного и распространенного объектного расширения языка C - языка C++ - есть и другое его расширение - язык *Objective-C*, обладающий огромной простотой, полной совместимостью с языком C и очень мощной и выразительной объектной моделью, заимствованной из языка *Smalltalk*.

Язык был придуман Брэдом Коксом (**Brad Cox**) в начале 80-х годов прошлого века. Целью Кокса было создание языка, поддерживающего концепцию *software IC*. Под этой концепцией понимается возможность собирать программы из готовых компонент (объектов), подобно тому как сложные электронные устройства могут быть легко собраны из набора готовых интегральных микросхем (*IC, integrated circuits*).

При этом такой язык должен быть достаточно простым и основанным на языке C, чтобы облегчить переход разработчиков на него.

Одной из целей было также создание модели, в которой сами классы также являются полноценными объектами, поддерживалась бы интроспекция и динамическая обработка сообщений.

Получившийся в результате язык *Objective-C* оказался крайне прост - его освоение у C-программиста займет всего несколько дней. Он является именно расширением языка C - в язык C просто добавлены новые возможности для объектно-ориентированного программирования. При этом любая программа на C является программой и на *Objective-C* (для языка C++ это не верно).

Одной из отличительных черт *Objective-C* является его динамизм - целый ряд решений, обычно принимаемых на этапе компиляции, здесь откладывается непосредственно до этапа выполнения.

Еще одной из особенностей языка является то, что он ориентирован на посылку сообщений (*message-oriented*) в то время как C++ - функционально-ориентированный язык (*function-oriented*). Это значит, что в нем вызовы метода интерпретируются не как вызов функции (хотя к этому обычно все сводится), а именно как посылка сообщения (с именем и аргументами) объекту.

Такой подход дает целый ряд плюсов - так любому объекту можно послать любое сообщение. Объект может вместо обработки сообщения просто переслать его другому объекту для обработки (так называемое делегирование), в частности именно так можно легко реализовать распределенные объекты (т.е. объекты находящиеся в различных адресных пространствах и даже на разных компьютерах).

Привязка сообщения к соответствующей функции происходит непосредственно на этапе выполнения.

Язык *Objective-C* поддерживает нормальную работу с метайнформацией - так у объекта непосредственно на этапе выполнения можно спросить его класс, список методов (с типами передаваемых аргументов) и *instance*-переменных, проверить, является ли класс потомком заданного и поддерживает ли он заданный протокол и т.п.

В языке есть нормальная поддержка протоколов (т.е. понятие интерфейса объекта и протокола четко разделены). Для объектов поддерживается наследование (не множественное), для протоколов поддерживается множественное наследование. Объект может быть унаследован от другого объекта и сразу нескольких протоколов (хотя это скорее не наследование протокола, а его поддержка).

На данный момент язык *Objective-C* поддерживается компилятором **gcc** (соответственно для форточек он поддерживается **mingw** и **cygwin**). Судя по заявлениям в новой версии **Mac OS X 10.5** будет кроме **gcc** (используемого для всех предыдущих версий) еще и компилятор от фирмы Intel. Также в этой версии операционной системы будет использован *Objective-C 2*, куда в частности войдет поддержка сборки мусора (*garbage collection*).

Довольно много в языке перенесено на *runtime*-библиотеку и сильно зависит от нее. Вместе с компилятором **gcc** поставляется минимальный вариант такой библиотеки. Также можно свободно скачать *runtime*-библиотеку от компании Apple.

Эти две *runtime*-библиотеки довольно похожи (в основном отличие заключается в именах методов), хотя далее будем ориентироваться на *runtime*-библиотеку от компании Apple.

4.2 Синтаксис языка

В языке *Objective-C* для обозначения объектов используется специальный тип **id**. Переменная типа **id** фактически является указателем на произвольный объект. Для обозначения нулевого указателя на объект используется константа **nil**.

При этом вместо **id** можно использовать и более привычное обозначение с явным указанием класса. В частности последнее позволяет компилятору осуществлять некоторую проверку поддержки сообщения объектами - если компилятор из типа переменной не может сделать вывод о поддержке объектом данного сообщения, то он выдаст предупреждение (а не ошибку!).

Тем самым язык поддерживает проверку типов, но в нестрогой форме (т.е. найденные несоответствия возвращаются как предупреждения, а не ошибки).

Для отправки сообщений используется следующий синтаксис:

```
[receiver message];
```

В этой конструкции *receiver* является указателем на объект, а *message* - именем метода.

В отличие от языка C++ отправка сообщения **nil**'у является законной операцией, всегда возвращающей нулевое значение (**nil**).

Сообщение может также содержать параметры:

```
[myRect setOrigin:30.0 :50.0];
```

В этом примере именем метода (сообщения) является **setOrigin::**. Обратите внимание, что каждому передаваемому аргументу соответствует ровно одно двоеточие. При этом в приведенном примере первый аргумент имеет метку (текст перед двоеточием), а второй - нет.

Язык *Objective-C* позволяет снабжать метками каждый аргумент, что заметно повышает читаемость кода и снижает вероятность передачи неправильного параметра.

```
[myRect setWidth:10.0 height:20.0];
```

В этом примере в качестве имени сообщения выступает **setWidth:height::**.

Также поддерживается возможность передачи произвольного количества аргументов в сообщении:

```
[myObject makeGroup: obj1, obj2, obj3, obj4, nil];
```

Как и функции, сообщения могут возвращать значения, при этом в отличие от языка C, типом возвращаемым по умолчанию значения является **id**.

```
float area = [myRect area];
```

Результат одного сообщения можно сразу же использовать в другом сообщении:

```
[myRect setColor:[otherRect color]];
```

Как уже говорилось, в *Objective-C* классы сами являются объектами. Основной задачей таких объектов (называемых *class objects*) является создание экземпляров данного класса (это очень похоже на паттерн **Abstract Factory**).

При этом само имя класса играет двойную роль - с одной стороны оно выступает как тип данных (т.е. он может быть использован для описания указателей на объекты данного класса). А с другой стороны имя класса может выступать в качестве объекта, которому посылаются сообщения (в сообщениях имя класса может принимать участие *только* как *receiver*).

```
Rect * myRect = [[Rect alloc] init];
```

В языке *Objective-C* нет встроенного типа для булевских величин, поэтому обычно такой тип вводится искусственно. Далее для логических величин будет использоваться

тип **BOOL** с возможными значениями **YES** и **NO** (как это делается в операционных системах **NextStep, Mac OS X**).

Первым достаточно серьезным применением языка *Objective-C* было его использование в операционной системе **NextStep**. Для этой системы было написано большое количество различных классов на *Objective-C*, многие из которых до сих пор используются в **Mac OS X**.

Имена всех этих классов начинаются с префикса **NS**, обозначающего свою принадлежность к операционной системе **NextStep**.

Один из таких классов - **NSString** – будет использоваться далее. Этот класс служит для работы со строками (при этом в качестве внутреннего представления символов используется Юникод).

Компилятор поддерживает данный тип, автоматически переводя конструкции вида `@"my string"` в указатель на объект класса **NSString**, содержащий данную строку (точнее его подкласса, соответствующего константным строкам).

4.3 Создание новых классов

Все новые директивы компилятору в языке *Objective-C* начинаются с символа `@`.

Как и в C++ описание класса и его реализация разделены (обычно описание помещается в заголовочные файлы с расширением **h**, а реализации - в файлы с расширением **m**).

Ниже приводится общая структура описания нового класса:

```
@interface ClassName : SuperClass
{
    instance variable declarations
}

method declarations
@end
```

В версии *runtime* от Apple все классы имеют общего предка - класс **NSObject**, содержащий целый ряд важных методов.

Описание переменных ничем не отличается от описания переменных в структурах в языке C:

```
@interface Rect : NSObject
{
    float    width;
    float    height;
    BOOL     isFilled;
    NSColor * color;
}
@end
```

Описания же методов заметно отличаются от принятых в C++ и очень сильно похожи на описания методов в языке *Smalltalk*.

Каждое описание начинается со знака плюс или минус. Знак плюс обозначает, что данный метод является методом класса (т.е. его можно посылать только *class object*'у, а не экземплярам данного класса). Фактически методы класса являются аналогами статических методов в классах в языке C++.

Знак минус служит для обозначения методов объектов - экземпляров данного класса. Обратите внимание, что в *Objective-C* все методы являются виртуальными, т.е. могут быть переопределены.

Ниже приводятся описания возможных методов для класса **Rect**.

```
@interface Rect : NSObject
{
    float    x, y;
    float    width;
    float    height;
    BOOL     isFilled;
}
```

```

    NSColor * color;
}
+ newRect;
- (void) display;
- (float) width;
- (float) height;
- (float) area;
- (void) setWidth: (float) theWidth;
- (void) setHeight: (float) theHeight;
- (void) setX: (float) theX y: (float) theY;
@end

```

Обратите внимание, что имя метода может совпадать с именем *instance*-переменной данного класса (например, *width* и *height*).

Тип возвращаемого методом значения указывается в круглых скобках сразу же после знака плюс или минус (но перед названием метода). Если тип не указан, то считается, что возвращается значение типа **id**.

Далее идет имя метода, где после каждого двоеточия задается тип аргумента (в круглых скобках) и сам аргумент.

Язык *Objective-C* позволяет для аргументов метода задавать также один из следующих описателей - **oneway**, **in**, **out**, **inout**, **bycopy** и **byref**. Данные описатели служат для задания направления передачи данных и способа передачи. Их наличие заметно упрощает реализацию и работу с распределенными объектами (которые были реализованы в операционной системе **NextStep** к началу 90-х годов прошлого века).

Метод, принимающий произвольное количество параметров, может быть описан следующим образом:

```

- makeGroup: (id) object, ...;

```

Для подключения заголовочного файла в *Objective-C* вместо директивы **#include** используется директива **#import**, полностью аналогичная **#include**, но гарантирующая что данный файл будет подключен всего один раз.

В ряде случаев возникает необходимость в объявлении того, что данное имя является именем класса, но без явного его описания (такая необходимость возникает при описании двух классов, каждый из которых ссылается на другой класс).

В этом случае можно воспользоваться директивой **@class**, объявляющей, что следующие за ней имена являются именами классов.

```

@class Shape, Rect, Oval;

```

Реализация методов класса выглядит следующим образом:

```

#import "ClassName.h"

```

```

@implementation ClassName

```

```

    method implementations

```

```

@end

```

Ниже приводится пример реализации методов класса **Rect**, описанного выше.

```

#import "Rect.h"

```

```

@implementation Rect

```

```

+ newRect

```

```

{
    Rect * rect = [[Rect alloc] init];

    [rect setWidth: 1.0f];
    [rect setHeight: 1.0f];
    [rect setX: 0.0f y: 0.0f];
}

```

```

- (float) width

```

```

{

```

```

        return width;
    }

    - (float) height
    {
        return height;
    }

    - (float) area
    {
        return [self width] * [self height];
    }

    - (void) setWidth: (float) theWidth
    {
        width = theWidth;
    }

    - (void) setHeight: (float) theHeight
    {
        height = theHeight;
    }

    - (void) setX: (float) theX y: (float) theY
    {
        x = theX;
        y = theY;
    }

@end

```

Как видно из примера выше, в методах доступны все *instance*-переменные. Однако, как и в C++, есть возможность управлять видимостью переменных (видимостью методов управлять нельзя) при помощи директив **@private**, **@protected** и **@public** (действующих полностью аналогично языку C++).

```

@interface Worker : NSObject
{
    char * name;
@private
    int    age;
    char * evaluation;
@protected
    int    job;
    float  wage;
@public
    id     boss
}

```

При этом к *public* переменным класса можно обращаться непосредственно используя оператор *->* (например *objPtr -> fieldName*).

4.4 Механизм сообщений

Компилятор переводит каждую посылку сообщения, т.е. конструкцию вида **[object msg]** в вызов функции *objc_msgSend*.

Эта функция в качестве своего первого параметра принимает указатель на объект-получатель сообщения, в качестве второго параметра выступает т.н. селектор, служащий для идентификации посылаемого сообщения. Если в сообщении присутствуют аргументы, то они также передаются *objc_msgSend* как третий, четвертый и т.д. параметры.

Каждый объект *Objective-C* содержит в себе атрибут **isa** - указатель на *class object* для данного объекта. *class object* автоматически создается компилятором и существует как один экземпляр, на который через **isa** ссылаются все экземпляры данного класса.

Каждый *class object* обязательно содержит в себе указатель на *class object* для родительского класса (*superclass*) и *dispatch table*. Последняя представляет из себя словарь, сопоставляющий селекторам сообщений фактические адреса реализующих их методов (функций).

Т.о. функция *objc_msgSend* ищет метод с данным селектором в *dispatch table* для данного объекта. Если его там нет, то поиск продолжается в *dispatch table* для его родительского класса и т.д.

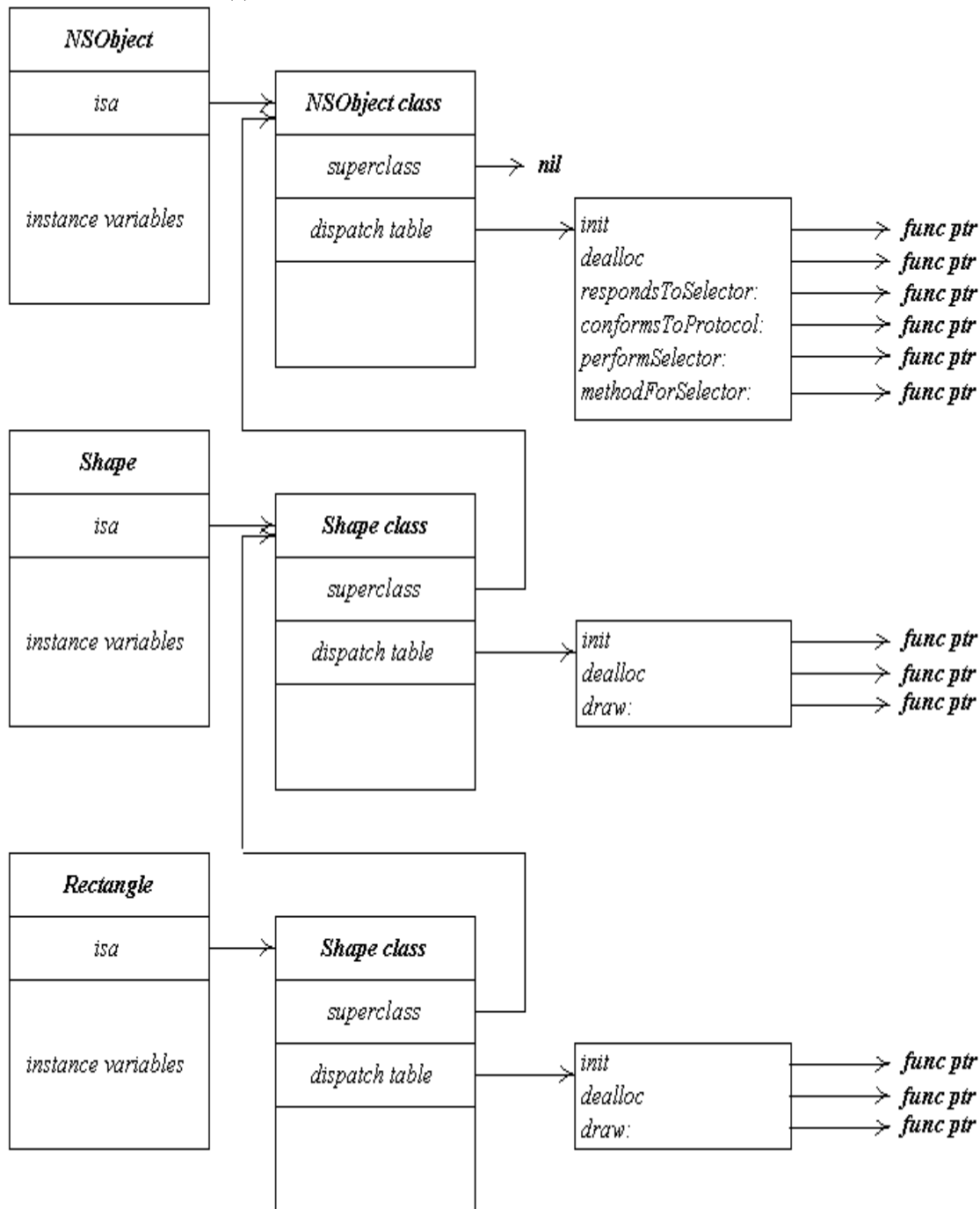


Рисунок 4.1 Диаграмма для классов (со следующим порядком наследования **Rectangle:Shape:NSObject**)

Если метод (т.е. соответствующая ему функция) находится, то осуществляется его вызов с передачей всех необходимых аргументов.

В противном случае объекту дается последний шанс обработать сообщение перед вызовом исключения - селектор сообщения вместе с параметрами "заворачивается" в специальный объект типа **NSInvocation** и объекту посылается сообщение *forwardInvocation:*, где в качестве параметра выступает объект класса **NSInvocation**.

Если объект поддерживает *forwardInvocation:*, то он может либо сам обработать посылаемое сообщение, либо переслать другому объекту для обработки:

```
- (void)forwardInvocation: (NSInvocation *)anInvocation
{
    if ( [someOtherObject respondsToSelector: [anInvocation selector]] )
        [anInvocation invokeWithTarget: someOtherObject];
    else
        .....
}
```

Для ускорения поиска сообщений по *dispatch table* используется кэширование, позволяющее заметно снизить затраты на пересылку сообщений. Стоит заметить, что язык *Objective-C* активно применялся для построения пользовательского интерфейса и библиотеки классов для операционной системы **NextStep**, в то время когда эта система работала на компьютерах с 25-Мгц процессорами Motorola 68030.

Если даже на таком процессоре язык обеспечивал приемлемое быстродействие, то сейчас о нем можно практически не думать.

Для облегчения поиска метода по таблицам в *Objective-C* вместо имен методов используются так называемые селекторы. Обычно селектор представляет собой 32-битовую величину, позволяющую однозначно идентифицировать метод.

Тип селектора обозначается как **SEL** и существует ряд функций и конструкций, позволяющих осуществлять преобразование имени в селектор и обратно.

Так для получения селектора сообщения непосредственно по имени служит конструкция **@selector()**:

```
SEL setWidth = @selector(setWidth:);
SEL setPos   = @selector(setX:y:);
```

Для получения селектора по строке символов (на этапе выполнения) и перевода селектора в строку служат функции *NSStringFromSelector* и *SelectorFromString*:

```
SEL      setWidth   = NSStringFromSelector ( @"setWidth:" );
NSString * methodName = NSStringFromSelector ( setPos );
```

Мощная поддержка метаинформации в *Objective-C* позволяет прямо на этапе выполнения проверить поддерживает ли объект метод с данным селектором при помощи посылки ему сообщения *respondsToSelector:*:

```
if ( [anObject respondsToSelector: @selector(setWidth:)] )
    [anObject setWidth: 200.0];
```

Довольно легко можно послать сообщение, соответствующее данному селектору (без аргументов, с одним, двумя или тремя аргументами) при помощи метода *performSelector:*, *performSelector:withObject:*, *performSelector:withObject:withObject:* и *performSelector:withObject:withObject::withObject:*.

```
[myObject performSelector:sel withObject: nil];
```

Обратите внимание, что методы **performSelector:** всегда возвращают значение типа **id**.

Можно получить класс для данного объекта, пошлав ему сообщение **class**. Это сообщение возвращает класс в виде указателя на объект типа **Class**.

```
Class * cls      = [anObject class];
NSString * clsName = NSStringFromClass ( cls );
```

С другой стороны также можно легко получить соответствующий *class object* по имени класса:

```
Class * cls = NSClassFromString ( clsName );
```

Каждый метод фактически представляет собой функцию с двумя невидимыми аргументами - *self* и *_cmd*.

Первый является аналогом *this*, т.е. указывает на сам объект - получатель сообщения. Второй - содержит селектор данного метода.

Аргумент *self* может использоваться для посылки сообщений самому себе, как например в следующем методе:

```
- (float) area
{
```

```

    return [self width] * [self height];
}

```

Однако кроме *self* есть еще одна величина, которой могут посылаться сообщения - *super*. На самом деле *super* не является нормальной переменной - это всего лишь еще одно обозначение для указателя на текущий объект. Но при посылке сообщения *super* поиск метода начинается не с *dispatch table* текущего объекта, а с *dispatch table* родительского объекта.

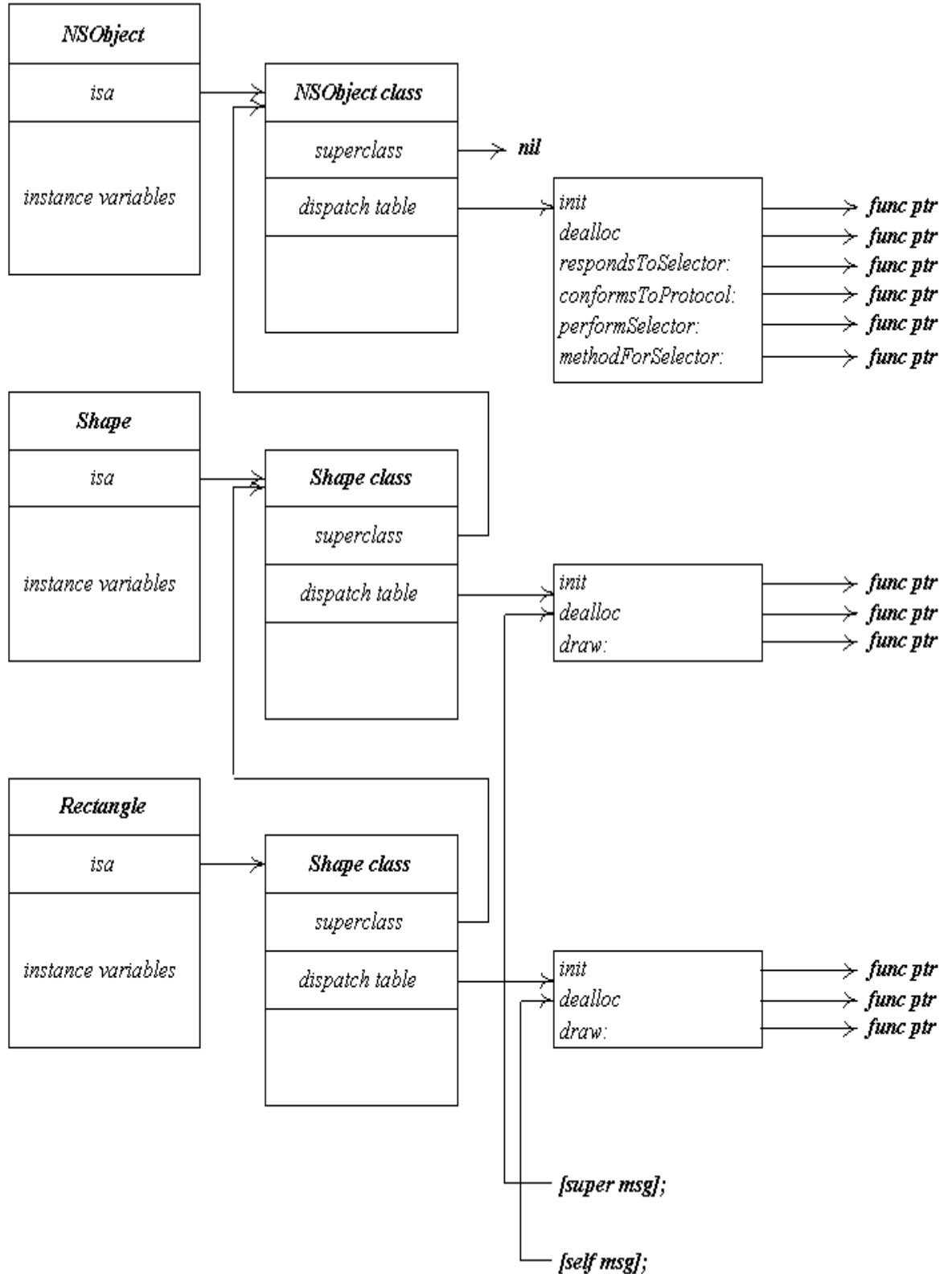


Рисунок 4.2 Разница при посылке сообщения *self* и *super* (для объекта класса *Rectangle*).

Таким образом, при посылке сообщения *super*, вызываются старые версии методов, переопределенные данным классом.

В отличие от языка C++ (в котором несмотря на то, что всем известно, что метод - это просто функция с дополнительным параметром *this*, нет никакого официально документированного способа получить указатель на эту функцию) в языке *Objective-C* можно по селектору метода получить адрес реализующей его функции (именно как функции языка C).

Такая функция отличается от описания метода только вставкой в начала списка аргументов двух дополнительных параметров - указателя на сам объект (*self*) и селектора данного метода (*_cmd*).

Послав объекту сообщение *methodForSelector*: мы получаем в ответ адрес реализующей этот метод функции.

```
typedef float (*WidthFunc)( id, SEL );
typedef void (*SetWidthFunc)( id, SEL, float );

WidthFunc widthFunc = (WidthFunc) [myRect methodForSelector:
@selector (width)];
SetWidthFunc setWidthFunc = (SetWidthFunc) [myRect methodForSelector:
@selector (setWidth:)];

(*setWidthFunc)( myRect, @selector (setWidth:), 27.5f );
```

Это позволяет при необходимости многократного вызова одного и того же метода у заданного объекта полностью избежать всех расходов, связанных с пересылкой сообщений.

4.5 Протоколы

Язык *Objective-C* содержит полноценную поддержку протоколов. Протокол представляет из себя просто список описаний методов. Объект реализует протокол, если он содержит реализации всех методов, описанных в протоколе.

Протоколы удобны тем, что позволяют выделять общие черты у разнородных объектов и передавать информацию об объектах заранее неизвестных классов.

Простейшее описание протокола выглядит следующим образом:

```
@protocol ProtocolName
method declarations
@end
```

Так протокол **Serializable** может быть описан следующим образом:

```
@protocol Serializable
- (id) initWithCoder: (NSCoder *) coder;
- (void) encodeWithCoder: (NSCoder *) coder;
@end
```

Протокол может быть унаследован от произвольного количества других протоколов:

```
@protocol MyProto <Protocol1, Protocol2, Serializable, Drawable>
```

Точно также можно при описании класса задать не только родительский класс, но и набор протоколов:

```
@interface MyClass : SuperClass <Protocol1, Protocol2,
Serializable, Drawable>
```

Для проверки во время выполнения программы поддерживается ли объектом заданный протокол объектов можно использовать сообщение *conformsToProtocol*::

```
if ( [myObject conformsToProtocol: @protocol (Serializable)] )
[myObject encodeWithCoder: myCoder];
```

Кроме того имя протокола (протоколов) можно использовать при описании переменных для явного указания компилятору о поддержке соответствующими объектами протокола (протоколов).

Так если переменная *myObject* содержит указатель на объект заранее неизвестного класса, но при этом удовлетворяющий протоколам **Serializable** и **Drawable**, то ее можно описать следующим образом:

```
id <Serializable,Drawable> myObject;
```

Точно также, если заранее известно, что *myObject* будет содержать указатель на объект, унаследованный от класса **Shape** и поддерживающего протокол **Serializable**, то эту переменную можно описать следующим образом:

```
Shape <Serializable> myObject;
```

Обратите внимание, что подобное описание служит только для сообщения компилятору какие сообщения поддерживает данный объект.

Как и классы, все протоколы в *Objective-C* представлены при помощи объектов (класса **Protocol**):

```
Protocol * myProto = @protocol ( Serializable );
```

Для предварительного объявления протоколов можно использовать следующую конструкцию:

```
@protocol MyProto, Serializable, Drawable;
```

Это конструкция сообщает компилятору о том, что **MyProto**, **Serializable** и **Drawable** являются именами протоколов, которые будут определены позже.

4.6 Обработка исключений

В языке *Objective-C* поддерживается обработка исключений очень похожая на используемую в языках C++ и Java.

Для этого служат директивы **@try**, **@catch**, **@finally** и **@throw**.

```
Cup * cup = [[Cup alloc] init];
```

```
@try
{
    [cup fill];
}
@catch ( NSError * exc )
{
    NSLog ( @"Exception caught: %@", exc );
}
@catch ( id ex )
{
    NSLog ( @"Unknown exception caught" );
}
@finally
{
    [cup release];
}
```

Для запуска исключения используется директива **@throw**, в качестве аргумента берущая указатель на объект-исключение. Обычно в **Mac OS X/NextStep** для этой цели используются объекты класса **NSError**.

```
NSError * exc = [NSError exceptionWithName: @"my-exception"
reason: @"unknown-error"
userInfo: nil];
```

```
@throw exc;
```

Внутри **@catch**-блоков директива **@throw** может использоваться без параметра для повторного запуска обрабатываемого исключения (*rethrowing exception*).

4.7 Синхронизация

Язык *Objective-C* поддерживает синхронизацию для многопоточных приложений. При помощи директивы **@synchronized ()** можно защитить фрагмент кода от одновременного выполнения сразу несколькими потоками.

@synchronized () берет на вход указатель на объект языка *Objective-C* (можно использовать для этой цели любой объект, в том числе и *self*), который играет роль мьютекса(*mutex*).

При попытке потока начать выполнение защищенного фрагмента, проверяется не выполняется ли уже этот фрагмент каким-либо потоком. Если да, то сравниваются объекты, переданные этими потоками в **@synchronized ()**.

Если эти указатели различаются, то поток, пытающийся войти в защищенный блок будет приостановлен (*suspended*) до тех пор, пока первый поток не выйдет из блока. Тогда выполнение второго потока продолжится и уже она "запрет" этот блок для всех остальных потоков.

Наличие подобной возможности заметно облегчает жизнь при написании многопоточковых приложений, когда необходимо отслеживать попытки одновременного изменения одних и тех же данных сразу несколькими потоками.

```
- (void) criticalMethod
{
    @synchronized ( self )
    {
        // perform modifications to shared objects
        . . .
    }
}
```

4.8 Создание и уничтожение объектов

В самом языке *Objective-C* нет специальных команд для создания и уничтожения объектов (подобных **new** и **delete**). Эта задача ложится на *runtime*-библиотеку и реализуется при помощи механизма послышки сообщений.

Реально используемой и наиболее широко распространенной схемой создания и уничтожения объектов в *Objective-C* является используемая в операционных системах **NextStep** и **Mac OS X**, которая и будет описана ниже.

Создание нового объекта разбивается на два шага - выделение памяти и инициализация объекта. Первый шаг реализуется методом класса *alloc* (реализованном в классе **NSObject**), который выделяет необходимое количество памяти (данный метод используется для выделения памяти не только для объектов класса **NSObject**, но и любого унаследованного от него класса). При этом выделяемая память обнуляется и в атрибут *isa* записывается указатель на *class object* соответствующего класса.

Обратите внимание, что сообщение *alloc* посылается *class object*-у требуемого класса и это сообщение возвращает указатель на выделенную под объект память.

Собственно сама инициализация объекта (т.е. установка значений его *instance*-переменных, выделение дополнительных ресурсов и т.п.) осуществляется другими методами, по традиции имена этих методов начинаются с *init*. Обычно такое сообщение посылается сразу же после сообщения *alloc*, по адресу, возвращенному этим сообщением.

```
id anObject = [[Rectangle alloc] init];
```

Приведенная выше конструкция является правильным способом создания объекта. Обратите внимание, что следующая конструкция может в ряде случаев не работать:

```
id anObject = [Rectangle alloc];
```

```
[anObject init];
```

Это связано с тем, что для ряда классов метод *init* может вернуть совсем другой указатель (а не *self*).

Простейшими примерами того, когда может возникать подобная ситуация, являются синглтоны (тогда, если один экземпляр класса уже существует, то метод *init* освободит выделенную *alloc*-ом память и вернет указатель на уже созданный единственный экземпляр) и кэширование объектов, когда для увеличения производительности, выделение объектов происходит сразу блоками и объекты не уничтожаются, а сохраняются для переиспользования.

При создании нового класса обычно нет необходимости переопределять метод *alloc*, а вот необходимость переопределения метода *init* возникает достаточно часто (хотя во многих случаях можно положиться на обнуление памяти *alloc*'ом).

Обратите внимание, что метод(ы) *init* является обычным методом, ничем не выделяющимся среди остальных (в отличии от C++, где конструктор - это особый метод, у которого например нельзя взять адрес).

Поэтому при создании нового класса и метода *init* вызов переопределенного метода *init* (при помощи *[super init]*) должен быть произведен явно в самом начале метода.

Довольно часто у объектов бывает сразу несколько методов, начинающихся с *init*, например *initWithName:*, *initWithContentsOfFile:* и т.д.

Установившейся практикой в таком случае является выделение среди всех *init*-методов одного, называемого *designated initializer*. Все остальные *init*-методы должны вызывать его и только он вызывает унаследованный *init* метод.

```
- initWithName: (const char *) theName // designated initializer
{
    [super init];                      // call inherited method

    name = strdup ( theName );
}

- init
{
    return [self initWithName: ""];
}
```

В ряде случаев оказывается удобным совместить выделение памяти и инициализацию объекта в один метод (класса), например в классе **NSString** есть ряд методов класса, возвращающих уже готовый (проинициализированный) объект:

```
+ (NSString *) initWithCString: (const char *) str;
+ (NSString *) initWithFormat: (NSString *) format, ...;
```

Mac OS X (как и **NextStep**) для управления временем жизни объектов используют *reference counting* - каждый объект содержит внутри себя некоторый счетчик, при создании устанавливаемый в единицу.

Посылка объекту сообщения *retain* увеличивает значение этого счетчика на единицу (так все контейнерные классы библиотеки **Foundation** при помещении в них объекта, посылают ему сообщение *retain*).

Установившейся практикой является посылка объекту сообщения *retain* всеми, заинтересованными в нем сторонами (объектами), т.е. если вы запоминаете ссылку на объект, то следует послать ему сообщение *retain*.

Когда объект перестает быть нужен, то ему просто посылается сообщение *release*.

Данное сообщение уменьшает значение счетчика на единицу и, если это значение стало меньше единицы, уничтожает данный объект.

Перед уничтожением объекта ему посылается сообщение *dealloc*, позволяющее объекту произвести свою деинициализацию. При этом это также является обычным сообщением и в нем Вы явно должны в конце вызвать унаследованную реализацию через *[super dealloc]*.

```
- (void) dealloc
{
    . . .
    [super dealloc];
}
```

4.9 Категории

Язык **Objective-C** обладает крайне редко встречающейся возможностью добавлять новые методы к уже существующим классам. При этом не требуется исходных текстов класса и добавленные методы автоматически становятся доступными всем классам, унаследованным от изменяемого.

Так можно добавить новый метод классу **NSObject** и этот метод автоматически добавится во все остальные классы.

Аналогичной возможностью обладает язык **Ruby**, но **Objective-C** является компилируемым языком, а **Ruby** - интерпретируемым.

Механизм, позволяющий расширять уже существующие классы (путем добавление новых методов, новые *instance*-переменные добавить таким образом нельзя), называется категориями.

Категория имеет свое имя, список методов и имя класса, который она расширяет. Описание категории имеет следующий вид:

```
#import "ClassName.h"
@interface ClassName ( CategoryName )
    methods declarations
@end
Реализация категории выглядит следующим образом:
#import "CategoryName.h"
@interface ClassName ( CategoryName )
    methods bodies
@end
```

4.10 Объекты Class objects и Objective-C runtime

При компиляции программы на **Objective-C** компилятор для каждого введенного класса автоматически создает так называемый *class object* - полноценный объект, содержащий в себе всю информацию о данном классе, включая название, суперкласс, список методов и *instance*-переменных.

При этом такой объект является полноценным объектом, т. е. ему можно посылать сообщения, передавать в качестве параметра.

Одной из особенностей *class object'a* является то, что он поддерживает все методы класса **NSObject**, т.е. когда ему посылается сообщение, то сначала идет поиск среди методов класса, а если метод не найден, то поиск продолжается среди *instance*-методов класса **NSObject**.

Еще одной интересной особенностью является возможность инициализации *class object'ов* - в начале работы приложения каждому *class object'y* посылается сообщение (класса) *initialize*.

Это сообщение гарантированно посылается каждому *class object'y*, причем всего один раз и до того, как ему будет послано любое другое сообщение. Простейшим примером применения такого сообщения является реализация *Singleton'ов* - именно в методе *initialize* следует создать тот самый единственный экземпляр объекта и запомнить его в *static*-переменной.

Если посмотреть на **Objective-C runtime** от **Apple**, то мы найдем большое количество **С-функций**, служащих для работы с классами (непосредственно во время выполнения программы).

Наиболее интересными являются следующие:

```
Method  class_getInstanceMethod( Class aClass, SEL aSelector );
Method  class_getClassMethod   ( Class aClass, SEL aSelector );
struct objc_method_list * class_nextMethodList(Class theClass, void **
iterator);
void     class_addMethods      ( Class aClass, struct objc_method_list
* methodList );
void     class_removeMethods   ( Class aClass, struct objc_method_list
* methodList );
unsigned method_getNumberOfArguments ( Method method );
unsigned method_getSizeOfArguments  ( Method method );
unsigned method_getArgumentInfo     ( Method method, int argIndex,
const
char ** type, int * offset );
Ivar     class_getInstanceVariable ( Class aClass, const char *
aVariableName );
```


Функция **class_getInstanceMethod** возвращает указатель на структуру (*objc_method*) описывающую заданный instance-метод данного класса.

Функция **class_getClassMethod** возвращает указатель на структуру (*objc_method*) описывающую заданный метод данного класса.

Функция **class_nextMethodList** возвращает один из списков методов для заданного класса. Приводимый ниже фрагмент кода позволяет перебрать все методы для данного класса.

```
void                                * iterator = 0;
struct objc_method_list * mlist;

//
// Each call to class_nextMethodList returns one methodList
//

methodList = class_nextMethodList( classObject, &iterator )

while ( methodList != NULL )
{
    // ...do something with the method list here...

    methodList = class_nextMethodList ( classObject, &iterator );
}
```

Функция **class_addMethods** позволяет добавлять новые методы к заданному классу.

Функция **class_removeMethods** позволяет убирать методы из заданного класса.

Функция **method_getNumberOfArguments**. Возвращает количество аргументов для заданного метода.

Функция **method_getSizeOfArguments** возвращает размер места на стеке, занимаемого всею аргументами данного метода.

Функция **method_getArgumentInfo** возвращает информацию об одном из аргументов для заданного метода..

Функция **class_getInstanceVariable** возвращает информацию об instance-переменной класса в виде указателя на структуру *objc_ivar*.

Для кодирования информации о типах используется специальное строковое представление, однозначно сопоставляющее каждому типу данных некоторую строку. Явно получить такую строку для произвольного типа можно при помощи конструкции **@encode ()**.

```
char * buf1 = @encode ( int ** );
char * buf2 = @encode ( struct key );
char * buf3 = @encode ( Rectangle );
```

4.11 Язык Objective-C 2.0

В **Mac OS X 10.5 Leopard** вошла новая версия языка - **Objective-C 2.0**, в которую было добавлено много элементов, упрощающих использование языка. В частности именно эта версия (2.0) используется для разработки приложений под платформу **iPhone/iPod Touch**.

Одним из наиболее значительных изменений является опциональная поддержка сборки мусора (*garbage collection*). Для подключения этой возможности достаточно просто задать в свойствах проекта флаг использования сборки мусора. Т.е. это необязательная опция - каждый сам может решать стоит ли ему ее использовать или нет. Кроме того, любое корректно написанное приложение (использующее библиотеки **Apple**), работающее без *garbage collection*, будучи откомпилированным с этой опцией, также будет полностью корректно работать. Следует правда отметить, что для разработки приложений под платформу **iPhone/iPod Touch** *garbage collection* пока не поддерживается.

Также в **Objective-C 2.0** вошел более привычный синтаксис для доступа к свойствам (properties) объектов, используя точку как для доступа к полям структуры в **C/C++/Java**.

Пусть есть класс **Person**, описанный ниже.

```
#import <Cocoa/Cocoa.h>
```

```
@interface Person : NSObject
{
    NSString * firstName;
    NSString * lastName;
}

- (NSString *) firstName;
- (void) setFirstName: (NSString *) newFirstName;
- (NSString *) lastName;
- (void) setLastName: (NSString *) newFirstName;

@end
```

Тогда, если у нас переменная *bill* является указателем на объект класса **Person**, то следующие конструкции для доступа к полям (свойствам) этого объекта полностью законны.

```
NSLog ( @"First name is %@", bill.firstName );
```

```
bill.firstName = @"Bill"
bill.lastName = @"Gates";
```

При этом на самом деле подобный синтаксис является всего лишь *syntax sugar* и переводится компилятором в нормальный с использованием *set/get*-методов:

```
NSLog ( @"First name is %@", [bill firstName] );
```

```
[bill setFirstName: @"Bill]"
[bill setLastName: @"Gates"];
```

Очень удобной возможностью **Objective-C 2.0** стала поддержка автоматического создания *set/get*-методов. Реализации этих методов обычно крайне просты и однообразны, но отнимают время и силы программиста. Поэтому эта работа легко может быть переложена на компилятор.

Для этого необходимо в разделе **@interface** описать каждое свойство, методы доступа к которому должны быть созданы при помощи директивы **@property**, а затем в разделе **@implementation** использовать директиву **@synthesize** или **@dynamic**.

Директива **@property** имеет следующую структуру:

```
@property (attributes) type name;
```

Через *type* и *name* обозначены тип значения и имя свойства. В круглых скобках задается список атрибутов, задающих компилятору какие именно *set/get*-методы нужно создать.

Допустимы следующие атрибуты:

```
getter=getterName - явно задать имя get-метода;
setter=setterName - явно задать имя set-метода;
readonly - для данного свойства следует создать только get-метод;
readwrite - для данного свойства следует создать и set- и get-методы (по умолчанию);
assign - set-метод делает простое присваивание значений;
retain - set-метод делает присваивание значений с использованием retain/release;
copy - set-метод делает простое присваивание копии исходного значения;
atomic - обеспечить атомарность доступа (имеет смысл только для многопоточных приложений);
nonatomic - не надо обеспечивать атомарность (по умолчанию).
```

Обратите внимание, что опции *readonly/readwrite* являются взаимоисключающими. Также взаимоисключающими являются опции *assign/retain/copy*.

Ниже приводятся реализации set-метода для каждого из трех опций - *assign/retain/copy*.

```
        // assign
- (void) setFirstName: (NSString *) newName
{
    firstName = newName;
}

        // retain
- (void) setFirstName: (NSString *) newName
{
    if ( firstName != newName )
    {
        [firstName release];

        firstName = [newName retain];
    }
}

        // copy
- (void) setFirstName: (NSString *) newName
{
    if ( firstName != newName )
    {
        [firstName release];

        firstName = [newName copy];
    }
}
```

Обратите внимание, что для корректной работы со счетчиками числа ссылок при работе с объектами следует использовать либо *retain*, либо *copy*. При этом в первом случае будет присвоена ссылка на тот же объект, а в последнем - ссылка на независимо существующую копию.

С использованием директивы **@property** описание класса **Person** может выглядеть следующим образом:

```
#import <Cocoa/Cocoa.h>
```

```
@interface Person : NSObject
{
    NSString * firstName;
    NSString * lastName;
}

@property (nonatomic,retain) NSString * firstName;
@property (nonatomic,retain) NSString * lastName;
```

```
@end
```

Для создания реализации set/get-методов в разделе @implementation следует использовать директиву @synthesize:

```
#import "Person.h"
```

```
@implementation Person
```

```
- (void) dealloc
{
    [firstName autorelease];
    [lastName autorelease];
}
```

```
@synthesize firstName;
```

```
@synthesize lastName;
```

```
@end
```

Вместо директивы **@synthesize** можно использовать директиву **@dynamic** (по умолчанию), говорящую, что *set/get*-методы Вы зададите сами.

Еще одним нововведением языка **Objective-C 2.0** стало упрощенная форма цикла по содержимому контейнера. Так если *array* - это объект класса, унаследованного от **NSArray**, а *dict* - объект класса, унаследованного от **NSDictionary**, то для перебора элементов в них можно использовать следующие фрагменты кода:

```
for ( NSString * str in array )  
    NSLog ( @"Element: %@", str );
```

```
for ( id key in dict )  
    NSLog ( @"Key: %@ Value: %@", id, [dict valueForKey: key] );
```

Для того, чтобы можно было подобным образом итерировать по контейнеру он должен поддерживать протокол **NSFastEnumeration** (все стандартные контейнеры его поддерживают). Большим плюсом подобного перебора элементов является то, что во время перебора любая попытка изменить контейнер приведет к выбрасыванию исключения, т.е. это безопасный перебор.

5 Структура программы и основные типы данных языка Objective-C

5.1 Структура программы

Программа, составленная на языке Си, может содержать одну или более функций, которые представляют собой ряд последовательных инструкций. Многие функции (к примеру, извлечение квадратного корня), уже написаны и находятся в библиотеках (например, в библиотеках, входящих в iOS SDK) [1].

Все программы на Си должны начинаться с функции, называемой `main()`. Структура функции:

```
function(параметры)
{
    Начало функции
    ...;          Здесь помещаются инструкции, которые могут использовать
«параметры»
}                Здесь функция заканчивается
```

Из функции мы можем вызывать другую функцию. Комментарии отделяются либо знаком “//”, и весь текст до конца строки считается комментарием, либо помещаются между /* и */. Тогда весь текст внутри данных символов считается комментарием.

Точку с запятой “;” необходимо ставить после каждой отдельной инструкции.

5.2 Типы данных

Рассмотрим четыре типа данных, которые будем использовать:

int: целые числа от -32768 до $+32767$

float: число с плавающей точкой от $-3.4E-38$ до $3.4E+38$

double: число с плавающей точкой от $1.7E-308$ до $1.7E+308$

char: символьная переменная, которая может хранить один символ — букву, цифру или другой символ.

Переменную, объявленную с типом **int**, можно использовать только для хранения целых значений. Переменную, которая объявлена с типом **float**, можно использовать для хранения чисел с плавающей точкой (то есть значений, имеющих дробную часть). Тип **double** тоже хранит числа с плавающей точкой, но двойной точностью. Тип данных **char** можно использовать для хранения одного символа, например, буквы «а», цифры «6», знака «;» (подробнее об этом см. ниже) [2].

Тип int.

В Objective-C целая константа состоит из одной или нескольких цифр. Знак «минус» перед цифрами указывает, что это отрицательное значение. 158, -10 и 0 — примеры целых констант. Между цифрами не допускаются пробелы, и значения между тройками цифр нельзя разделять запятыми. (То есть число 12,000 должно быть записано как 12000.)

В Objective-C имеется два специальных формата записи целых констант с основанием, отличным от 10. Если первая цифра целого значения равна 0, это значение считается восьмеричным, octal (по основанию 8). В этом случае остальные цифры должны быть допустимыми восьмеричными цифрами от 0 до 7. Например, чтобы представить в Objective-C восьмеричное значение 50 (эквивалентное десятичному значению 40), используется форма записи 050. Восьмеричная константа 0177 эквивалентна десятичному значению

127 (1 * 64 + 7 * 8 + 7).

Целое значение можно вывести на экран в восьмеричной записи, вызвав NSLog и указав символы формата %o в строке форматирования. Значение выводится в восьмеричной записи без ведущего нуля. Символы формата %#o выводят ведущий нуль перед восьмеричным значением. Если перед целой константой ставятся 0 и буква x, то значение считается шестнадцатеричным (по основанию 16). После буквы x следуют цифры шестнадцатеричного значения (это цифры 0-9 и буквы a-f разного

регистра). Буквы представляют, соответственно, десятичные числа от 10 до 15. Например, чтобы присвоить шестнадцатеричное значение FFEF0D целой переменной с именем rgbColor, можно использовать оператор:

```
rgbColor = 0xFFEF0D;
```

Символы форматирования %x выводят значения в шестнадцатеричном формате без ведущих символов 0x и с использованием строчных букв a-f. Чтобы вывести значение с ведущими символами 0x, нужно использовать символы форматирования %#x:

```
NSLog(@"»This color is %#x\n», rgbColor);
```

Для вывода ведущего символа x и шестнадцатеричных цифр прописными буквами в символах форматирования ставится прописная буква X: %X или %#X. Каждое значение, будь то символ, целое число или число с плавающей точкой, имеет связанный с ним диапазон значений. Этот диапазон связан с количеством памяти, выделяемой для хранения определенного типа данных. Обычно он зависит от компьютера на котором выполняется работа, и поэтому называется машинно-зависимым диапазоном (machine dependment). Так, целое значение может занимать на вашем компьютере 32 или 64 бита.

Никогда не пишите программы, в которых предполагается определенный размер типов данных. Для каждого базового типа данных гарантируются определенные размеры памяти, например, для целого значения отводится не менее 32 бит памяти. Однако на некоторых машинах это не выполняется.

Тип float.

Для хранения значений, содержащих знаки после запятой (точки), можно использовать переменные с типом float. Признаком константы с плавающей точкой является присутствие десятичной точки. Можно не указывать цифры до десятичной точки или после нее, но, очевидно, нельзя указывать одновременно обе составляющие. В NSLog для вывода значения с плавающей точкой используются символы формата %f, для вывода значения в экспоненциальном представлении, в строке формата NSLog должны быть заданы символы форматирования %e. Если использовать символы формата %g, то процедура NSLog, сама определит как выводить значения с плавающей точкой: в обычном режиме записи, или же в экспоненциальном представлении.

Тип double.

Тип double аналогичен типу float. Он используется, если диапазон, представляемый переменной типа float, не дает достаточной точности. В переменных типа double можно хранить примерно вдвое больше цифр, чем в переменных типа float. На большинстве ПК для значений типа double выделяют 64 бита. Для вывода значений типа double можно использовать символы формата: %f; %e; %g; как и для значений типа float.

Тип char

Переменную типа char, можно использовать для хранения одного символа. Чтобы задать символ константу, нужно заключить символ в апострофы. Не следует путать символ-константу с символьной строкой в стиле языка C, которая может хранить в себе любое кол-во символов заключенных в кавычки. Строка символов, заключенная в кавычки, перед которой поставлен символ @, — это объект — строка типа NSString.

Символ-константа '\n' (символ новой строки) является допустимым символом-константой. Обратный слэш (обратная наклонная черта) — это специальный символ в системе Objective-C, который не учитывается как отдельный символ. Имеются и другие специальные символы: которые иницируются с помощью обратного слэша. Для вывода значения переменной типа char можно использовать символы формата %c при вызове NSLog.

Пример использования базовых типов данных:

```
?  
#import
```

```

int main(int argc, char *argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc]
init];

    int a_int = 10;
    float a_float = 21.22;
    double a_double = 8.44e+11;
    char a_char = 'Q';

    NSLog(@"integer = %i", a_int);
    NSLog(@"float = %f", a_float);
    NSLog(@"double = %e", a_double);
    NSLog(@"char = %c", a_char);

    [pool release];
    return 0;
}

```

Вывод программы:

```

integer = 10;
float = 21.220000
double = 8.440000e+11;
char = 'Q';

```

Тип id.

Тип данных id используется для хранения объекта любого типа. Его можно назвать обобщенным типом объекта. Ниже объявляется переменная number с типом id.

id number;

Можно так же объявлять методы которые возвращают значения типа id.

-(id) newObject: (int) type;

Здесь объявляется метод экземпляра с именем newObject, который принимает один целый аргумент с именем type и возвращает значение типа id. id — это тип по умолчанию для возвращаемого значения и типа аргумента. Тип id — является основой таких важных средств языка Objective-C, как полиморфизм и динамическое связывание.

5.3 Переменные и константы

Переменная задается <тип данных> название_переменной:

```

int yearsOld;
float tempCelcium;

```

В языке Си в названии переменной допускаются латинские буквы, цифры и символ подчеркивания, но не допускаются пробелы, однако в программах на **Objective-C** используется camelStyle, когда переменная начинается с прописной буквы, а каждое слово в названии переменной — с заглавной. Его мы и будем использовать сегодня в примерах.

Использование переменных в Objectiv-C не чем не отличается от использования их в других языках. (Думаю не придется объяснять что такое переменная и для чего она нужна.)

Для примера рассмотрим программу вывода суммы 2х чисел, для этого используем 2 переменных (x,y) типа Integer — то есть наши переменные будут целочисленными. Исходный текст программы должен выглядеть следующим образом:

```

?
#import <Foundation/Foundation.h>

int main (int argc, const char * argv[])
{

```

```

        NSAutoreleasePool * pool = [[NSAutoreleasePool alloc]
init];

        int x=5, y=7;
        x=x+y;

        NSLog(@"Сумма 2х переменных равна:  %i", x);
        [pool drain];
        return 0;
    }

```

Обратите внимание на то, что переменные объявляются через запятую, ключевое слово `int` — это сокращение от `integer`. В 6-ой строке мы присваиваем переменной `x` сумму двух переменных — `x` и `y`. При выводе результата в строке 7, мы используем символ формата «`%i`» — он указывает на место в котором должно быть выведено значение переменной. Для каждого типа переменной существует свой символ формата, основные форматы были рассмотрены выше.

Теперь попробуйте изменить программу так, что бы складывались 2 дробных значения, используйте тип `float`, так как он применяется для хранения значений с «плавающей точкой» то есть дробных значений. Исходный код должен будет выглядеть примерно следующим образом:

```

#import <Foundation/Foundation.h>

int main (int argc, const char * argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc]
init];

    float x=5.3, y=7.4;
    x=x+y;

    NSLog(@"Сумма 2х переменных равна:  %f", x);
    [pool drain];
    return 0;
}

```

Обратите внимание, что для отделения дробной части нужно использовать точку, а не запятую! Это ошибка очень часто встречается на первых порах.

Попробуйте воспроизвести более сложные операции деления, умножения, вычитания. Так же попробуйте объединить их в более сложное выражение.

Константа задается с помощью директивы `#define ИМЯ_КОНСТАНТЫ`, помещенной перед функцией `main()`. При этом имя константы состоит из заглавных букв:

```
#define NUMBER_PI 3.14
```

Далее в программе мы сможем использовать `NUMBER_PI` вместо значения 3.14. Если переменной можно при выполнении программы присваивать различные значение, то константа имеет постоянное значение.

В Objective-C любое число, символ или строка символов называется константой (`constant`). Например, число 58 является целым значением-константой. Строка `@»Hello world\n»` — это пример объекта — константы со строкой — символов. Выражение, содержащее только значения-константы, называется выражением-константой, или константным выражением (`constant expression`).

Следующее выражение является константным, поскольку каждый из членов этого выражения является константой: `128 + 7- 17`; Но если `i` объявлена как переменная целого типа, то это выражение уже не будет константным: `128 + 7- i`;

```
#import <Foundation/Foundation.h>
```

```

int main(int argc, char *argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc]

```



```

init];

    int i = 0, b = 128;

    //Константное вырожение...
    i = 128 + 7 - 17;
    NSLog(@"i = %i", i);

    //Не константное вырожение...
    i = b + 7 - 17;
    NSLog(@"i = %i", i);

    [pool release];
    return 0;
}

```

5.4 Массивы (Arrays)

Массив — это множество объектов, которыми можно оперировать как группой. В Си массив может содержать значения одного типа. Для определения массива мы указываем тип значений и размер массива:

```
int someNumbers[5]; //массив из 5ти элементов от 0 до 4 типа int.
```

Нумерация элементов в массиве начинается с 0. Ввод значений в массив может осуществляться как по-отдельности:

```
someNumbers[0] = 34; //теперь нулевой элемент равен 34,
```

либо сразу всем элементам при определении массива:

```
int someNumbers[5] = {34, 45, 12, 98, 12};
```

Строка — это массив символьных значений:

```
char someName[20] = "Иванов Иван Иванович";
```

5.5 Структуры

Структура — это взаимосвязанное множество переменных. Определяется структура ключевым словом **struct**:

```

struct название_структуры {
    тип_данных название_элемента_структуры;
    тип_данных название_элемента_структуры_2;
    тип_данных название_элемента_структуры_3;
};

```

Например,

```

struct CD {
    char name[20];
    char description[40];
    float cost;
    int year;
};

```

Теперь мы можем создать переменную disc:

```
struct CD disc = {"Избранное", "Мой playlist из iPod", 125.30, 2011};
```

Для того, чтобы обратиться к члену структуры, необходимо указать его вместе с именем переменной используя следующий формат:

```
disc.name; //мы указали имя структурной переменной, поставили точку и указали имя элемента структуры, в результате получили значение "Избранное".
```

5.6 Указатели

Переменная занимает какое-то место в памяти и имеет определенный адрес в памяти. Когда мы обращаемся к ней с помощью имени, нам возвращается значение этой переменной.

Однако, мы можем получить адрес переменной в памяти, где хранится её значение, используя оператор получения адреса (& — амперсанд) перед названием переменной:

```
addressOfName = &name; //получаем адрес переменной name.
```

Указателем называется переменная, которая содержит значение адреса элемента памяти, где хранится значение другой переменной. Для создания указателя используется синтаксис:

```
type *variableName;  
type – тип переменной, адрес которой будет содержаться в указателе.  
* – звездочка определяет указатель.  
variableName – имя переменной типа «указатель».
```

С помощью указателя мы можем не только получить адрес какой-либо переменной, но также и ее значение:

```
int *taxPointer;  
int tax;  
taxPointer = &tax; //мы получили значение адреса переменной tax с  
помощью &  
tax = 25; //присвоим значение переменной tax  
напечатать taxPointer; //напечатается адрес переменной tax  
напечатать *taxPointer; //напечатается значение 25  
tax = 40;  
напечатать *taxPointer; //напечатается значение 40
```

Таким образом, мы можем получать значение переменной `tax`, зная ее адрес в памяти, и в случае изменения значения переменной `tax` мы получаем ее последнее значение, так как `taxPointer` хранит не значение переменной `tax`, а её адрес.

На этом мы заканчиваем обзорную статью по языку программирования Си и в следующей части статьи, которая выйдет завтра, перейдем к изучению ООП и синтаксису Objective-C.

6 Основные операторы языка Objective-C

6.1 Операторы

В Си используются следующие основные операторы:

+ сложение

– вычитание

* умножение

/ деление

% получение остатка от деления нацело (например, $13\%6 = 1$, или $178/12 = 10$, или $21\%7 = 0$) [1].

Также используются следующие операторы присваивания:

variable = 3 + 5; //операция присваивания, в результате которой **variable = 8**.

Существуют сокращенные операторы присваивания:

variable += 5; эквивалентна **variable = variable + 5;**

variable -= 5; эквивалентна **variable = variable – 5;**

variable *= 5; эквивалентна **variable = variable * 5;**

variable /= 5; эквивалентна **variable = variable / 5;**

variable %= 5; эквивалентна **variable = variable % 5;**

6.2 Условия

6.2.1 Инструкция if

В любом языке программирования имеется возможность принятия решений. В языке программирования **Objective-C** имеются несколько конструкций для принятия решений [2]:

- оператор **if**;

- оператор **switch**;

- оператор **if-else**.

Рассмотрим инструкции, используемые для построения логики программы.

Проверяет условие, и если оно выполняется — выполняет участок кода:

```
if (условие)
{
...; //инструкции
}
```

для проверки условия используются:

a == b — равно

a > b — больше

a < b — меньше

a >= b — больше или равно

a <= b — меньше или равно

Если условие не выполняется, то содержимое скобок {...;} игнорируется.

В общем виде оператор **if** имеет следующий формат:

```
if ( выражение )
```

программный оператор

Представим, что нам нужно преобразовать утверждение «Если нет дождя, я пойду купаться» в операторы языка **Objective-C**. Используя показанный выше формат, это можно «записать» следующим образом.

```
if ( нет дождя )
```

я пойду купаться

Оператор **if** используется, чтобы выполнить программный оператор в зависимости от указанных условий (или операторы, если они заключены в фигурные скобки). Аналогичным образом, в программном операторе сообщение *maxExceeded* отправляется

playlist (список воспроизведения), только если значение переменной *count* больше значения *MAXIMUM_SONGS*, иначе этот оператор игнорируется.

```
if ( count > MAXIMUM_SONGS )
```

```
[playlist maxExceeded];
```

Рассмотрим пример. Предположим, что вам нужно написать программу, которая принимает целое значение, введенное с клавиатуры, и затем выводит абсолютное значение этого целого числа. Проще всего вычислить абсолютное значение числа — взять его с противоположным знаком, если оно меньше нуля. Фраза «если оно меньше нуля» означает, что программа должна принять решение. Для этого можно использовать оператор **if**, как показано в примере ниже.

```
#import <Foundation/Foundation.h>
```

```
int main (int argc, char *argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];

    int number;

    NSLog (@"Type in your number: ");

    scanf ("%i", &number);

    if ( number < 0 )

        number = -number;

    NSLog (@"The absolute value is %i", number);

    [pool drain];

    return 0;
}
```

После вывода сообщения для пользователя и сохранения введенного целого значения в переменной *number* программа проверяет, не является ли значение *number* отрицательным. Если да, то выполняется следующий программный оператор, который изменяет знак *number* на противоположный. Если значение *number* не меньше нуля, этот оператор автоматически пропускается (в этом случае не нужно изменять знак на противоположный). Затем выполняется вывод значения *number* и завершается выполнение программы.

6.2.2 Конструкция if-else

Однако, можно задать выполнение других инструкций в случае невыполнения условия с помощью конструкции **if...else**:

```
if(a >=0) {
    float result = sqrt(a);
    return result;
} else {
    float result = sqrt(-a);
    return result;
}
```

Таким образом, мы извлекли квадратный корень с помощью Си функции *sqrt*, но перед этим проверили чтобы значение *a* было положительным, если же оно отрицательное, мы извлекли *sqrt* из “-a”.

Если нужно сказать, является ли число четным или нечетным, вы, скорее всего, определите это по последней цифре числа. Если эта цифра равна 0, 2, 4, 6 или 8, значит, число является четным. На Компьютере это можно сделать проще: достаточно проверить, делится ли это число на 2 без остатка. Если да, то число является четным, иначе оно является нечетным. Вы уже видели, как использовать оператор взятия по модулю (%) для вычисления остатка от деления одного целого числа на другое. Его удобно использовать, чтобы определить, делится ли число на 2 без остатка. Если остаток от деления на 2 равен 0, значит, это четное число, если нет — нечетное.

Напишем программу, которая определяет, является ли целое значение, которое вводит пользователь, четным или нечетным, и выводит соответствующее значение на терминам.

```
#import <Foundation/Foundation.h>

int main (int argc, char *argv[])
{
    NSAutoreleasePool * pool = ((NSAutoreleasePool alloc)
init];

    int number_to_test, remainder;

    NSLog (@"Введите число для проверки: ");
    scanf ("%i", &number_to_test);

    remainder = number_to_test % 2;

    if ( remainder == 0 )
        NSLog (@"Число является четным.");

    if ( remainder != 0 )
        NSLog (@"Число не является четным.");

    [pool drain];
    return 0;
}
```

После ввода числа вычисляется остаток от деления на 2. Первый оператор **if** проверяет, равно ли нулю значение этого остатка. Если да, то выводится сообщение «Число является четным.». Второй оператор **if** проверяет значение этого остатка на неравенство нулю. Если да, то выводится сообщение, что число является нечетным. Если выполняется первый оператор **if**, то второй оператор не должен выполняться, и наоборот. При написании программ концепция «иначе» (**else**), требуется настолько часто, что почти во всех современных языках программирования введена специальная конструкция для этой ситуации. В **Objective-C** это конструкция **if-else**, которая имеет следующий формат [2].

```
if (выражение)
    программный
оператор 1
else
    программный
оператор 2
```

На самом деле **if-else** является просто расширением общего формата оператора **if**. Если результатом оценки выражения является значение *TRUE*, то выполняется программный оператор 1; иначе выполняется программный оператор 2. Во всех случаях выполняется программный оператор 1 или программный оператор 2, но не оба оператора.

Теперь давайте вновь вернемся к программе определения четности пользовательского числа, и изменим ее код выше с учетом использования оператора **if-else**.

```
#import <Foundation/Foundation.h>
int main (int argc, char *argv[])
{
    NSAutoreleasePool * pool = ((NSAutoreleasePool alloc]
init];

    int number_to_test, remainder;

    NSLog (@"Введите число для проверки: ");
    scanf ("%i", &number_to_test);

    remainder = number_to_test % 2;

    if ( remainder == 0 )
        NSLog (@"Число является четным.");
    else
        NSLog (@"Число не является четным.");

    [pool drain];
    return 0;
}
```

Не сложно заметить что вариант программы с использованием конструкции **if-else**, является более читаемым, логичным и удобным по сравнению с использованием одного оператора **if**. Операторы **if**, как и конструкции вида **if-else**, могут быть включены друг в друга, для более решения более сложных логических задач.

6.2.3 Конструкция switch/case/default

Если нам нужно учесть более трех возможных вариантов, конструкция **if** может оказаться слишком перегруженной. Для таких случаев целесообразнее применить конструкцию **switch/case/default**:

```
errorCode = какой-то код ошибки от 1 до 10;
switch(errorCode) {
case '1':
    //ошибка №1: неправильно...
    break;
case '2':
    //ошибка №2: убедитесь...
    break;
.....
..... //другие case '#':
.....
case '10':
    //ошибка №10: попробуйте выполнить действия...
    break;
default:
    //неизвестная ошибка
}
```

Таким образом значение **errorCode** сравнивается со значениями, стоящими после **case** и в соответствие с ними выполняются определенные инструкции. **break** говорит о прекращении цикла после того, как соответствие найдено. **default** содержит инструкции на случай, если ни одного совпадения значение **errorCode** и аргумента после **case** не будет найдено.

6.3 Циклы

Язык **Objective-C** позволяет повторять последовательности кода различными способами. Они могут формироваться с помощью следующих операторов:

- оператор **for**;

- оператор **while**;
- оператор **do**.

Начнем с простого примера: подсчет суммы. Составим треугольник из 15 шаров. Пример треугольной компоновки: Первый ряд состоит из одного шара, второй ряд — из двух, и т.д. Количество шаров в треугольнике, состоящем из n рядов, равно сумме чисел от 1 до n . Эта сумма называется треугольным числом. Если начать с 1, то четвертое треугольное число будет равно сумме последовательных чисел от 1 до 4 ($1 + 2 + 3 + 4$), то есть 10. Напишем программу которая вычисляет и выводит на экран значение восьмого треугольного числа. Это число легко сосчитать в уме, но можно написать программу на Objective-C, чтобы выполнить эту задачу:

```
#import <Foundation/Foundation.h>

int main (int argc, char *argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc]
init];

    int triangularNumber;

    triangularNumber = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8;
    NSLog (@\"The eighth triangular number is %i\",
triangularNumber);

    [pool drain];
    return 0;
}
```

Этот способ вполне подходит для относительно небольших треугольных чисел, но что делать, если нам нужно вычислить, например, значение 200-го треугольного числа? Не складывать же в явном виде все целые числа от 1 до 200? К счастью, имеется более простой способ — возможность выполнения операций в цикле. Для реализации циклов в языке **Objective-C** имеются перечисленные выше три оператора.

6.3.1 Цикл for

Данный цикл используется когда известно точное количество повторов, которое нужно выполнить:

```
for (repeat = 1; repeat <= 10; repeat++) {
    //выполняется какой-то код, например печать значение repeat
}
```

в результате мы получим числа от 1 до 10.

Аргументы в *for* (*инициализируем начальное значение; указываем условие; устанавливаем приращение*).

Цикл **do...while**

Аналогичен циклу **for**, только условие проверяется в конце, таким образом цикл выполнится как минимум один раз:

```
do
{
    int stopAtTen;
    stopAtTen = 1;
    stopAtTen++;
    //печать stopAtTen
}
while (stopAtTen <=10);
```

В результате напечатаются числа от 1 до 11, так как условие, когда $stopAtTen > 10$ и цикл должен будет прекратиться, проверится после выполнения программы.

Сейчас мы найдем значение 200-го треугольного числа посредством оператора `for`. В результате мы должны получить значение 20100. Если вы получили другое значение, или не получили совсем, значит попробуйте записать код еще раз и будьте внимательнее. Так же не следует копипастить код, так как некоторые символы могут быть записаны на русской раскладке, и найти ошибку в таком случае будет крайне тяжело.

```
#import <Foundation/Foundation.h>

int main (int argc, char *argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc]
init];

    int n, triangularNumber;
    triangularNumber = 0;

    for (n = 1; n <= 200; n=n+1)
        triangularNumber += n;

    NSLog (@\"The 200th triangular number is %i\",
triangularNumber);
    [pool drain];
    return 0;
}
```

Программа требует некоторых пояснений. Для расчета 200-го треугольного числа используется тот же метод, что и для 8-го треугольного числа: вычисляется сумма чисел от 1 до 200.

Перед оператором **for** переменной *triangularNumber* присваивается значение 0. Все переменные необходимо инициализировать (присвоить им значение) до того, как они будут использованы в программе. Определенным типам переменных начальные значения присваиваются по умолчанию, но надежнее присвоить переменным нужные значения явно.

Оператор **for** позволяет обойтись без явного написания всех чисел от 1 до 200. В общем виде оператор **for** имеет формат:

for (начальное_выражение; условие_цикла; выражение_цикла) программный оператор

Три выражения, заключенные в круглые скобки: начальное_выражение, условие_цикла и выражение_цикла — задают среду выполнения программного цикла. Следующий за ними программный оператор (который заканчивается символом «точка с запятой») может быть любым оператором **Objective-C**. Количество выполнений этого оператора определяется параметрами, заданными в операторе **for**.

Первый компонент оператора **for**, начальное_выражение, задает начальные значения до выполнения цикла. В программе эта часть оператора **for** задает начальное значение *n*, равное 1. Второй компонент оператора `for` указывает условие, необходимые для продолжения цикла. Цикл повторяется, пока это условие истинно. В программе, условие_цикла указывается следующим выражением отношения:

n <= 200

Это выражение означает «*n* меньше или равно 200». Оператор «меньше или равно» (знак «меньше» [`<`], после которого сразу следует знак «равно» [`=`]) — это оператор отношения. Операторы отношения используются для проверки условий. Результатом проверки является ответ «да» (или *TRUE*), если условие выполняется, или «нет» (или *FALSE*), если условие не выполняется. Операторы отношения имеют меньший приоритет, чем все арифметические операторы.

6.3.2 Вложенные циклы **for**

Теперь предположим, что нам надо вычислить пять треугольных чисел. Пользователь мог запускать эту программу пять раз, вычисляя все числа по очереди. С

точки зрения изучения Objective-C, нужно, чтобы программа сама обрабатывала эту ситуацию. Для этого достаточно вставить в программу цикл повторения всей последовательности вычислений. Вы можете использовать оператор `for`, чтобы задать такой цикл. Рассмотрим на примере:

```
#import <Foundation/Foundation.h>

int main (int argc, char *argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int n, number, triangularNumber, counter;

    for(counter = 1; counter <= 5; ++counter)
    {
        NSLog(@"Какое число вы хотите проверить?");
        scanf("%i", &number);

        triangularNumber = 0;

        for (n = 1; n <= number; ++n)
            triangularNumber += n;

        NSLog(@"Значение %i треугольного числа - %i", number,
            triangularNumber);
    }
    [pool drain];
    return 0;
}
```

Эта программа содержит два уровня операторов **for**. Внешний оператор **for**:
for(counter = 1; counter <= 5; ++counter)

Он указывает, что цикл программы должен быть выполнен пять раз. Значение счетчика (*counter*) задается равным 1 и затем наращивается с шагом 1 и условием, что он меньше или равен 5 (т.е. пока он не достигнет 6). В отличие от предыдущей программы, переменная *counter* в других местах программы не используется. Она применяется исключительно как счетчик цикла в операторе **for**. Тем не менее, поскольку это переменная, вы должны объявить ее в программе. В цикл этой программы включены все остальные программные операторы, заключенные в фигурные скобки. Эту программу можно описать следующим образом:

```
Пять раз
{
    Получение числа от пользователя.
    Вычисление запрашиваемого треугольного
числа.
    Вывод результатов.
}
```

Часть «Вычисление запрашиваемого треугольного числа» фактически состоит из задания значения 0 для переменной *triangularNumber* и цикла **for**, в котором вычисляется треугольное число. Таким образом, оператор **for** оказывается внутри другого оператора **for**. Вложение циклов может продолжаться до любого уровня. Использование отступов становится крайне важным при работе с усложненными программными конструкциями, такими как вложенные операторы **for**. Так вы сможете быстро определить, какие операторы содержатся внутри каждого оператора **for**.

6.3.3 Варианты цикла **for**

Прежде чем закончить рассмотрение цикла **for**, рассмотрим синтаксические вариации, допустимые при формировании цикла. При написании цикла **for** может оказаться, что вам нужно инициализировать более одной переменной, прежде чем начнется цикл, или нужно проверять несколько условий на каждом шаге цикла. Вы

можете включать несколько выражений в любые поля цикла `for` разделяя выражения запятыми. Например, в операторе, который начинается с

```
for ( i = 0, j = 0; i < 10; ++i )
```

значения `i` и `j` до начала цикла задаются равными 0. Эти два выражения, `i=0` и `j=0`, разделены запятой, и оба выражения считаются частью поля » *начальное_выражение*» этого цикла. Рассмотрим еще один пример. Он начинается с оператора

```
for ( i = 0, j = 100; i < 10; ++i, j -= 10)
```

Здесь задаются две индексные переменные, `i` и `j`, которым присваиваются до начала цикла начальные значения: 0 и 100. После каждого выполнения тела цикла значение `i` увеличивается на 1, а значение `j` уменьшается на 10. Вы можете включать несколько выражений в определенное поле оператора **for** или пропускать несколько полей в этом операторе. Пропустив соответствующее поле, на его месте нужно оставить точку с запятой. Наиболее распространенный случай пропуска поля в операторе `for` возникает, когда не нужно инициализировать начальное выражение. Вы можете просто оставить поле » *начальное_вырожение*» пустым, как в следующем случае, оставив только символ «точка с запятой».

```
for ( ; j != 100; ++j )
```

Этот оператор можно использовать, например, если для переменной `j` до входа в цикл уже задано некоторое начальное значение. Цикл `for`, в котором опущено поле » *условие_цикла*», на самом деле является бесконечным циклом. Такой цикл можно использовать, если для выхода из цикла используются другие средства (например, операторы **return**, **break** или **goto**, которые описываются далее).

Вы можете определять переменные в составе начального выражения цикла `for` с помощью рассмотренных ранее типичных способов определения переменных. Например, следующий оператор задает цикл `for` с целой переменной `counter`, которая одновременно определяется и инициализируется со значением 1.

```
for ( int counter = 1; counter <= 5; ++counter )
```

Переменная `counter` действует только при выполнении этого цикла (она называется локальной переменной цикла) и недоступна извне данного цикла.

6.3.4 Цикл while

Конструкция **while** отдаленно напоминает конструкцию **if**, которая рассматривалась выше: она тоже состоит из выражения и блока кода, заключенного в фигурные скобки. в конструкции **if** в случае истинности выражения блок кода выполняется только один раз. В конструкции **while** блок выполняется снова и снова, пока выражение не станет равно *false*.

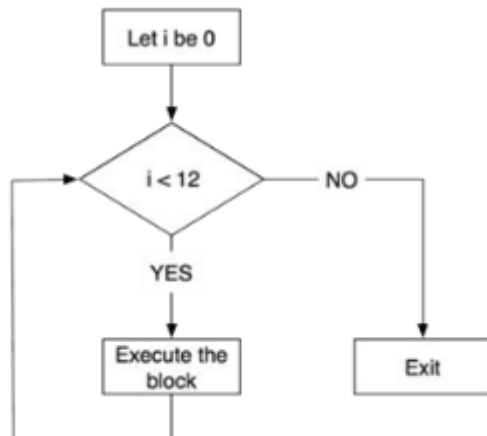


Рис. 6.1. Цикл while

Перепишите функцию `main()` в следующем виде:

```
#include <stdio.h>
```

```
int main(int argc, const char * argv[])
{
    int i = 0; while (i < 12) {
        printf("%d. Aaron is Cool\n", i);
        i++; }
    return 0;
}
```

Постройте и запустите программу

Условие ($i < 12$) проверяется перед каждым выполнением блока. Как только оно окажется ложным, управление передается следующей команде после блока.

Обратите внимание на увеличение i во второй строке блока. Это очень важный момент: без увеличения переменной цикл будет выполняться бесконечно, потому что выражение всегда будет оставаться истинным. На рис. 6.1 изображена блок-схема этого цикла **while**.

Аналогичен циклу **do...while**, только проверка условия происходит перед началом цикла:

```
while (условие) {
    //инструкции;
}
```

6.4 Операторы Break и Continue

6.4.1 Оператор Break

Иногда при выполнении цикла требуется выйти из цикла, как только возникает определенное условие, например, если определено состояние ошибки или преждевременно достигнут конец данных. Для этой цели можно использовать оператор **break**. При выполнении оператора **break** программа сразу выходит из цикла любого типа (**for**, **while** или **do**). Все остальные операторы цикла пропускаются, и выполнение цикла прекращается. Выполнение продолжается с первого оператора, следующего после цикла. Если оператор **break** выполняется из набора вложенных циклов, то прекращается выполнение только вложенного цикла, в котором выполнен этот оператор **break**.

```
#import <Foundation/Foundation.h>

int main (int argc, const char * argv[])
{
    int b = 10, c = 0;
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc]
init];

    while (c <= b)
    {
        ++c;

        if (c == 5)
        {
            NSLog(@"Выход из цикла, выполнение Break");
            break;
        }

        NSLog(@"c = %i", c);
    }
    NSLog(@"Finish!");

    [pool drain];
    return 0;
}
```

В выше приведенной программе, оператор **break** используется для выхода из цикла при выполнении условия $c=5$. Оператор **break** может быть использован в качестве

обработчика исключительных ситуаций, то есть ситуаций при которых не избежать ошибки.

6.4.2 Оператор Continue

Оператор **continue** не прекращает выполнение цикла. Если выполнен оператор **continue**, то пропускаются все следующие операторы до конца цикла, в противном случае выполнение цикла продолжается обычным образом. Оператор **continue** чаще всего используется для обхода группы операторов внутри цикла в зависимости от некоего условия. Не используйте операторы **break** или **continue**, пока не освоитесь с написанием программных циклов и выхода из них. Эти операторы нужно применять с осторожностью, поскольку они затрудняют чтение и отслеживание работы программ.

```
#import <Foundation/Foundation.h>

int main (int argc, const char * argv[])
{
    int a = 1, b = 10, c = 0;
    NSAutoreleasePool * pool = [[NSAutoreleasePool
alloc] init];

    while (c < b) {
        ++c;

        if (c == 5)
        {
            NSLog(@"Возврат к началу, выполнение
Continue");
            continue;
        }

        NSLog(@"c = %i", c);
    }
    NSLog(@"Finish!");
    [pool drain];

    return 0;
}
```

В выше приведенной программе, оператор **continue** используется тогда, когда переменная $c = 5$, при выполнении этого условия, выводится запись о том что оператор **continue** выполняется, и программа переходит в начало цикла **while**. По достижению условия $c > b$, выводится сообщение «*Finish*», и программа завершает свою работу.

7 Основы ООП

7.1 Жизненный цикл объекта

7.1.1 Основные понятия

Во многих языках программирования поддерживается концепция *объектов*. Объекты, как и структуры, содержат некоторые данные. Однако в отличие от структуры, объект также содержит набор функций для работы с этими данными. Чтобы вызвать такую функцию, следует отправить объекту *сообщение*. В объектной терминологии функция, вызываемая при помощи сообщения, называется *методом*.

В начале 1980-х годов Брэд Кокс и Том Лав решили включить объектно-ориентированные возможности в язык C. Они использовали идею выделения памяти в куче и дополнили ее синтаксисом отправки сообщений. Так появился язык Objective-C.

Объекты по своей природе очень «общительны». В ходе своей работы они постоянно принимают и отправляют сообщения, касающиеся всего, что они делают. Сложная программа на языке **Objective-C** может одновременно хранить в памяти сотни объектов, которые работают и отправляют сообщения друг другу.

Класс описывает конкретную разновидность объектов. В описание включаются методы и *переменные экземпляров*, в которых объекты некоторого типа хранят свои данные. Программист обращается к классу с запросом о создании объекта его типа в куче. Полученный объект называется *экземпляром* этого класса.

Например, на **iPhone** установлено множество классов, одним из которых является класс *CLLocation*. Вы можете обратиться к классу *CLLocation* с запросом на создание экземпляра *CLLocation*. Объект *CLLocation* содержит несколько переменных для хранения географических данных: широты, долготы и высоты. Объект также содержит набор методов. Например, можно «спросить» у одного экземпляра *CLLocation*, на каком расстоянии он находится от другого объекта *CLLocation*.

7.1.2 Создание и использование объектов

Для написания первой программы на **Objective-C** создайте новый проект командной строки (*Command Line Tool*), но вместо типа C выберите тип *Foundation* (рисунок 7.1). Присвойте проекту имя *TimeAfterTime*.

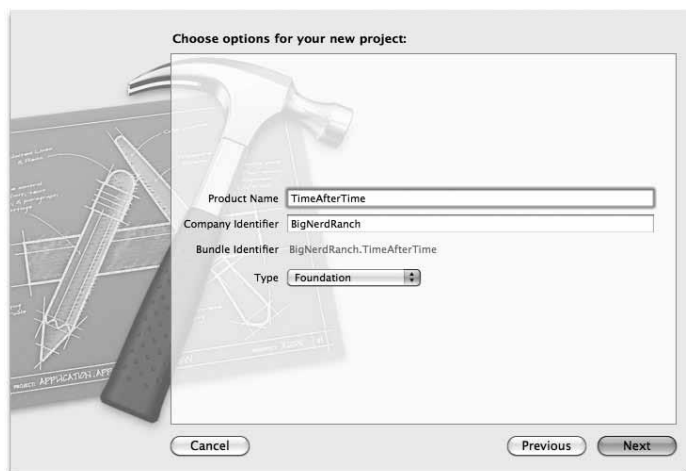


Рисунок 7.1 Создание программы командной строки Foundation

Файлы с кодом **Objective-C** обычно имеют суффикс *.m*. Найдите и откройте файл *main.m*, введите следующие две строки кода:

```
#import <Foundation/Foundation.h>
```

```

int main (int argc, const char * argv[])
{
    @autoreleasepool {

        NSDate *now = [NSDate date];
        NSLog(@"The new date lives at %p", now);

    }
    return 0;
}

```

Мы отправили сообщение *date* классу *NSDate*. Метод *date* приказывает классу *NSDate* создать экземпляр *NSDate*, инициализировать его текущей датой/временем и вернуть адрес, с которого размещается в памяти новый объект. Полученный адрес сохраняется в переменной *now*. Соответственно переменная является указателем на объект *NSDate*.

NSLog() — функция **Objective-C**, отчасти напоминающая *printf()*; она тоже получает форматную строку, заменяет заполнители % фактическими значениями и выводит результат на консоль. Однако ее форматная строка всегда начинается с символа @ и ее не нужно завершать символом новой строки \n.

После запуска программы результат должен выглядеть примерно так:

```

2011-08-05 11:53:54.366 TimeAfterTime[4862:707] The new date lives at
0x100114dc0

```

В отличие от *printf()* функция *NSLog()* выводит перед своими данными дату, время, имя программы и идентификатор процесса.

В дальнейшем, приводя результаты вызова *NSLog()*, мы будем опускать эти служебные данные — строки получаются слишком длинными.

В *NSLog()* заполнитель %p выводит адрес объекта. Чтобы результат больше напоминал дату, используйте заполнитель %@, который приказывает объекту вывести свое описание в виде строки:

```

#import <Foundation/Foundation.h>

int main (int argc, const char * argv[])
{
    @autoreleasepool {
        NSDate *now = [NSDate date];
        NSLog(@"The date is %@", now);

    }
    return 0;
}

```

Новый результат будет выглядеть примерно так:

```

The date is 2011-08-05 16:09:14 +0000

```

7.1.3 Анатомия сообщений

Отправка сообщения всегда заключается в квадратные скобки и всегда состоит из двух частей как минимум:

- указатель на объект, получающий сообщение;
- имя вызываемого метода.

Отправка сообщения (как и вызов функции) может иметь аргументы. Рассмотрим пример.

Объекты *NSDate* представляют конкретную дату и время. Экземпляр *NSDate* может сообщить разность (в секундах) между датой/временем, которые представляет

он сам, и временем 0:00 (по Гринвичу) 1 января 1970 года. Чтобы получить эту информацию, следует отправить сообщение *timeIntervalSince1970* объекту *NSDate*, на который ссылается указатель *now*.

```
#import <Foundation/Foundation.h>

int main (int argc, const char * argv[])
{
    @autoreleasepool {

        NSDate *now = [NSDate date]; NSLog(@"The date is %@", now);
        double seconds = [now timeIntervalSince1970];
        NSLog(@"It has been %f seconds since the start of 1970.", seconds);

    }
    return 0;
}
```

Допустим, вам понадобился новый объект *date* — соответствующий моменту времени на 100 000 секунд позднее уже существующего. Отправив это сообщение исходному объекту *date*, вы получите новый объект *date*. Метод получает аргумент: величину смещения в секундах. Воспользуемся им для создания нового объекта *date* в функции *main()*:

```
#import <Foundation/Foundation.h>

int main (int argc, const char * argv[])
{
    @autoreleasepool {

        NSDate *now = [NSDate date]; NSLog(@"The date is %@", now);

        double seconds = [now timeIntervalSince1970];
        NSLog(@"It has been %f seconds since the start of 1970.", seconds);

        NSDate *later = [now dateByAddingTimeInterval:100000]; NSLog(@"In
100,000 seconds it will be %@", later);

    }
    return 0;
}
```

В отправляемом сообщении *[now dateByAddingTimeInterval:100000]*:
now — указатель на объект, получающий сообщение («получатель»);
dateByAddingTimeInterval — имя метода («селектор»);
100 000 — единственный аргумент.

7.1.4 Объекты в памяти

На рисунке 7.2 изображена диаграмма объектов. На ней изображены два экземпляра *NSDate* в куче. Две переменные *now* и *later* являются частью кадра функции *main()*. Они указывают на объекты *NSDate* (связи изображены стрелками).

Пока что мы видели только один класс: *NSDate*, но в *iOS* и *Mac OS X* входят сотни классов. С самыми распространенными из них мы познакомимся в следующих главах.

id

При объявлении указателя на объект обычно указывается класс объекта, на который будет ссылаться этот указатель:

```
NSDate *expiration;
```

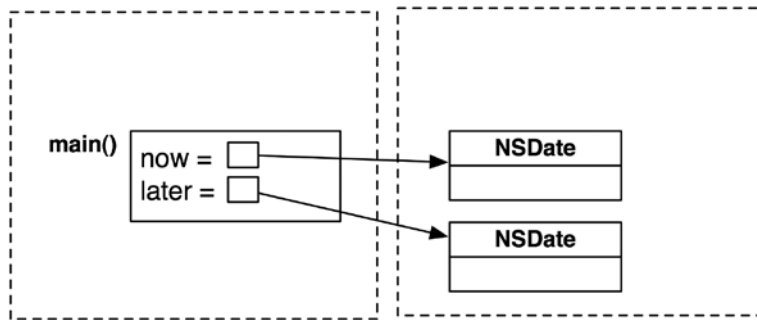


Рисунок 7.2 Диаграмма объектов приложения TimeAfterTime

Но время от времени бывает нужно создать указатель, не зная точно, на какой объект он будет ссылаться. В таких случаях используется тип *id*, который означает «указатель на какой-либо объект Objective-C». Пример использования *id*:

```
id delegate;
```

Обратите внимание на отсутствие звездочки (*) в этом объявлении — она подразумевается при наличии *id*.

На первых порах довольно трудно усвоить концепции классов, объектов, сообщений и методов. Мы будем использовать эти конструкции снова и снова, и с каждым разом они будут выглядеть все более осмысленно.

Используйте два экземпляра *NSDate* для вычисления продолжительности вашей жизни в секундах. Подсказка: новый объект *date* по заданному году, месяцу и т. д. создается следующим образом:

```
NSDateComponents *comps = [[NSDateComponents alloc] init]; [comps
setYear:1969];
[comps setMonth:4]; [comps setDay:30]; [comps setHour:13]; [comps
setMinute:10]; [comps setSecond:0];
NSCalendar *g = [[NSCalendar alloc]
initWithCalendarIdentifier:NSGregorianCalendar]; NSDate *dateOfBirth =
[g dateFromComponents:comps];
```

Для получения количества секунд между двумя экземплярами *NSDate* следует использовать метод *timeIntervalSinceDate*:

```
double d = [laterDate timeIntervalSinceDate:earlierDate];
```

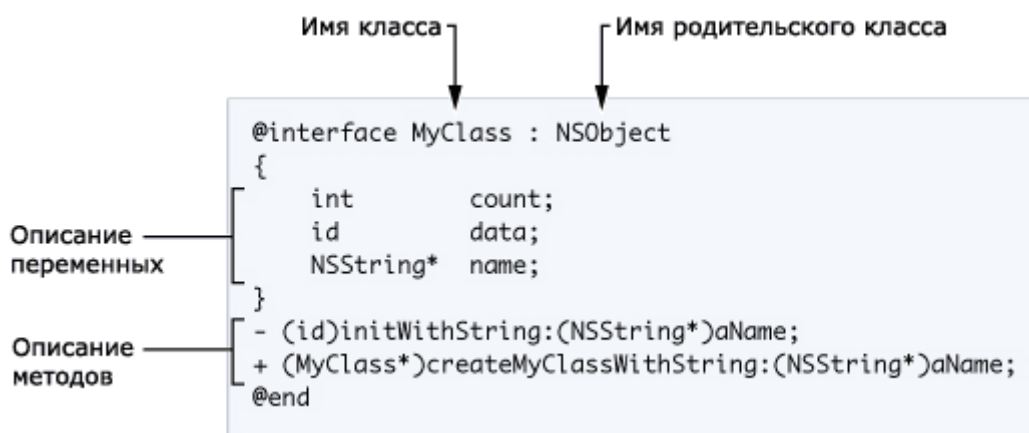
7.2 Создание классов

Как и в большинстве других объектно-ориентированных языков, классы в **Objective-C** обеспечивают основную конструкцию для инкапсуляции некоторых данных с действиями, которые работают с этими данными. Объект является средой выполнения экземпляра класса, и содержит в своей памяти копию переменных экземпляра, декларируемых им в классе, и указателей на методы класса.

Определение класса в **Objective-C** требует двух различных частей: интерфейса и реализации. Интерфейсная часть содержит объявление класса и определяет переменные экземпляра и методы, связанные с классом. Интерфейс, как правило, заносится в *.h* файлы. Реализация содержит фактический код для методов класса и, как правило, содержится в *.m* файлы.

Иллюстрация 1 показывает синтаксис, объявляющий класс по имени *MyClass*, который наследуется из базового класса *NSObject*. Декларация класса всегда начинается с директивы компилятора **@interface** и заканчивается директивой **@end**. После имени класса следует имя родительского класса, отделенное двоеточием. Переменные экземпляра класса объявляются в блоке кода, который

заклучен в скобки ({ и }). После блока описания переменных экземпляра следует список методов, объявляемых в классе. После каждого объявления переменной экземпляра или метода ставится точка с запятой.



Objective-C

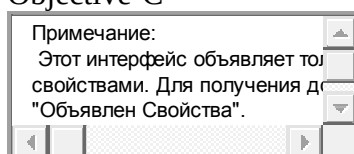


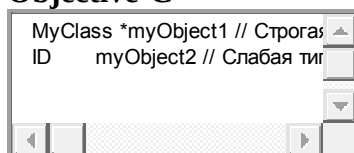
Рисунок 7.3 Объявление класса

Примечание: Этот интерфейс объявляет только методы; классы могут также заявить свойствами.

Objective-C поддерживает как строгую, так и слабую типизацию для переменных, содержащих объекты. Строго типизированные переменные включают имя класса в объявлении типа переменной. Слабо типизированные переменные используют тип **ID** для объекта. Слабо типизированные переменные часто используются для таких вещей, как набор классов, где точный тип каждого объекта в наборе может быть неизвестным. Возможно вы привыкли использовать строго типизированные переменные, и можете думать, что использование слабо типизированных переменных повлечет за собой проблемы, но они на самом деле обеспечивают исключительную гибкость и приносят гораздо больше динамичности в приложения на **Objective-C**.

В следующем примере показаны строго и слабо типизированные переменные:

Objective-C



```

MyClass *myObject1 // Строгая типизация
ID      myObject2 // Слабая типизация
  
```

Обратите внимание на символ звездочки * в первой строке. В **Objective-C**, ссылки на объекты являются указателями. Если это вам ни о чем не говорит, не волнуйтесь, вы не должны быть экспертом и знать все об указателях, для того, чтобы начать программировать в **Objective-C**. Вы просто должны не забывать ставить звездочку * перед именем переменной для строго типизированной декларации объекта и указывать тип **ID** для слабой типизации.

Для объявления класса создается в заголовочном файле с расширением **.h** и использует синтаксис:

```
@interface имя_класса : super_Class
{
    объявление переменных; (имеют зону видимости private)
}
    объявление методов;
@end
```

Реализация класса создается в файле с расширением **.m** и использует синтаксис:

```
#import "имя_класса.h"
@implementation имя_класса

    реализация методов;

@end
```

Пример:

Файл *first.h*

```
#import <Cocoa/Cocoa.h>
@interface first : NSObject
{
    int a;
    int b;
}
- (id)init;
- (void)setA : (int)A andB : (int)B;
- (long)summa;
- (long)mul;
@end
```

Здесь мы видим объявление класса с именем **first** порожденный от класса **NSObject** (В Apple runtime библиотеке все классы являются наследниками класса **NSObject**, если мы будем писать программы с использованием Cocoa, то будет хорошей практикой наследовать свои классы также от **NSObject**, предоставляющего множество полезных возможностей).

Директива **#import** подключает к нашему объявлению заголовок **Cocoa.h**, содержащий объявления всех классов и протоколов пространства **Cocoa.framework**.

Далее следует объявление переменных типа **Integer** **a** и **b**, следом идут методы, начинающиеся с символа **-** (иногда с **+**), далее в скобках указывается возвращаемый тип данных, следом название метода, далее аргументы через пробел, перед скобками с типом аргумента ставится двоеточие, перед аргументом может стоять метка (в методе **setA** - меткой аргумента **B** является слово **andB** - наследие Smalltalk), используется для большей читабельности кода.

Файл *first.m*

```
#import "first.h"
@implementation first
- (id)init {
    self = [super init];
    if(self){
        //инициализация вашего класса
    }
    return self;
}
- (void)setA : (int)A andB : (int)B{
    a = A;
    b = B;
}
```

```

- (long)summa{
    return a+b;
}
- (long)mul{
    return a*b;
}
@end

```

Начало файла реализации класса стандартно: с директивы **#import "first.h"**, загружающей объявление класса.

Директива **@implementation first** говорит компилятору, что далее следует реализация класса **first**

Далее следует функция **init**, которая служит для инициализации нашего класса. При создании экземпляра класса после выделения памяти всегда необходимо инициализировать класс, метод **init** - это перегруженный метод наследуемого класса (супер класса) *NSObject*, поэтому в самом начале мы вызываем метод **init** наследуемого класса, для определения которого зарезервировано слово **super**. Значение, возвращаемое инициализатором, является указателем на наш класс, если инициализация неудачна, то возвращаемое значение равно *nil*.

self - зарезервированный указатель, который указывает на наш класс внутри самого класса

Инициализатор можно и не перегружать, просто опустив его объявление и реализацию. Также класс может содержать несколько инициализаторов, также содержащих и аргументы.

Далее следует реализация методов, практически ничем не отличающаяся от обычного **C**

В вышеуказанном примере переменные *a* и *b* имеют зону видимости только внутри своего класса. Мы написали метод *setA:(int) andB:(int)* для установки их значений. Если же нам необходимо прочитать значение *a* или *b*, то необходимо написать методы возвращающие их.

Например:

```

- (int)valueA{
    return a;
}

```

Начиная с **Objective C 2.0** появилось понятие - свойство, под свойство выделена директива **@property**, которая указывает, что данная переменная класса имеет метод для чтения и записи в нее. Данная директива объявляется в файле объявлении класса.

Сопутствующая директиве **@property**, директива говорящая компилятору, что необходимо создать методы для доступа к переменной носит наименование **@synthesize** и объявляется в файле реализации класса.

Синтаксис объявления свойства:>

@property (атрибуты, через запятую) тип имя;

Пример:

 *Файл myclass.h*

```

@interface myclass
{
    int a;
    int b;
}
@property int a;
@property int b;

```

```
@end;
```

Как видим здесь объявляется класс *myclass* с двумя переменными *a* и *b*, доступными извне с помощью объявления свойств.

 *Файл myclass.m*

```
#import "myclass.h"
@implementation myclass

@synthesize a;
@synthesize b;

@end
```

Вот так все стало проще. Теперь мы имеем для переменных методы, позволяющие считывать и записывать их значения.

Не надо забывать, что определение свойств сокращает лишь исходный код в редакторе, а компилятор создает методы, которые предоставляют механизм доступа к переменным класса, тем самым как раз увеличивая размер конечного приложения. Поэтому, если нет необходимости видимости переменной, то не стоит к ней объявлять и свойство.

Внимание. Начиная с **Xcode 4.5** использование **@synthesize** в файле реализации стало необязательным. В этом случае в файле - заголовке объявляется свойство **@property NSInteger a**, доступ к самой переменной будет начинаться с символа подчеркивания *_a*, а доступ к сеттеру или геттеру внутри класса *[self a]*.

7.3 Определение методов и свойств

Класс в Objective-C может объявлять два типа методов: методы экземпляра и методы класса. Метод экземпляра класса – метод, выполнение которого находится в границах юрисдикции определенного экземпляра класса. Другими словами, прежде, чем вы вызываете метод экземпляра класса, вы должны сначала создать этот экземпляр класса. Методы класса, к примеру, не требуют, чтобы вы создавали экземпляр класса, но поговорим об этом позже.

Объявление метода состоит из идентификатора типа метода, типа возвращаемого значения, одного или нескольких ключевых слов сигнатур, а также информацию о типах и именах параметров.

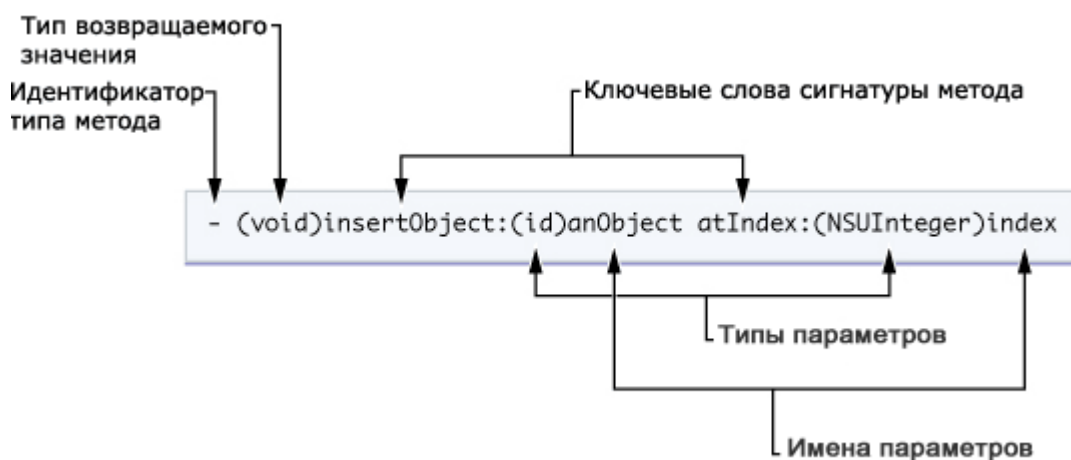


Рисунок 7.4 Синтаксис объявления метода

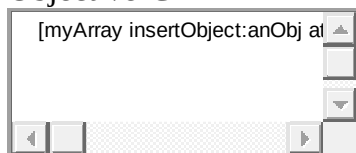
Объявлению предшествуют знак минуса (-), который указывает, что это метод

экземпляра класса. Фактическое имя метода (insertObject:atIndex:) – это связь всех ключевых слов сигнатуры, включая символы двоеточия. Символы двоеточия объявляют присутствие параметра. Если у метода нет никаких параметров, то опустите двоеточие только после первого ключевого слова сигнатуры. В приведенном примере метод получает два параметра.

Когда вы хотите вызвать метод, вы делаете это через “обмен сообщениями” с соответствующим объектом. Сообщение в этом случае – это сигнатура метода вместе с информацией параметра, необходимые методу. Все сообщения, которые вы посылаете объекту, отправляются динамически, таким образом обеспечивая полиморфизм в поведении классов Objective-C. Другими словами, если подкласс определяет метод с той же самой сигнатурой, как один из ее родительских классов, сначала сообщение получает подкласс, решая отправлять сообщение своему родительскому классу или нет.

Сообщения заключаются в квадратные скобки [и]. Внутри скобок указывается объект, которому предназначено сообщение (включая все необходимые параметры) и само сообщение. Например, чтобы послать сообщение insertObject:atIndex: объекту из myArray переменной, следует использовать следующий синтаксис:

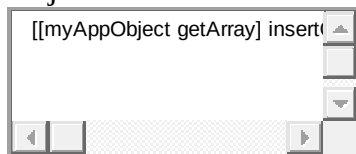
Objective-C



```
[myArray insertObject:anObj atIndex:0];
```

Чтобы избежать объявления множества локальных переменных хранящих временные значения, Objective-C позволяет вкладывать сообщения друг в друга. Возвращаемое значение из каждого вложенного сообщения используется как параметр, или как цель для другого сообщения. Например, вы можете заменить любую из переменных, используемых в предыдущем примере, сообщениями чтобы получить значения. Таким образом, если бы у вас был другой объект, названный myAppObject, у которого были бы методы обращения к объекту типа массив и объект, который необходимо вставить в массив, то вы могли бы написать предыдущий пример, таким образом:

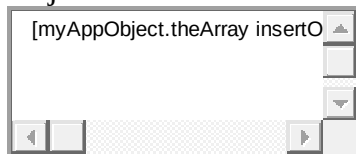
Objective-C



```
[[myAppObject getArray] insertObject:[myAppObject getObjectToInsert] atIndex:0];
```

Objective-C также обеспечивает точечный синтаксис вызова. Методы доступа получают и устанавливают состояние объекта, и обычно принимают форму - (type)propertyName и -(void)setPropertyName:(type). Используя точечный синтаксис, вы можете переписать предыдущий пример следующим образом:

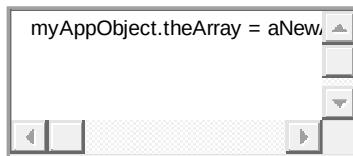
Objective-C



```
[myAppObject.theArray insertObject:[myAppObject objectToInsert] atIndex:0];
```

Вы также можете использовать точечный синтаксис для назначения:

Objective-C



```
1 myAppObject.theArray = aNewArray;
```

Это просто другой синтаксис для записи, [myAppObject setTheArray:aNewArray];.

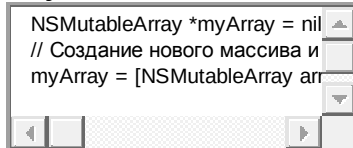
Хотя предыдущие примеры показывают отправку сообщений экземпляру класса, но вы также можете отправлять сообщения самому классу. Когда посылается сообщение классу, указанный метод должен быть определен как метод класса, а не метод экземпляра.

Обычно вы будете использовать методы класса, как фабричные методы для создания новых экземпляров класса или для доступа к некоторой части общей информации, связанной с классом. Синтаксис для объявления метода класса идентичен синтаксису объявления метода экземпляра, за одним исключением — вместо знака минус перед идентификатором типа метода, вы используете знак плюс (+).

The following example illustrates how you use a class method as a factory method for a class. In this case, the array method is a class method on the NSArray class—and inherited by NSMutableArray—that allocates and initializes a new instance of the class and returns it to your code.

Следующий пример показывает, как использовать метод класса как фабричный метод для класса. В этом случае метод массива является методом класса NSArray и наследуется классом NSMutableArray, который выделяет и инициализирует новый экземпляр класса и возвращает его в ваш код.

Objective-C

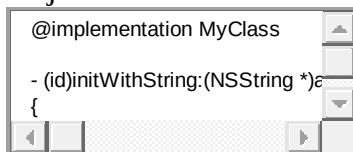


```
NSMutableArray *myArray = nil; // nil - это тоже самое что и NULL
// Создание нового массива и присвоение его переменной myArray.
myArray = [NSMutableArray arrayWithObjects:];
```

Листинг 1 показывает реализацию класса MyClass из предыдущего примера. Как и объявление класса, реализация класса заключена между двумя директивами компилятора — @implementation и @end. Эти директивы предоставляют обзорную информацию, которая нужна компилятору для того, чтобы связать вложенные методы с соответствующим классом. Поэтому, объявление метода соответствует заданному объявлению в интерфейсе, кроме включения блока с кодом.

Листинг 1. Реализация класса

Objective-C



```
1 @implementation MyClass
2
3 - (id)initWithString:(NSString *)aName
4 {
5     self = [super init];
```

```

6     if (self) {
7         name = [aName copy];
8     }
9     return self;
10 }
11
12 + (MyClass *)createMyClassWithString: (NSString *)aName
13 {
14     return [[[self alloc] initWithString:aName] autorelease];
15 }
16 @end

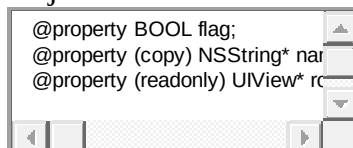
```

Свойства – удобное средство для замены объявления средств доступа метода (*accessor method*). Свойства не создают новые переменные экземпляра в объявленном классе. На самом деле, свойства – это упрощенный способ (*shorthand*) для определения методов, обращающихся к существующим переменным экземплярам. Классы, которые подставляют переменные экземпляры, могут использовать обозначение свойства вместо того, чтобы использовать синтаксис *getter* и *setter*. Классы также могут использовать свойства для того, чтобы подставить “виртуальные” переменные экземпляров класса – данные вычисленные динамически, но фактически не сохраненные в переменных экземплярах класса.

Проще говоря, свойства позволяют сократить количество избыточного кода, который вы будете писать. Поскольку большинство методов доступа осуществляется именно подобным способом, свойства избавляют нас от необходимости явно устанавливать методы *getter* и *setter* для каждой переменной экземпляра, указанной в классе. Вместо этого вы описываете поведение, которое вы хотите использовать для объявления свойств и затем синтезируете фактические *getter* и *setter* методы, основанные на этом объявлении, прямо во время компиляции.

Включаете объявления свойств с объявлениями метода в вашем интерфейсе-классе. Для объявления свойств используют директиву **@property**, сопровождаемую информацией о типе и имени свойства. Вы можете также описывать свойства с собственными параметрами, которые определяют, как методы доступа должны себя вести. Ниже приведены примеры объявления нескольких простых свойств:

Objective-C



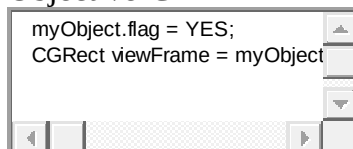
```
@property BOOL flag;
```

```
@property (copy) NSString* nameObject; // Копирование объекта во время присваивания.
```

```
@property (readonly) UIView* rootView; // Создание только метода getter.
```

Еще один удобный нюанс в применении свойств – это то, что вы можете использовать синтаксис с точкой, обращаясь к свойствам вашего кода. А вот пример его использования:

Objective-C



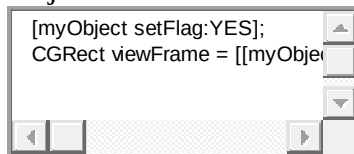
```
myObject.flag = YES;
```

```
CGRect viewFrame = myObject.rootView.frame;
```

Хотя имена объекта и свойства в предыдущем примере надуманы, они все же

демонстрируют гибкость свойств. Синтаксис с применением точки фактически маскирует соответствующий набор вызовов метода. Каждое доступное для чтения свойство поддерживается методом с таким же именем, что и само свойство. Каждое доступное для записи свойство поддерживается дополнительным методом формы `setPropertyNamed:`, где имя свойства начинается с большой буквы. (Эти методы являются фактической реализацией свойств и являются причиной, по которой вы можете включать объявления свойства для атрибутов вашего класса не поддерживаемых переменными экземпляра класса). Давайте перепишем предыдущий пример с использованием методов вместо свойств:

Objective-C



```
[myObject setFlag:YES];
CGRect viewFrame = [[myObject rootView] frame];
```

В случае необходимости можно переопределить методы установки или чтения переменной вместе называемые аксессорами (Accessor) и соответственно чтения - геттер (getter), записи - сеттер (setter):

```
getter=метод геттера
setter=метод сеттера
```

Бывает не всегда нужно предоставлять полный доступ к переменной класса, в этом случае нужно ограничить его в объявлении `@property`:

- **readwrite** - и чтение и запись
- **readonly** - только чтение

Свойство только для чтения будет выглядеть так: `@property (readonly) int a;`

strong - обозначает строгое переназначение объекта, делая указатель на объект владельцем этого объекта

weak - в отличии от **strong** обозначает нестрогое соответствие и если объект был освобожден из памяти (из другого класса или потока), то значение установится в *nil*. Не поддерживается OS X v10.6 и iOS 4; используйте `assign`

copy - указывает на то, что для присваивания будет использована копия переданного объекта. Первоначальному объекту посылается сообщение **release**. Может быть использовано только для объектов, поддерживающих протокол **NSCopying**, например `NSString`.

assign - для задания нового значения используется оператор присваивания. Используется только для скалярных типов (`NSInteger` и `CGRect`) либо для объектов, которыми мы не владеем.

retain - указывает на то, что для объекта, используемого в качестве нового значения `instance-переменной`, управление памятью происходит вручную (не забываем потом освободить память).

nonatomic - делает более быстрым способ доступа к объекту в не многозадачной среде, так-как в случае с **atomic** (по умолчанию все аксессоры **atomic**) и использованием `strong`, `copy`, или `retain`, во избежание коллизий с возможностью доступа к свойству из других потоков в сеттере строится код блокирующий переключение между потоками.

7.4 Указатели

Файлы модулей на **Objective-C** имеют расширение **“.m”** (если использовалась смесь **C++** и **Objective-C**, то расширение **“.mm”**). Заголовочные файлы – **“.h”**. Все,

создаваемые в **Objective-C** объекты классов должны размещаться в динамической памяти. Поэтому особое значение приобретает тип **id**, который является указателем на объект любого класса (по сути **void ***). Нулевой указатель именуется константой **nil**. Таким образом, указатель на любой класс можно привести к типу **id**. Возникает проблема: как узнать к какому классу относится объект, скрывающийся под **id**? Это делается благодаря инварианту **isa**, который присутствует в любом объекте класса, унаследовавшего специальный базовый класс **NSObject** (приставка **NS** обозначает **NeXT Step**). Инвариант **isa** относится к зарезервированному типу **Class**. Объект такого типа позволяет узнавать имена своего и базового класса, набор инвариантов класса, а также прототипы всех методов, которые реализовал этот объект и их адреса (посредством локального списка селекторов).

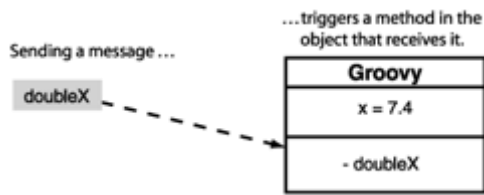
Все зарезервированные слова **Objective-C**, отличающиеся от зарезервированных слов языка **C**, начинаются с символа **@** (например **@protocol**, **@selector**, **@interface**). Обычно имена инвариантов классов с ограниченной областью видимости (**@private**, **@protected**) начинаются с символа подчеркивания. Для строк имеется очень удобный класс **NSString**. Строковая константа такого класса записывается как **@”Hello world”**, а не как обычная для **C** строковая константа **”Hello world”**. Тип **BOOL** (по сути *unsigned char*) может принимать константные значения **YES** и **NO**. Все особые для **Objective-C** зарезервированные слова (которые отличаются от языка **C** и находятся в заголовочном файле **objc/objc.h**) приведены ниже:

- **@interface**. Начинает объявление класса или категории (категория – расширение класса дополнительными методами без наследования).
- **@implementation**. Начинает определение класса или категории.
- **@protocol**. Начинает объявление протокола (аналог класса **C++**, состоящего из чисто виртуальных функций).
- **@end**. Завершает объявление/определение любого класса, категории или протокола.
- **@private**. Ограничивает область видимости инвариантов класса методами класса (аналогично **C++**).
- **@protected**. Стоит по умолчанию. Ограничивает область видимости инвариантов класса методами класса и методами производных классов (аналогично **C++**).
- **@public**. Удаляет ограничения на область видимости (аналогично **C++**).
- **@try**. Определяет блок с возможной генерацией исключений (аналогично **C++**).
- **@throw**. Генерирует объект-исключение (аналогично **C++**).
- **@catch ()**. Обрабатывает исключение, сгенерированное в предшествующем блоке **@try** (аналогично **C++**).
- **@finally**. Определяет блок после блока **@try**, в который передается управление независимо от того, было или нет сгенерировано исключение.
- **@class**. Сокращенная форма объявления класса (только имя (аналогично **C++**)).
- **@selector(method_name)**. Возвращает скомпилированный селектор для имени метода **method_name**.
- **@protocol(protocol_name)**. Возвращает экземпляр класса-протокола с именем **protocol_name**.
- **@encode (type_spec)**. Инициализирует строку символов, которая будет использована для шифрования данных типа **type_spec**.
- **@synchronized()**. Определяет блок кода, выполняющегося только одной нитью в любой определенный момент времени.

7.5 Сообщения и методы

Чтобы заставить объект выполнить какой-нибудь метод нужно послать ему сообщение, именуемое так же, как и требуемый метод. Такое сообщение называется селектор метода. Синтаксис послышки таков:

```
[receiver method];
```



В сообщении можно передавать параметры для вызываемого метода:

```
[receiver method: 20.0 : 30.0];
```

Перед каждым параметром необходимо ставить двоеточие. Сколько двоеточий – столько и параметров. Имя метода может продолжаться после каждого такого двоеточия-параметра:

```
[receiver methodWithFirstArgument: 10 andSecondArgument: 20];
```

Методы с неограниченным количеством аргументов вызываются следующим образом:

```
[receiver undefinedNumberParameters: one, two, three, four, five, six, seven];
```

Посылка сообщения, как и любая функция C, возвращает определенное (может void) значение:

```
BOOL booleanValue;  
booleanValue = [receiver method];
```

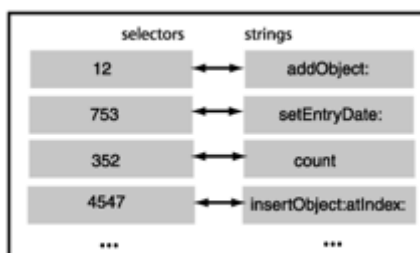
При посылке сообщения **nil** оно просто пропадает. При посылке сообщения объекту, который принадлежит классу, не реализовавшему заказанный метод, возникает исключение, которое, будучи не перехваченным, приводит всю программу к незапланированному завершению. Для проверки, отвечает ли данный объект на какое-либо сообщение можно использовать следующий шаблон кода:

```
if ([anObject respondsToSelector: @selector(myMethodWith2Arguments: :)])  
{  
    //можно вызывать  
    [anObject myMethodWith2Arguments: @"first" : @"second"];  
}  
else  
{  
    //ни в коем случае не вызывать  
}
```

Посылка сообщения транслируется в C-функцию с прототипом:

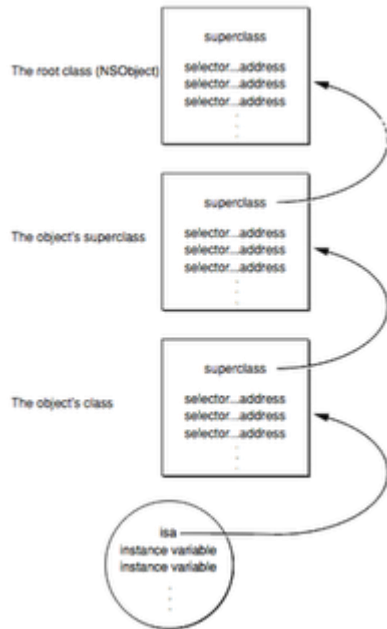
```
id objc_msgSend(id receiver, SEL method, ...);
```

Тип **SEL**, по сути, определен как *char const **, но лучше воспринимать его как *int*, поскольку во время выполнения все селекторы индексируются целыми значениями согласно глобальной таблице селекторов.



Пользуясь инвариантом *isa* объекта *receiver* (при использовании фреймворка **Foundation**, все классы должны наследовать класс **NSObject**, поэтому наличие *isa* неизбежно), эта функция просматривает локальный список селекторов класса с целью

определить, отвечает ли объект данного класса на сообщение *method*. Если такой селектор находится, то управление передается соответствующему методу класса, которому передается *id* объекта (указатель на его инварианты) и указанные после селектора параметры функции *objc_msgSend()*. Значение, возвращенное методом, отдается как результат послышки сообщения. Если у объекта-приемника данный селектор отсутствует, функции *objc_msgSend()* просматривает список селекторов его базового класса.



При такой схеме вызов, например:

```
[receiver addObject: otherObject];
```

Транслируется в:

```
objc_msgSend(receiver, 12, otherObject);
```

Так как в глобальной таблице селекторов 12 соответствует строке "*addObject:*". Далее функция *objc_msgSend()* выполняет поиск по списку селекторов объекта *receiver* и, найдя его (пусть это объект класса **NSArray**, который реализовал метод с селектором 12), производит вызов типа:

```
addObject(receiver, otherObject);
```

Интересно отметить, что прототип метода *addObject* из предыдущего раздела в объявлении класса выглядел так:

```
- (void)addObject: (id)otherObject;
```

То есть принимал всего один параметр. Но, исходя из принципа объектно-ориентированной парадигмы, что методы – это подпрограммы, обрабатывающие определенные наборы данных, методу необходимо передавать адрес данных, подлежащих обработке. Поэтому такой параметр передается во всякий метод класса неявно. Компилятору об этом дополнительном параметре дает понять минус ("-"), стоящий первым в прототипе метода. Такой метод (с минусом впереди) называется методом объекта (или экземпляра), т.к. может быть вызван только для объекта какого-нибудь класса. В теле метода этот указатель на экземпляр данных (или адрес объекта, которому послали сообщение) доступен посредством зарезервированного слова *self* (аналог *this* в C++), а указатель на экземпляр базового класса – через зарезервированное слово *super*. Кроме того, в метод объекта также передается неявный параметр *_cmd* – селектор этого метода из глобальной таблицы селекторов. С точки зрения программиста C++ все методы объектов в **Objective-C** как-будто объявлены с ключевым словом *virtual*, и всегда следуют динамическому полиморфизму.

Если в начале прототипа метода поставить знак плюс (“+”), то такой метод будет считаться методом класса, и, естественно, не будет принимать неявный параметр *self* (это аналогично объявлению *static*-метода в C++). А без инварианта *isa* объекта, на который указывает *self*, указатель *super* работать, конечно, тоже не будет. Таким образом, прототип любого метода объявляется так:

```
-|+ (<тип возвращаемого значения>) основнаяЧастьИмениМетода
[ : (<тип первого параметра>) имяПервогоФормальногоПараметра
[ [дополнительнаяЧастьИмениМетода] : (<тип второго
параметра>) имяВторогоФормальногоПараметра]
... ]
```

Например:

```
+ (Class)class;
+ (id)alloc;
- (id)init;
- (void)addObject: (id)anObject;
+ (NSString *)stringWithCString: (const char*)aCString usingEncoding: (enum
NSStringEncoding)encoding;
- (NSString *)initWithFormat: (NSString *)format, ...;
```

Если метод возвращает некий объект (тип *id*) или класс (тип *Class*), можно воспользоваться вложенным синтаксисом вызова:

```
[myLabel setText: [[NSString stringWithString: @"Hello"]
stringByAppendingString: @" world"]];
```

Здесь объекту класса *UILabel* из фреймворка **UIKit** устанавливается значение инварианта *text* равное строке *@“Hello world”*. Эта строка, в свою очередь, образована конкатенацией строк *@“Hello”* и *@“ world”*. Первая является результатом послышке сообщения *stringWithString* классу **NSString** с параметром-константой *@“Hello”*. Такой вызов возвращает объект класса **NSString**, инициализированный строкой-параметром. Затем этому объекту посылается сообщение *stringByAppendingString* с параметром *@“ world”*. Результат послышки этого сообщения и есть объект класса **NSString**, содержащий конкатенацию значения объекта-приемника и строкового аргумента. Этот объект и попадает как параметр в сообщение *setText*: объекта *myLabel*.

Объявим простой класс комплексного числа в файле **Complex.h**:

```
#import <Foundation/Foundation.h> //для NSObject и строк NSString

@interface Complex : NSObject
{
double _re; //инвариант для действительной части
double _im; //инвариант для мнимой части
NSString *_format; //строка формата для метода description
}
- (id)initWithRe: (double)re andIm: (double)im; //специализированный
конструктор
+ (Complex *)complexWithRe: (double)re andIm: (double)im; //метод класса для
одноэтапного создания объекта
- (Complex *)add: (Complex *)other; //метод для сложения
- (Complex *)sub: (Complex *)other; //метод для вычитания
- (NSString *)format; //метод доступа к _format
- (void)setFormat: (NSString *)format; //метод установки _format
- (double)re; //остальные методы доступа к действительной и мнимой частям
- (void)setRe: (double)re;
- (double)im;
- (void)setIm: (double)im;
@end
```

Как видим, все объявление заключено в ключевые слова **@interface** и **@end**. Первым делом объявляются инварианты (в фигурных скобках). Вне фигурных скобок объявляются методы. Метод *description* отсутствует в объявлении класса не случайно. Дело в том, что он, как и метод *dealloc* и *init*, присутствует в определении класса. При послышке объекту класса **Complex** сообщения *description* будет рассмотрен его локальный список селекторов, куда, после компиляции, попадут селекторы всех методов,

реализованных классом этого объекта, даже не объявленные в интерфейсной части. То есть *init*, *description* и *dealloc* будут вызывать абсолютно корректно.

7.6 Создание объекта

В связи с тем, что все объекты распределяются в динамической памяти, создание объекта приходится проводить в два этапа: 1) выделении памяти (сообщение *alloc*) и 2) инициализация инвариантов (конструкторы класса).

```
MyClass *myObject = [[MyClass alloc] init]; //метод класса MyClass alloc
выделяет участок памяти нужного размера и возвращает указатель на него, метод
объекта init инициализирует инварианты объекта myObject
```

После создания объекта им можно смело пользоваться:

```
NSMutableArray *array = [[NSMutableArray alloc] init]; //создаем изменяемый
массив
MyClass *myObject = [[MyClass alloc] init]; //наш объект
[myObject myMethod]; //посылка некоторого сообщения
[array addObject: myObject]; //помещаем объект в массив
MyClass *otherObject = [array getLastObject:]; //достаем его из массива,
указываем на него другим указателем
[otherObject myOtherMethod: YES]; //посылаем ему другое сообщение с
аргументом типа BOOL
```

Некоторые классы обладают методом для быстрого (в один этап) создания собственных экземпляров. Такие методы являются методами класса, возвращают указатель на объект своего класса и их имя обычно начинается с названия самого класса. Например метод:

```
+ (NSString *)stringWithCString: (char const *)string encoding:
(NSStringEncoding)encoding;
```

Возвращает уже готовую строку, инициализированную соответствующей сторокой с завершающим нулем, без вызовов *alloc* и *init*:

```
NSString *myString = [NSString stringWithCString: "Bla-bla-bla" encoding:
NSASCIIStringEncoding];
```

Как только указатель на объект выходит за свою область видимости, память, выделенная под него, безвозвратно теряется (если, конечно, это был последний указатель на тот объект) и происходит утечка. Дабы избежать таких нежелательных последствий в **Objective-C** поддерживается парадигма подсчета ссылок на ресурсы. Таким образом, у каждого объекта есть целочисленный счетчик, который показывает количество ссылающихся на него указателей. По достижению этим счетчиком нуля, память, выделенная для данного объекта, возвращается системе. После вызова метода класса *alloc*, этот счетчик равен единице. Чтобы увеличить его значение необходимо послать объекту сообщение *retain*, а чтобы уменьшить – *release*. Все эти методы реализует **NSObject**, который любой наш класс непременно наследует. Интересно отметить, что значение счетчика для статических объектов класса **NSString** (например @"I am a string") равно -1, то есть максимально возможное. Вот пример работы со счетчиком:

```
id anObject = [SomeClass alloc]; //вначале счетчик == 1
[anObject init]; //тут создаются инварианты объекта
[anObject retain]; //увеличим его значение (теперь он == 2)
[anObject release]; //уменьшим (счетчик опять == 1 и объект по прежнему
жизнеспособен)
[anObject release]; //счетчик обнуляется, уменьшаются на 1 счетчики
инвариантов и выделенная под объект память возвращается ОС
```

Реализация *init* очень важна. Это конструктор класса. Конструкторы отличаются тем, что возвращают *id* и их названия всегда начинается со слова *init*, а конструктор по умолчанию – это и есть просто *init*. Схема любого конструктора примерно следующая:

```
- (id)init
{
self = [super init]; //вызываем конструктор базового класса для
//инициализации его инвариантов
if (self) //если в конструкторе базового класса все прошло удачно
//и он вернул корректный объект, а не освободив память вернул nil
```

```

{
//то тут можно смело инициализировать свои инварианты
}
return self; //и возвращать самого себя
}

```

Вот типичный специализированный (не по умолчанию) конструктор для класса с двумя членами типа некоторого класса и одним целочисленным инвариантом:

```

- (id)initWithInt: (int)number
{
if (self = [super init])
{
_myMember1 = [[SomeClass alloc] init]; //все как положено: выделили память,
затем ее инициализировали
_myMember2 = [[SomeClass alloc] init];
_myIntMember = number; //здесь конструктор ни к чему
//инициализируем переданным параметром
}
return self;
}

```

Реализация *release* и *retain* для **NSObject** идеологически примерно следующая, и ее не нужно переопределять в производных классах, в силу отсутствия доступа к инварианту счетчика ссылок:

```

- (void)retain
{
[_internalLock lock]; //блокировка для синхронизации
_referenceCounter++; // пусть _referenceCounter - скрытый инвариант счетчика
[_internalLock unlock];
}
- (void)release
{
[_internalLock lock];
_referenceCounter--; //уменьшим счетчик
if (!_referenceCounter) //если он равен нулю
{
[_internalLock unlock];
[self dealloc]; //скажем себе, что пора умирать (блокировка освободится тут)
}
[_internalLock unlock];
}

```

То есть самому объекту посылается сообщение *dealloc*, в реализации метода которого он может, по необходимости, уменьшить счетчики своих инвариантов и передать аналогичное сообщение объекту базового класса, чтобы он сделал то же самое. Очевидно, реализация метода *dealloc* для **NSObject** освободит память, выделенную объекту. Обычно *dealloc* для какого-нибудь класса выглядит так:

```

- (void)dealloc
{
[_myMember1 release]; //уменьшим счетчик своего инварианта
[_myMember2 release]; //уменьшим счетчик другого своего инварианта
//[_myIntMember release]; это полный бред, т.к. встроенные типы сообщений не
принимают вообще и счетчиков не ведут
[super dealloc]; //скажем объекту базового класса, что пора освобождать
память
}

```

Правильная работа с подсчетом ссылок очень важна при возврате адреса объекта из метода или инициализации инварианта формальным параметром. Обычно такими вещами занимаются так называемые методы доступа, возвращающие и устанавливающие инварианты объектов. Принято именовать метод, возвращающий значение инварианта, так же как и инвариант, а имя метода, устанавливающего его значение, начинать со слова *set*:

```

- (void)setRe: (double)re

```

```
{
    _re = re;
}
```

Так как инвариант `_re` относится к встроенному типу, никаких сложностей с изменением его значения не возникает. Но если инвариант – объект некоторого класса – то простым присваиванием не обойтись, ведь надо учитывать счетчики ссылок. Для решения этой проблемы применяются следующие три метода:

```
//например, нужно изменить текст у ярлыка
[label setText: @"Hello world"]; //устанавливаем инвариант text
//объекта label равным текстовой константе типа NSString *

//примерная реализация setText в классе UILabel (вариант №1)
- (void)setText: (NSString *)text
{
    [text retain]; //увеличиваем счетчик ссылок на формальный параметр
    [_text release]; //уменьшаем счетчик ссылок текущего значения своего
    инварианта _text
    _text = text; //инициализируем инвариант новым значением
}

//примерная реализация setText в классе UILabel (вариант №2)
- (void)setText: (NSString *)text
{
    if (_text != text) //сравниваем указатели на объекты
    {
        [_text release]; //уменьшаем счетчик ссылок текущего значения
        //своего инварианта _text
        _text = [text retain]; //увеличиваем счетчик ссылок
        //на формальный параметр и инициализируем свой инвариант
    }
}

//примерная реализация setText в классе UILabel (вариант №3 – нежелательный)
- (void)setText: (NSString *)text
{
    if (_text != text)
    {
        [_text autorelease]; //скинем текущее значения своего
        //инварианта _text в самовыгружаемый пул
        _text = [text retain]; //увеличиваем счетчик ссылок
        //на формальный параметр и инициализируем свой инвариант
    }
}
```

Вариант №3 не очень удачный потому, что засоряет текущий самовыгружаемый пул, а обычно это не очень желательно. Метод доступа для чтения значения инварианта всегда очень прост:

```
- (NSString *)text
{
    return _text;
}
```

Теперь попробуем вернуть из метода созданный внутри него объект:

```
-(NSString *)sayHelloToName: (NSString *)name withSurname: (NSString
*)surname
{
    NSString *retString = [[NSString alloc] initWithFormat: @"%@ %!", name,
    surname]; //инициализируем созданный объект посредством строки формата
    return retString;
}
```

Строка формата соответствует стандарту языка C. Но если в ней необходимо указать тип `id`, то используется спецификатор формата `%@`. Каким образом метод, разбирающий формат, понимает какие символы подставить вместо `id`? Он просто

подставит то, что вернет метод описания `description` данного объекта. Этот метод изначально объявлен для класса **NSObject**. **NSString** переопределил его на вывод своего строкового содержания. Переопределив его, любой объект может представлять свое строковое содержание. Например, так это может сделать класс комплексного числа с двумя инвариантами типа *double*:

```
- (NSString *)description
{
    return [NSString stringWithFormat: @"re: %lf im: %lf", _re, _im];
    //возвращает строку @"re: 1.0 im: 2.5" для _re == 1.0 и _im == 2.5
}
```

После выполнения метода `sayHelloToName:withSurname:` определенно произойдет утечка памяти, так как вызывающий код скорее всего не догадывается, что возвращенному объекту нужно после обработки послать сообщение *release*. Даже если он догадается это сделать, возможно, что возвращался указатель на инвариант объекта, а значит его уничтожение чревато серьезными последствиями. Хотелось бы иметь механизм самоосвобождения объектов когда либо в будущем, чтобы пользовательский код вообще не думал об их освобождении. Решается эта проблема с помощью объекта класса **NSAutoreleasePool** – самовыгружаемого пула объектов.

После создания объекта такого класса всем объектам, созданным после него, можно послать сообщения *autorelease*. При этом данный объект помещается в текущий (последний созданный) самовыгружаемый пул. Когда некий пул получит сообщение *release*, то он отошлет такое же сообщение и всем своим объектам, уменьшая их счетчик ссылок (по сути, уничтожая). Таким образом, объект, помещенный в самовыгружаемый пул, продолжает жить и занимать память во все время жизни пула. Это удобно для небольших временных объектов, но может с течением времени занять значительную часть памяти. Потому рекомендуется циклы, способные порождать большое количество временных объектов, которые отправляются в самовыгружаемый пул, обрамлять локальными (вложенными) пулами. Любой поток в программе, использующей Сосоа, должен создавать объект класса **NSAutoreleasePool** в самом начале (прежде создания других объектов), и в самом конце его уничтожать (после уничтожения всех других объектов). Функция `main()`, являющаяся главным потоком любой программы на **Objective-C**, при использовании фреймворка Сосоа должна всегда выглядеть вот так:

```
int main(int argc, char *argv[]) // или же просто main()
{
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init]; //создаем пул, он
автоматически становится текущим
    int retVal;
    //теперь можно программировать спокойно
    [pool drain]; //освобождаем пул и все объекты, помещенные в него вызовами
autorelease
    return retVal;
}
```

А корректный метод `sayHelloToName:withSurname:` теперь будет выглядеть вот так:

```
-(NSString *)sayHelloToName: (NSString *)name withSurname: (NSString
*)surname
{
    NSString *retString = [[NSString alloc] initWithFormat: @"%@ %@!", name,
surname]; //инициализируем созданный объект посредством строки формата
[retString autorelease]; //помещаем в пул, теперь retString освободится
вместе с пулом
    return retString;
}
```

К слову, метод *drain* самовыгружаемого пула аналогичен *release* с той лишь разницей, что, кроме освобождения себя самого и всех содержащихся объектов, еще дает подсказку сборщику мусора вступить в игру. Однако, это актуально только для Mac OS 10.4 и выше, так как на iOS сборки мусора нет.

Теперь рассмотрим файл **Complex.m** с определением методов класса **Complex**:


```

#import "Complex.h"

@implementation Complex
- (id)init
{
return [self initWithRe: 0.0 andIm: 0.0];
}
- (id)initWithRe: (double)re andIm: (double)im
{
if (self = [super init])
{
_re = re;
_im = im;
_format = @"re: %.11f im: %.11f"; //формат вывода по умолчанию
}
}
+ (Complex *)complexWithRe: (double)re andIm: (double)im
{
return [[[Complex alloc] initWithRe: re andIm: im] autorelease];
}
- (Complex *)add: (Complex *)other
{
return [[Complex alloc] initWithRe: _re + other->_re andIm: _im + other->_im];
}
- (Complex *)sub: (Complex *)other
{
return [[Complex alloc] initWithRe: _re - other->_re andIm: _im - other->_im];
}
- (NSString *)format
{
return _format;
}
- (void)setFormat: (NSString *)format
{//стандартный порядок действий для инварианта-объекта
[format retain];
[_format release];
_format = format;
}
- (double)re
{
return _re;
}
- (void)setRe: (double)re
{
_re = re;
}
- (double)im
{
return _im;
}
- (void)setIm: (double)im
{
_im = im;
}
- (NSString *)description
{//используем установленный формат вывода
return [NSString stringWithFormat: _format, _re, _im];
}
- (void)dealloc
{
[_format release]; //для этого и переопределялся dealloc
[super dealloc];
}

```

```

}
@end

```

Конструктор по умолчанию вызывает специализированный конструктор с определенными начальными параметрами. Метод *complexWithRe:andIm:* возвращает инициализированный объект класса **Complex**, размещенный в текущем самовыгружаемом пуле. То же самое делает и метод *description*, возвращая объект класса **NSString**. Вот пример программы, где используется класс **Complex**:

```

#import "Complex.h"
#import <stdio.h> //для printf()

int main()
{
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
    Complex *num1 = [[Complex alloc] init]; //0.0+0.0*i
    Complex *num2 = [[Complex alloc] initWithRe: 1.5 andIm: -2];
    //1.5-2.0*i
    Complex *num3 = [Complex complexWithRe: 5 andIm: 7];
    //5.0+7.0*i
    printf("%s\n", [[num2 description] cStringUsingEncoding:
    NSASCIIStringEncoding]); //вывод> re: 1.5 im: -2.0
    printf("%s\n", [[[num2 add: num3] description] cStringUsingEncoding:
    NSASCIIStringEncoding]); //вывод> re: 5.5 im: 5.0
    [num1 setRe: [num2 re]]; //задаем _re для num1 как у num2
    [num1 setIm: [num3 im]]; //задаем _im для num1 как у num3
    [num1 setFormat: @"%.2lf+%.2lf*i"]; //меняем формат вывода для num1
    printf("%s\n", [[num1 description] cStringUsingEncoding:
    NSASCIIStringEncoding]); //вывод> 1.50+7.00*i
    [num1 release];
    [num2 release];
    //[num3 release]; не нужно, т.к. он уже в самовыгружаемом пуле
    [pool drain];
    return 0;
}

```

Если к уже написанному (а, возможно, и откомпилированному) классу нужно добавить\переопределить некоторые методы без наследования – категории позволяют это сделать без особых усилий:

//файл "CategorizedComplex.h"

```

#import "Complex.h"
@interface Complex (CategorizedComplex)
- (Complex *)mul: (Complex *)other;
- (Complex *)div: (Complex *)other;
@end

```

//файл "CategorizedComplex.m"

```

#import "CategorizedComplex.h"

```

```

@implementation Complex (CategorizedComplex)
- (Complex *)mul: (Complex *)other
{
    return [Complex complexWithRe: _re * other->_re - _im * other->_im andIm: _re
    * other->_im + _im * other->_re];
}
- (Complex *)div: (Complex *)other
{
    double retRe, retIm, denominator;
    denominator = other->_re * other->_re + other->_im * other->_im;
    if (!denominator)
        return nil;
    retRe = (_re * other->_re + _im * other->_im) / denominator;
    retIm = (_im * other->_re - _re * other->_im) / denominator;
    return [Complex complexWithRe: retRe andIm: retIm];
}

```

```
}
@end
```

А пользоваться этим можно вот так:

```
CategorizedComplex *num1 = [[CategorizedComplex alloc] initWithRe: 1 andIm:
999];
Complex *num2 = [Complex complexWithRe: 0 andIm: 0];
CategorizedComplex *num3 = [num1 div: num2]; //num3 == nil
```

Расширения несут добрую службу как безымянные категории:

```
//файл "CategorizedComplex.m"
#import "CategorizedComplex.h"
@interface Complex ()
- (void)zeroComplex; //тайный метод для обнуления числа
@end

@implementation Complex
- (void)zeroComplex //им могут пользоваться только методы самого класса
{
    _re = 0;
    _im = 0;
}
@end
```

7.7 Протоколы

Протоколы описывают методы, которые могут быть реализованы каким-либо классом. Это подобно объявлению интерфейсов в C++ для построения СОМ модели.

Цели для которых используются протоколы:

- Ожидание, что класс поддерживающий протокол выполнит описанные в протоколе функции.
- Поддержка протокола на уровне объекта, не раскрывая методы и реализацию самого класса.
- В виду отсутствия множественного наследования - объединить общие черты нескольких классов.

Объявление формального протокола гарантирует, что все методы объявленные протоколом будут реализованы классом.

Простейшее описание протокола:

```
@protocol ProtocolName
method declarations
@end
```

За словом **@protocol** следует перечисление методов, также как и в обычном объявлении методов класса.

Протокол может быть унаследован от произвольного количества других протоколов:

```
@protocol MyProtocol <Protocol1, Protocol2, Serializable, Drawable>
```

Для указания классу поддержки определенного протокола необходимо включить заголовочный файл протокола в файл описания класса, и после в <скобках, через запятую>перечислить протоколы:

```
#import "protocolName.h"
@interface MyClass : SuperClass
    описание класса
@end
```

В файле реализации класса необходимо реализовать все методы, которые описывает протокол! Но - в **Objective-C 2.0** добавлены еще два новых модификатора **@optional** и **@required**:

```
@protocol asdfg
- (void) readMy;
```

```
@optional
- (void) saveReadMy;
- (NSString*) loadReadMy;

@required
- (NSInteger) countReadMy;
@end
```

Данный протокол показывает, что протокол *asdfg* должен реализовать методы *readMy* и *countReadMy*, а методы *saveReadMy* и *loadReadMy* необязательны к реализации.

Добавление категории к классу **NSObject** - называется созданием неформального протокола. При работе с неформальными протоколами мы реализуем только те методы, которые хотим.

7.8 Категории

Язык **Objective-C** обладает возможностью добавлять новые методы к уже существующим классам. Аналогичной возможностью обладает язык Ruby, C#, JavaScript и другие. При этом не требуется исходников класса и добавленные методы автоматически становятся доступными всем классам, унаследованным от изменяемого. Так можно добавить новый метод классу **NSObject** и этот метод автоматически добавится во все остальные классы.

Механизм, позволяющий расширять уже существующие классы (путем добавления новых методов, новые *instance*-переменные добавить таким образом нельзя), называется категорией.

Категория имеет свое имя, список методов и имя класса, который она расширяет. Описание категории имеет следующий вид:

```
#import "ClassName.h"
@interface ClassName ( CategoryName )
    объявление методов
@end
```

Реализация категории выглядит следующим образом:

```
#import "CategoryName.h"
@implementation ClassName ( CategoryName )
    реализация методов
@end
```

Ограничения при создании категорий:

- Невозможность добавления переменных.
- Возможная коллизия имен с самим классом, поэтому необходимо использовать оригинальные префиксы в наименовании своих методов.

7.9 Поток

Objective-C поддерживает многопоточность в приложениях. Проблемы могут возникнуть если два и более потока решат одновременно модифицировать одни и те-же данные экземпляра класса. Чтобы избежать данных коллизий необходимо синхронизировать потоки во время изменения критических значений переменных

Для этого используется директива **@synchronized()**, которая переключает приложение в однопотоковый режим на время выполнения критического кода:

```
- (void)criticalMethod
{
    @synchronized(self) {
        // Критический код.
    }
}
```

```

...
}
}

```

В скрытом виде **Objective-C** синхронизирует код в сеттерах свойств **@properties**, используемых по умолчанию как *atomic*.

7.10 Исключения

Поддержка компилятором исключений базируется на четырех директивах компилятора:

- **@try** - определяет блок кода, который является областью обработки исключений: код, который потенциально может вызвать исключение.
- **@catch()** - определяет блок, содержащий код для обработки исключения в **@try** блоке. Параметр **@catch** это объект исключения, возбужденный на локальном уровне; как правило это объект **NSException**, но могут быть и другие типы объектов, например **NSString**.
- **@finally** - определяет блок кода, который будет выполнен в любом случае, было возбуждено исключение или нет.
- **@throw** - возбуждает исключение, эта директива практически идентична в поведении методу *raise NSException*. Вы обычно возбуждаете *NSException* объекты, но можете не ограничиваться ими.

Важно: Cocoa framework может обрабатывать только исключения *NSException*, поэтому, чтобы не наплодить ошибок старайтесь, также придерживаться *NSException* в своем коде для возбуждения исключений, и их обработки, но это ни в коей мере не ограничивает Вас в использовании своих объектов возбуждения и обработки исключений.

Следующий пример покажет обработку и возбуждение исключения *NSException*, а также действие *@finally*, которое будет вызвано в любом случае и гарантирует в данном примере, что не произойдет утечки памяти из за объекта-примера *myObj*:

```

myObj* my=[[myObject alloc] init]; // создали какой либо свой объект
@try {
    NSException *e = [NSException
        exceptionWithName:@"FileNotFoundException"
        reason:@"File Not Found on System"
        userInfo:nil]; // Создали объект класса NSException
    @throw e; // возбудили исключение
}
@catch(NSException *e) {
    @throw; // перевозбудили e исключение
}
@finally {
    [myObj release]; // освободили свой объект вне зависимости от исключения
}

```

7.11 Блоки

Блочные объекты это C-уровневые синтаксические функции. Они похожи на стандартные функции C, но в дополнение к исполняемому коду они также могут содержать переменные, привязанные к автоматической (стеку) или управляемой (куче) памяти. Поэтому блок может поддерживать набор состояний (данные), которые он может использовать, чтобы повлиять на поведение при выполнении.

Блоки особенно полезны в качестве обратного вызова, потому что блок несет как код, который будет выполняться на обратном вызове, так и данные, необходимые во время этого выполнения.

Блоки доступны в **GCC** и **Clang** в **Xcode** начиная с **OS X v10.5**. Вы можете использовать блоки с OS X 10.6 и iOS 4.0 и выше. Блоки являются средой с открытым исходным кодом и могут быть найдены в репозитории компилятора LLVM. Блоки были также представлены в C стандарте рабочей группы в качестве N1370: Apple, Расширения

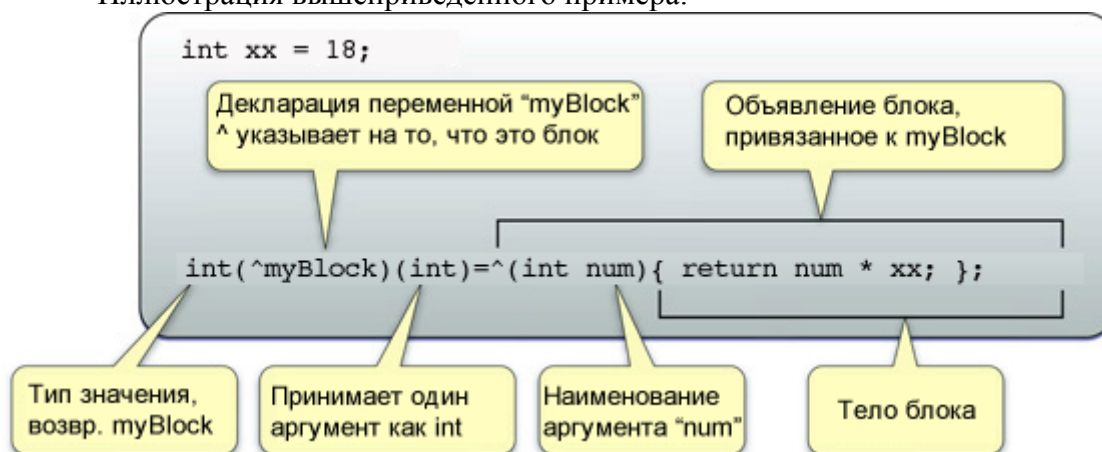
С. В Objective-C и C++ являются производными от C, блоки предназначены для работы со всеми тремя языками (а также Objective-C++).

Вы можете использовать оператор `^` для объявления переменной блока и указать на начало литерала блока. Тело самого блока содержится в `{}`, как показано в примере ниже (как обычно, в языке C, ; указывает на конец объявления):

```
int xx = 18;
```

```
int (^myBlock)(int) = ^(int num) {  
    return num * xx;  
};
```

Иллюстрация вышеприведенного примера:



Обратите внимание, что блок может использовать переменные из той же области, в которой он был определен. А также если вы объявляете блок в качестве переменной, вы можете использовать его так же, как функцию:

```
int xx = 21;
```

```
int (^myBlock)(int) = ^(int num) {  
    return num * xx;  
};
```

```
printf("%d", myBlock(2));
```

```
// Напечатает "42"
```

Во многих случаях вам не нужно объявлять переменные блока, вместо этого вы просто пишете блок буквально встраивая его там, где это требуется в качестве аргумента. В следующем примере используется `qsort_b` функция. `qsort_b` похожа на стандартную функцию `qsort_r`, но принимает блок в качестве последнего аргумента.

```
char *myCharacters[3] = { "TomJohn", "George", "Charles Condomine" };
```

```
qsort_b(myCharacters, 3, sizeof(char *), ^(const void *l, const void *r) {  
    char *left = *(char **)l;  
    char *right = *(char **)r;  
    return strcmp(left, right, 1);  
});
```

```
// myCharacters теперь { "Charles Condomine", "George", "TomJohn" }
```

Некоторые методы в Cocoa framework принимают блок в качестве аргумента, как правило, либо для выполнения операций над коллекцией объектов, или используют в качестве обратного вызова после завершения операции. В следующем примере показано, как использовать блок с методом `NSArray sortedArrayUsingComparator`. Метод принимает один аргумент-блок. Для примера, в этом случае блок определяется как `NSComparator` локальной переменной:

```
NSArray *stringsArray = @[ @"string 1",  
                           @"String 21",  
                           @"string 12",
```

```

        @"String 11",
        @"String 02" ];

static NSStringCompareOptions comparisonOptions = NSCaseInsensitiveSearch |
NSNumericSearch |
        NSWidthInsensitiveSearch | NSForcedOrderingSearch;

NSLocale *currentLocale = [NSLocale currentLocale];

NSComparator finderSortBlock = ^(id string1, id string2) {
    NSRange string1Range = NSMakeRange(0, [string1 length]);
    return [string1 compare:string2 options:comparisonOptions
range:string1Range locale:currentLocale];
};

NSArray *finderSortArray = [stringsArray
sortedArrayUsingComparator:finderSortBlock];

NSLog(@"finderSortArray: %@", finderSortArray);

/*
Вывод:

finderSortArray: (
    "string 1",
    "String 02",
    "String 11",
    "string 12",
    "String 21"
)
*/

```

Мощью функций блоков является то, что они могут изменить переменные в той же лексической области. Вы сигнализируете, что блок может изменить переменную с помощью `__block` модификатора. В адаптации примера, показанного в "Блоки в Cocoa", можно использовать блок, чтобы подсчитать, сколько строк сравниваются как равные, как показано в следующем примере. Для примера, в этом случае блок используется непосредственно и использует `currentLocale` как переменную только для чтения в блоке:

```

NSArray *stringsArray = @[ @"string 1",
                            @"String 21", // <-
                            @"string 12",
                            @"String 11",
                            @"String 21", // <-
                            @"String 21", // <-
                            @"String 02" ];

NSLocale *currentLocale = [NSLocale currentLocale];

__block NSUInteger orderedSameCount = 0;

NSArray *diacriticInsensitiveSortArray = [stringsArray
sortedArrayUsingComparator:^(id string1, id string2) {
    NSRange string1Range = NSMakeRange(0, [string1 length]);
    NSComparisonResult comparisonResult = [string1 compare:string2
options:NSDiacriticInsensitiveSearch
range:string1Range locale:currentLocale];

    if (comparisonResult == NSOrderedSame) {
        orderedSameCount++;
    }
    return comparisonResult;
}];

NSLog(@"diacriticInsensitiveSortArray: %@", diacriticInsensitiveSortArray);

```

```

NSLog(@"orderedSameCount: %d", orderedSameCount);

/*
Вывод:
diacriticInsensitiveSortArray: (
    "String 02",
    "string 1",
    "String 11",
    "string 12",
    "String 21",
    "Str\U00eeng 21",
    "Stri\U00flg 21"
)

orderedSameCount: 2
*/

```

Блочные переменные содержат ссылки на блоки. Вы объявляете их, используя синтаксис, аналогичный тому который вы используете, чтобы объявить указатель на функцию, кроме того, что вы используете `^` вместо `*`. Тип блока полностью взаимодействует с остальной частью системы C. Ниже приведены все допустимые декларации переменной блока:

```

void (^blockReturningVoidWithVoidArgument)(void);

int (^blockReturningIntWithIntAndCharArguments)(int, char);

void (^arrayOfTenBlocksReturningVoidWithIntArgument[10])(int);

```

Блоки также поддерживают переменное число (...) аргументов. В блоке, который не принимает аргументов необходимо указать *void* в списке аргументов.

Вы также можете создать типизацию блоков, что, как правило, считается лучшей практикой при использовании блока с заданной сигнатурой в нескольких местах:

```

typedef float (^MyBlockType)(float, float);

MyBlockType myFirstBlock = // ... ;
MyBlockType mySecondBlock = // ... ;

```

Вы можете использовать оператор `^` для обозначения начала блока. Он может сопровождаться списком аргументов, содержащихся в `()`. Тело блока содержится в `{ }`. В следующем примере определяется простой блок и присваивается ранее объявленной переменной (*oneFrom*).

```

float (^oneFrom)(float);

oneFrom = ^(float aFloat) {
    float result = aFloat - 1.0;

    return result;
};

```

На уровне файла, вы можете использовать блок в качестве глобального буквально:

```

#import <stdio.h>

int GlobalInt = 0;
int (^getGlobalInt)(void) = ^{ return GlobalInt; };

```

7.12 Наследование, инкапсуляция, полиморфизм

Инкапсуляция — свойство языка программирования, позволяющее объединить и защитить данные и код в объекте и скрыть реализацию объекта от пользователя (прикладного программиста). При этом пользователю предоставляется только спецификация (интерфейс) объекта.

Пользователь может взаимодействовать с объектом только через этот интерфейс. Реализуется с помощью ключевого слова: *public*. Пользователь не может использовать

закрытые данные и методы. Реализуется с помощью ключевых слов: `private`, `protected`, `internal`.

Соккрытие реализации целесообразно применять в следующих случаях:

1. предельная локализация изменений при необходимости таких изменений,
2. прогнозируемость изменений (какие изменения в коде надо сделать для заданного изменения функциональности) и прогнозируемость последствий изменений.

Полиморфизм — возможность объектов с одинаковой спецификацией иметь различную реализацию. Язык программирования поддерживает полиморфизм, если классы с одинаковой спецификацией могут иметь различную реализацию — например, реализация класса может быть изменена в процессе наследования. Кратко смысл полиморфизма можно выразить фразой: «Один интерфейс, множество реализаций». Полиморфизм позволяет писать более абстрактные программы и повысить коэффициент повторного использования кода. Общие свойства объектов объединяются в систему, которую могут называть по-разному — интерфейс, класс.

Общность имеет внешнее и внутреннее выражение:

1. внешняя общность проявляется как одинаковый набор методов с одинаковыми именами и сигнатурами (именем методов и типами аргументов и их количеством);
2. внутренняя общность — одинаковая функциональность методов. Её можно описать интуитивно или выразить в виде строгих законов, правил, которым должны подчиняться методы. Возможность приписывать разную функциональность одному методу (функции, операции) называется перегрузкой метода (перегрузкой функций, перегрузкой операций).

В объектно-ориентированных языках класс является абстрактным типом данных.

Полиморфизм реализуется с помощью наследования классов и виртуальных функций. Класс-потомок наследует сигнатуры методов класса-родителя, а реализация, в результате переопределения метода, этих методов может быть другой, соответствующей специфике класса-потомка. Другие функции могут работать с объектом класса-родителя, но при этом вместо него во время исполнения будет подставляться один из классов-потомков. Это называется поздним связыванием.

Класс-потомок сам может быть родителем. Это позволяет строить сложные схемы наследования — древовидные или сетевидные. Абстрактные (или чисто виртуальные) методы не имеют реализации вообще (на самом деле некоторые языки, например C++, допускают реализацию абстрактных методов в родительском классе). Они специально предназначены для наследования. Их реализация должна быть определена в классах-потомках.

Класс может наследовать функциональность от нескольких классов. Это называется множественным наследованием. Множественное наследование создаёт известную проблему (в C++), когда класс наследуется от нескольких классов-посредников, которые в свою очередь наследуются от одного класса (так называемая «Проблема ромба»): если метод общего предка был переопределён в посредниках, неизвестно, какую реализацию метода должен наследовать общий потомок. Решается эта проблема путём отказа от множественного наследования для классов и разрешением множественного наследования для полностью абстрактных классов (то есть интерфейсов) (C#, Delphi, Java), либо через виртуальное наследование (C++).

Наследование — механизм объектно-ориентированного программирования, позволяющий описать новый класс на основе уже существующего (родительского), при этом свойства и функциональность родительского класса заимствуются новым классом. Другими словами, класс-наследник реализует спецификацию уже существующего класса (базовый класс). Это позволяет обращаться с объектами класса-наследника точно так же, как с объектами базового класса.

Простое наследование. Класс, от которого произошло наследование, называется базовым или родительским (англ. `base class`). Классы, которые произошли от базового,

называются потомками, наследниками или производными классами (англ. derived class).

В некоторых языках используются абстрактные классы. Абстрактный класс — это класс, содержащий хотя бы один абстрактный метод, он описан в программе, имеет поля, методы и не может использоваться для непосредственного создания объекта. То есть от абстрактного класса можно только наследовать. Объекты создаются только на основе производных классов, наследованных от абстрактного. Например, абстрактным классом может быть базовый класс «сотрудник вуза», от которого наследуются классы «аспирант», «профессор» и т. д. Так как производные классы имеют общие поля и функции (например, поле «год рождения»), то эти члены класса могут быть описаны в базовом классе. В программе создаются объекты на основе классов «аспирант», «профессор», но нет смысла создавать объект на основе класса «сотрудник вуза».

Множественное наследование. При множественном наследовании у класса может быть более одного предка. В этом случае класс наследует методы всех предков. Достоинства такого подхода в большей гибкости. Множественное наследование реализовано в C++. Из других языков, предоставляющих эту возможность, можно отметить Python и Эйфель. Множественное наследование поддерживается в языке UML.

Множественное наследование — потенциальный источник ошибок, которые могут возникнуть из-за наличия одинаковых имен методов в предках. В языках, которые позиционируются как наследники C++ (Java, C# и др.), от множественного наследования было решено отказаться в пользу интерфейсов. Практически всегда можно обойтись без использования данного механизма. Однако, если такая необходимость все-таки возникла, то, для разрешения конфликтов использования наследованных методов с одинаковыми именами, возможно, например, применить операцию расширения видимости — «::» — для вызова конкретного метода конкретного родителя.

Большинство современных объектно-ориентированных языков программирования (C#, Java, Delphi и др.) поддерживают возможность одновременно наследоваться от класса-предка и реализовать методы нескольких интерфейсов одним и тем же классом. Этот механизм позволяет во многом заменить множественное наследование — методы интерфейсов необходимо переопределять явно, что исключает ошибки при наследовании функциональности одинаковых методов различных классов-предков.

Единый базовый класс.

В ряде языков программирования все классы явно или неявно наследуются от некоего базового класса. Smalltalk был одним из первых языков, в которых использовалась эта концепция. К таким языкам относятся Objective-C (NSObject), Eiffel (ANY), Java (java.lang.Object), C# (System.Object), Delphi (TObject).

Абстракция — в объектно-ориентированном программировании это придание объекту характеристик, которые отличают его от всех других объектов, четко определяя его концептуальные границы. Основная идея состоит в том, чтобы отделить способ использования составных объектов данных от деталей их реализации в виде более простых объектов, подобно тому, как функциональная абстракция разделяет способ использования функции и деталей её реализации в терминах более примитивных функций, таким образом, данные обрабатываются функцией высокого уровня с помощью вызова функций низкого уровня. Такой подход является основой объектно-ориентированного программирования. Это позволяет работать с объектами, не вдаваясь в особенности их реализации. В каждом конкретном случае применяется тот или иной подход: инкапсуляция, полиморфизм или наследование. Например, при необходимости обратиться к скрытым данным объекта, следует воспользоваться инкапсуляцией, создав, так называемую, функцию доступа или свойство.

Абстракция данных — популярная и, в общем, неверно определяемая техника программирования. Фундаментальная идея состоит в разделении несущественных деталей реализации подпрограммы и характеристик существенных для корректного ее

использования. Такое разделение может быть выражено через специальный «интерфейс», сосредотачивающий описание всех возможных применений программы.

8 Управление памятью

8.1 Управление памятью Сосоа

Управление памятью приложения, во время выполнения программы - процесс выделения памяти под объекты, и освобождение ее после использования. Хорошо написанная программа использует мало памяти, насколько это возможно. В **Objective-C**, это также может рассматриваться как способ распределения ограниченного объема памяти между различными частями кода и данных. Данное руководство показывает, как можно управлять памятью Вашего приложения, управляя жизненным циклом Ваших объектов, освобождая их, когда они больше не нужны [3].

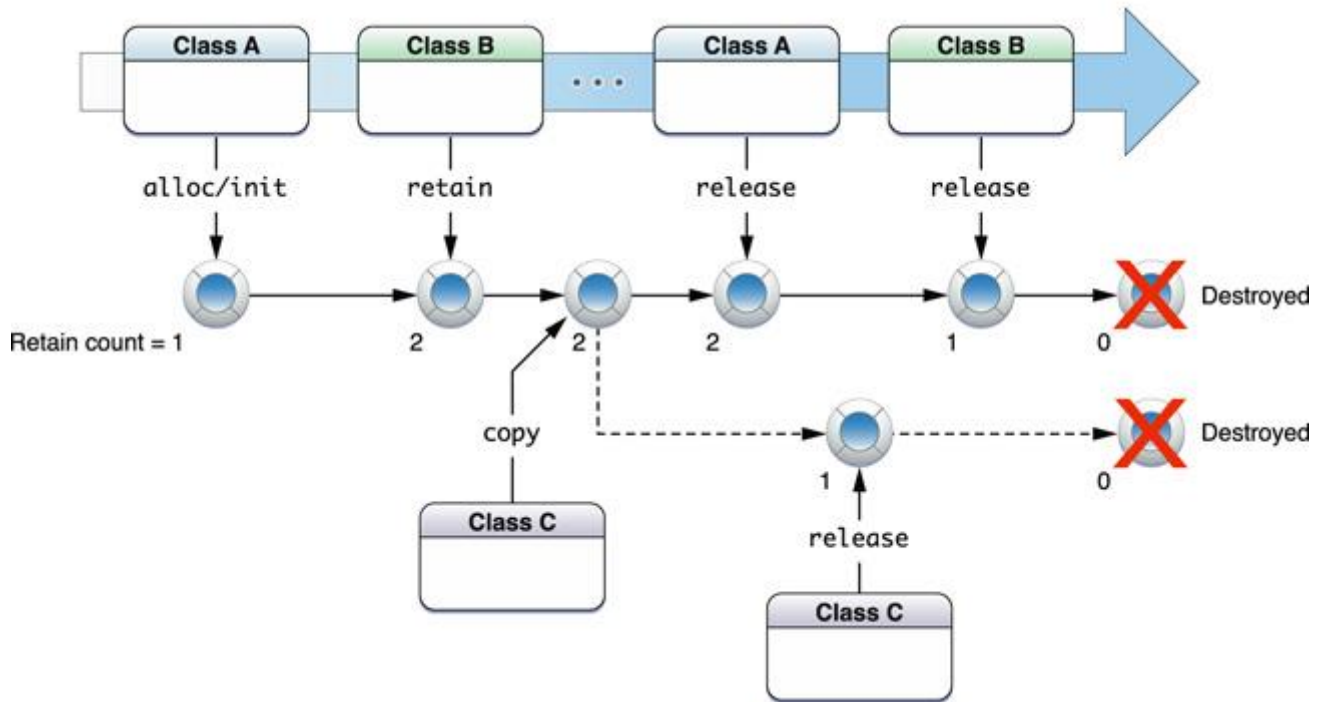


Рисунок 8.1 Схема управления памятью

Objective-C предоставляет три метода управления памятью:

1. В методике, описанной в данной главе, называемой "manual retain-release" (ручное сохранение-освобождение) или **MRR**, вы явно управляете памятью, отслеживая объекты, которые у вас есть. Это реализуется с помощью модели, известной как подсчет ссылок, что **Foundation** класса **NSObject** обеспечивает совместно со средой выполнения.
2. В **Automatic Reference Counting** (**автоматическом подсчете ссылок**), или **ARC**, система использует тот же подсчет ссылок, что и система **MRR**, но он вставляет соответствующий вызов метода управления памятью за вас во время компиляции. Вам настоятельно рекомендуется использовать **ARC** для новых проектов. Если вы используете **ARC**, обычно нет необходимости, понимать внутреннюю реализацию описанную в этом документе, хотя она может быть полезной в некоторых ситуациях.
3. В **garbage collection** (сборе мусора), система автоматически отслеживает, какие объекты владеют другими объектами. Затем она автоматически освобождает (или собирает мусор) объекты, на которые больше не ссылаются. Метод использует другой механизм, нежели, чем у используемых в **MRR** и **ARC**, и поддерживается только в среде выполнения на Mac OS X, а не IOS.

Если вы планируете писать код для IOS, вы должны использовать явное управление памятью (предмет данного руководства). Кроме того, если вы планируете

создавать библиотеки подпрограмм, плагины, или общий код, который может быть загружен в любом процессе со сборкой мусора или без сбора мусора, вы можете написать свой код с помощью методов управления памятью, описанных в этой главе. (Убедитесь, что вы затем проверите код в **Xcode**, со сборкой мусора выключенной и включенной.)

Существуют две основные проблемы, возникающие в результате неправильного управления памятью:

1. Освобождение или перезапись данных, которые используются до сих пор. Это приводит к повреждению памяти, и как правило, приводит к сбою в своем приложении, или еще хуже, повреждает данные пользователя.
2. Не освобождение памяти, выделенной под данные, которые больше не используются, приводит к утечке памяти. Утечка памяти это выделенная и не освобожденная память, которая больше не используется, что в свою очередь приводит к все большему выделению памяти и как результат - снижение производительности системы, а в случае с IOS и вылет приложения

Думаю способ управления памятью с точки зрения подсчета ссылок, может показаться, зачастую контрпродуктивным, потому что вы склонны рассматривать управление памятью с точки зрения деталей реализации, а не с точки зрения ваших фактических целей. Вместо этого, вы должны думать об управлении памятью с точки зрения владения объектом и объектом графов.

Cocoa использует прямую конвертацию имен, чтобы показать, когда метод возвращает объект.

Хотя основная политика проста, есть некоторые практические шаги которые можно предпринять, чтобы сделать управление памятью легче, для содействия обеспечения надежности вашей программы и в то же время сводя к минимуму потребности в ресурсах.

Autorelease pools (Автоматические пулы) обеспечивают механизм, посредством которого вы можете отправить объекту "отложенное" *release* сообщение. Это полезно в ситуациях, когда вы хотите отказаться от права владения объектом, но хотите, избежать ситуации когда объект освобождается сразу (например, когда вы возвращаете объект из метода). Есть случаи, когда вы можете использовать ваши собственные autorelease pools.

Для выявления проблем с вашим кодом во время компиляции, вы можете использовать **Clang Static Analyzer** (Статический анализатор кода), который встроен в **Xcode**.

Если проблемы управления памятью все-таки возникают, есть другие средства и методы, которые можно использовать для выявления и диагностики проблем.

Вы можете использовать инструменты для отслеживания подсчета ссылок событий и искать утечки памяти. См. "Просмотр и анализ данных трассировки".

8.2 Управление политикой памяти Cocoa

Базовая модель используемая для управления памятью со счетчиком ссылок обеспечивается сочетанием методов, определенных в протоколе **NSObject** и стандартными методами именования. Класс **NSObject** также определяет метод *dealloc*, который вызывается автоматически, когда объект будет освобожден. В этой статье описаны все основные правила, которые вы должны знать, чтобы правильно управлять памятью в **Cocoa** программах, а также предоставляет примеры правильного использования [3].

Модель управления памятью основывается на владении объектами. Любой объект может иметь одного или несколько владельцев. Пока объект имеет по крайней мере одного владельца, он продолжает существовать. Если объект не имеет владельцев, система уничтожает его автоматически. Чтобы убедиться, что объекты освобождаются, когда у вас есть объект, **Cocoa** устанавливает следующую политику:

- **Вы владеете каким-либо объектом** вами созданным. Вы можете создать объект, используя метод, имя которого начинается с "*alloc*", "*new*", "*copy*", или "*mutableCopy*".
- **Вы можете стать владельцем объекта, используя retain**. Полученный объект, как правило, гарантированно остается в силе в течение выполнения метода, в котором он был получен, и этот метод также может безопасно вернуть объект к вызвавшему его методу. Вы используете *retain* в двух случаях: (1) В реализации метода доступа или инициализации метода, чтобы стать владельцем объекта, который нужно сохранить как значение свойства, (2) Для предотвращения освобождения объекта как побочного эффекта от некоторых других операций.
- Если объект вам больше не нужен, вы должны отказаться от права собственности на него. Вы отказываетесь от права собственности на объект, пошлав ему сообщение *release* или *autorelease* сообщение. В Сосоа терминологии, из отказа от права владения объектом, как правило следует, "освобождение" объекта.
- Вы не должны отказаться от права владения объектом, которого у вас нет. Это всего лишь следствие предыдущих правил политики, заявленных в явном виде.

Простой пример:

```
{
    Person *aPerson = [[Person alloc] init];

    // ...

    NSString *name = person.fullName;

    // ...

    [aPerson release];
}
```

Объект *Person* создается с помощью метода *alloc*, так что впоследствии отправлено сообщение *release*, когда он больше не нужен. Переменная *name* объекта *Person* не получалась с помощью одного из захватывающих метода, поэтому ей не будет отправлено сообщение *release*. Обратите внимание, что пример использует *release*, а не *autorelease*.

Вы можете использовать *autorelease*, когда вам нужно отправить отложенное сообщение *release*, обычно при возврате объекта из метода. Например, вы могли бы реализовать *FullName* метод следующим образом:

```
- (NSString *)fullName {
    NSString *string = [[NSString alloc] initWithFormat:@"%s %s",
        self.firstName, self.lastName] autorelease];

    return string;
}
```

Вы владеете строкой, возвращаемой *alloc*. Чтобы соблюдать правила управления памятью, вы должны отказаться от собственности на строку, прежде чем потерять ссылку на нее. Если вы используете *release*, строка будет освобождена, еще до возврата (и метод будет возвращать неверный объект). Используя *autorelease*, Вы подтверждаете, что хотите отказаться от владения объектом, но вы позволяете вызывающему методу использовать возвращенную строку, пока не освободится.

Кроме того, можно осуществить метод *FullName* следующим образом:

```
- (NSString *)fullName {
```

```

NSString *string = [NSString stringWithFormat:@"%@" "%@",
    self.firstName, self.lastName];

return string;
}

```

В соответствии с основными правилами, вы не владеете строкой, возвращаемой *stringWithFormat:*, так что можно спокойно вернуть строку из метода.

Для сравнения, следующий пример является неправильным:

```

- (NSString *)fullName {

    NSString *string = [[NSString alloc] initWithFormat:@"%@" "%@",
        self.firstName, self.lastName];

    return string;
}

```

Согласно соглашению об именах, нет ничего, чтобы обозначить, что вызванный метод *FullName* владеет возвращаемой строкой. Поэтому вызывающий метод не имеет никаких оснований для освобождения возвращаемой строки, таким образом, произойдет утечка памяти, выделенной под объект *string*.

Некоторые методы в **Cocoa** указывают, что объект возвращается по ссылке (то есть, они не имеют аргумента типа *ClassName *** или *id **). Данная закономерность заключается в использовании *NSError* объекта, который содержит информацию об ошибках, если они происходят, о чем свидетельствуют *initWithContentsOfURL:options:error:* (*NSData*) и *initWithContentsOfFile:encoding:error:* (*NSString*).

В этих случаях применяются те же правила, как уже было описано. При вызове любого из этих методов, вы не создаете *NSError* объект, так что вы не являетесь его владельцем. Поэтому нет необходимости, его освобождать, как показано в следующем примере:

```

NSString *fileName = <#Get a file name#>;

NSError *error = nil;

NSString *string = [[NSString alloc] initWithContentsOfFile:fileName
    encoding:NSUTF8StringEncoding error:&error];

if (string == nil) {

    // Deal with error...
}

// ...

[string release];

```

Класс *NSObject* определяет метод *dealloc*, который вызывается автоматически, когда объект не имеет владельцев и его память будет утилизирована, в терминологии **Cocoa** это называется "освободил" или "освобождена". Роль *dealloc* метода заключается в освобождении собственной памяти объекта, и распоряжением любыми ресурсами которыми он располагает, в том числе любыми переменными экземпляра объекта, которыми он владеет.

Следующий пример иллюстрирует, как можно реализовать *dealloc* метод для класса *Person*:

```

@interface Person : NSObject {}

@property (retain) NSString *firstName;

@property (retain) NSString *lastName;

```

```

@property (assign, readonly) NSString *fullName;

@end

@implementation Person

@synthesize firstName=_firstName, lastName=_lastName;

// ...

- (void)dealloc

    [_firstName release];

    [_lastName release];

    [super dealloc];

}

@end

```

Важно: Вы никогда не должны ссылаться на *dealloc* метод другого объекта непосредственно.

Вы должны вызвать реализацию суперкласса в конце вашей реализации.

Вы не должны привязывать управление системными ресурсами, как к объектам контролируемым временем жизни.

Когда приложение завершается, *dealloc* сообщение не направляется в объекты. Потому что память процесса автоматически очищается при выходе, это более эффективно, чем позволить операционной системе, для очистки ресурсов вызывать все методы управления памятью.

8.3 Практическое управление памятью Cocoa

Хотя фундаментальные понятия, описанные в параграфе 8.2, являются простыми, есть некоторые практические шаги, которые вы можете предпринять, чтобы сделать управление памятью легче, а также в целях обеспечения надежности и прочности вашей программы, в то же время, сводя к минимуму потребности в ресурсах [3].

Если ваш класс имеет свойство, которое является объектом, вы должны убедиться, что любой объект, который устанавливается в качестве значения не освободится в то время как вы его используете. Поэтому вы должны претендовать на владение объектом, когда оно установлено. Вы также должны убедиться, что потом вы освободите объект, когда он больше не нужен.

Иногда это может показаться утомительным и педантичным, но если вы используете методы доступа последовательно, шансы возникновения проблем с управлением памятью уменьшаются значительно. Если вы используете *retain* и *release* к переменным экземпляров по всему вашему коду, вы практически наверняка делаете неправильные вещи.

Рассмотрим счетчик объекта, количество которого вы хотите установить.

```

@interface Counter : NSObject {

    NSNumber *_count;

}

@property (nonatomic, retain) NSNumber *count;

```



```
@end;
```

В свойстве объявляется два метода доступа. Как правило, вы должны попросить компилятор для синтеза методов, однако, поучительно посмотреть, как они могли бы быть реализованы.

В "get" аксессоре, вы просто возвращаете переменную экземпляра, так что нет необходимости в *retain* или *release*:

```
- (NSNumber *)count {  
    return _count;  
}
```

В "set" методе, если все остальные играют по тем же правилам, вы должны принять новое значение счетчика, которое может быть удалено в любое время, поэтому вам придется взять на себя ответственность объектно-, отправив ему *retain* сообщение, чтобы убедиться что его не будет. Вы должны также отказаться от права владения на старый объект, отправив ему *release* сообщение. (Отправка сообщения к *nil* допускается в **Objective-C**, так что реализация все равно будет работать, если *_count* до сих пор не установлена.) Вы должны послать это после [*newCount retain*] на случай, если два и более объекта захотят, ненароком ее освободить.

```
- (void)setCount:(NSNumber *)newCount {  
    [newCount retain];  
    [_count release];  
    // Make the new assignment.  
    _count = newCount;  
}
```

Предположим, вы хотите реализовать метод для сброса счетчика. У вас есть несколько вариантов. Первая реализация создает экземпляр *NSNumber* с *alloc*, который вы балансируете *release*-ом.

```
- (void)reset {  
    NSNumber *zero = [[NSNumber alloc] initWithInteger:0];  
    [self setCount:zero];  
    [zero release];  
}
```

Второй использует удобство конструктора для создания нового объекта *NSNumber*. Он уже существует поэтому нет необходимости в *retain* или *release*.

```
- (void)reset {  
    NSNumber *zero = [NSNumber numberWithInt:0];  
    [self setCount:zero];  
}
```

Обратите внимание, что оба используют *set* метод аксессуара.

Следующий код почти наверняка работает правильно для простых случаев, но как заманчиво бы это не выглядело, чтобы отказаться от методов доступа, сделав это, почти наверняка в будущем приведет к ошибке на определенном этапе (например, если вы

забыли сохранить или освободить, или если изменится семантика управления памятью для переменной экземпляра).

```
- (void)reset {  
    NSNumber *zero = [[NSNumber alloc] initWithInteger:0];  
    [_count release];  
    _count = zero;  
}
```

Отметим также, что если вы используете методику ключ-значение (KVO), то изменение переменной, таким путем, не совместимо с KVO.

Единственные места, где вы не должны использовать методы доступа для установки переменной экземпляра находятся в методах - инициализаторе и *dealloc*. Для инициализации счетчика объекта со значением нуль, вы можете реализовать метод инициализации следующим образом:

```
- init {  
    self = [super init];  
    if (self) {  
        _count = [[NSNumber alloc] initWithInteger:0];  
    }  
    return self;  
}
```

Чтобы позволить счетчику при инициализации принимать значение отличное от нуля, вы можете реализовать метод *initWithCount*: следующим образом:

```
- initWithCount:(NSNumber *)startingCount {  
    self = [super init];  
    if (self) {  
        _count = [startingCount copy];  
    }  
    return self;  
}
```

Так как переменная счетчика класса - экземпляр объекта, необходимо также реализовать *dealloc* метод. Следует отказаться от владения любыми переменными экземпляра, отправив им сообщение *release*, и в конечном счете он должен вызывать реализацию супер класса:

```
- (void)dealloc {  
    [_count release];  
    [super dealloc];  
}
```

Сохранение объекта создает сильную ссылку на этот объект. Объект не может быть освобожден, пока все его сильные ссылки не будут освобождены. Проблема, известная как сохранность цикла, это может возникнуть, если два объекта имеют циклические ссылки, то есть, у них есть сильные ссылки друг на друга (либо непосредственно, либо через цепочку других объектов, каждый из которых имеет сильную ссылку на следующую ведущую к первой).

Решением проблемы сохранности циклов является использование *weak* (слабых) ссылок. *Weak* ссылка не владеет отношениями, в которых исходный объект сохраняет объект, к которому он имеет ссылки.

Чтобы сохранить объекты графов нетронутыми, где-то должны быть сильные ссылки (если бы были только слабые ссылки, то страницы и параграфы не имели бы владельцев и были бы освобождены). **Cocoa** устанавливает конвенции, поэтому, объект "родитель" должен поддерживать сильные (*strong*) ссылки на своих "детей", а дети должны иметь слабые (*weak*) ссылки на своих родителей.

В качестве примера слабых ссылок в **Cocoa** можно привести источники данных для таблицы (*data sources*).

Вы должны быть осторожными относительно отправки сообщений на объекты, для которых вы имеете только слабые ссылки. Если вы посылаете сообщение объекту после того как он был освобожден, ваше сообщение потерпит крах. Вы должны четко определить условие, что объект является действительным. В большинстве случаев слабо указанный объект принимает слабую ссылку на другой объект, чтобы, как и в случае циклических ссылок, также нести ответственность за уведомление другого объекта, когда он освобождается. Например, при регистрации объекта с центром уведомлений, центра уведомления сохраняет слабую ссылку на объект и отправляет сообщение, когда соответствующие уведомления отправлены. Когда объект будет освобожден, необходимо отменить его регистрацию в центре уведомлений для прекращения каких-либо дальнейших сообщений на объект, который больше не существует. Аналогичным образом, когда объект делегата, будет освобожден, вы должны удалить ссылку делегата, отправив *setDelegate:* сообщение с *nil* в качестве аргумента для другого объекта. Эти сообщения, как правило, отправляются от метода *dealloc* объекта.

Контроль за политикой **Cocoa** уточняет, что полученные объекты должны, как правило, оставаться в силе во всей области вызывающего метода. Следовательно, возможно вернуть полученный объект из текущей области, не опасаясь его освобождения. Это не имеет значения для вашего приложения, если метод получения объекта возвращает кэшированный экземпляр переменной или вычисленное значение. Важно то, что объект остается в силе на данный момент как вам это нужно.

Есть редкие исключения из этого правила, которые, в первую очередь, делятся на две категории.

1. Когда объект удаляется из одной из фундаментальных коллекции классов.

```
heisenObject = [array objectAtIndex:n];

[array removeObjectAtIndex:n];

// heisenObject теперь может быть недействительным.
```

Когда объект удаляется из одной из фундаментальных коллекций классов, он отправляет *release* (а не *autorelease*) сообщение. Если коллекция была единственным владельцем объект удален, удаленный объект (*heisenObject* в примере), сразу же освобождается.

2. Когда "родительский объект" будет освобожден.

```
id parent = <#create a parent object#>;

// ...
```

```

heisenObject = [parent child] ;

[parent release]; // или к примеру: self.parent = nil;

// heisenObject теперь может быть недействительным.

```

В некоторых ситуациях вы получаете объект из другого объекта, а затем прямо или косвенно посылаете *release* родительскому объекту. Если освобождение родителя заставляет его освободиться, а родитель был единственным владельцем ребенка, то ребенок (*heisenObject* в примере) будет освобожден в то же время (предполагается, что он отправляет *release*, а не *autorelease* сообщение в метод *dealloc* родителя).

Для защиты от таких ситуаций, вы сохраняете *heisenObject* по мере его получения, и вы освободите его, когда вы закончите с этим. Например:

```

heisenObject = [[array objectAtIndex:n] retain];

[array removeObjectAtIndex:n];

// Используем heisenObject...

[heisenObject release];

```

Вы должны управлять как правило, не ограниченными ресурсами, такими как дескрипторы файлов, сетевые подключения, и буферы или кэш в *dealloc* методе. В частности, вы не должны проектировать классы так, чтобы *dealloc* был вызван, когда вы думаете, он будет вызван. Вызов *dealloc* может быть отложен или обойден, либо из-за ошибки или из-за падения приложения.

Вместо этого, если у вас есть класс, экземпляры которого управляют скудными ресурсами, следует разрабатывать приложения, которым например, когда вам больше не нужны ресурсы вы можете сказать "очистить" в этой точке. Вы, как правило, затем освобождаете объект, и *dealloc* делает свое дело, но вы не будете испытывать дополнительные проблемы, если он этого не сделает.

Проблемы могут возникнуть, если вы попытаетесь скомбинировать управление ресурсами на вершине *dealloc*. Например:

1. Заказ зависимостей от слетевших объектов графов.
Падение объектов графов по своей природе не заказывают. Если объект попадает в пул *autorelease* неожиданно, демонстрационный порядок может измениться, что может привести к неожиданным результатам.
2. Не осваивайте дефицитные ресурсы.
Утечки памяти это ошибки, которые должны быть фиксированными, но они, как правило, не сразу фатальны. Если скудные ресурсы не будут освобождены, когда вы ожидаете, что они освобождены, вы можете столкнуться с более серьезными проблемами. Если приложение выполняется из файловых дескрипторов, то к примеру, пользователь не сможет сохранить данные.
3. Логика очистки запущена на ошибочном потоке.
Если объект попадает в *autorelease* пул в неожиданное время, он будет освобожден на любом пуле того потока где это произойдет. Это легко может стать фатальным для ресурсов, которые должны быть затронуты только из одного потока.

При добавлении объекта в коллекцию (например, массив, словарь, или набор), коллекция становится его владельцем. Коллекция откажется от владения объектом, когда объект удаляется из коллекции или коллекция будет освобождена. Так, например, если вы хотите создать массив чисел вы можете выполнить одно из следующих действий:

```

NSMutableArray *array = <#Получаем изменяемый массив#>;

NSUInteger i;

```

```
// ...

for (i = 0; i < 10; i++) {

    NSNumber *convenienceNumber = [NSNumber numberWithInt:i];

    [array addObject:convenienceNumber];

}

В этом случае вы не вызываете alloc, так что нет необходимости вызывать release.
Значит, нет необходимости вызова retain для новых номеров (convenienceNumber), так как
массив это сделает.

NSMutableArray *array = <#Получаем изменяемый массив#>;

NSUInteger i;

// ...

for (i = 0; i < 10; i++) {

    NSNumber *allocatedNumber = [[NSNumber alloc] initWithInteger: i];

    [array addObject:allocatedNumber];

    [allocatedNumber release];

}
```

В этом случае, вам нужно отправить *allocatedNumber* сообщение *release* в рамках цикла, чтобы сбалансировать *alloc*. Так как массив сохранил номер, когда он был добавлен *addObject:*, он не будет освобожден, пока он в массиве.

Чтобы понять это, поставьте себя в положение человека, который реализован как класс коллекции. Вы хотите быть уверенным, что никакие объекты, о которых Вы заботитесь, не исчезнут, чтобы их сохранить вы отправляете им сообщение *retain*. Если они удаляются, вы отправляете сообщение *release* для балансировки, и все оставшиеся объекты должны направить *release* сообщение в свой собственный метод *dealloc*.

Политика владения реализуется подсчетом ссылок.

Политика владения реализуется через подсчет ссылок, обычно называемых "Счетчик ссылок" после метода *retain*. Каждый объект имеет счетчик ссылок.

- Когда вы создаете объект, счетчик ссылок устанавливается в 1.
- Когда вы посылаете сообщение *retain*, счетчик увеличивается на 1.
- Когда вы посылаете сообщение *release*, счетчик уменьшается на 1.

Когда вы посылаете объекту сообщение *autorelease*, его счетчик уменьшается на 1 на определенном этапе в будущем.

- Если счетчик ссылок достиг 0, объект уничтожается

8.4 Использование *Autorelease* пулов *Cocoa*

Autorelease пулы обеспечивают механизм, посредством которого вы можете отправить объекту "отложенные" *release* сообщения. Это полезно в ситуациях, когда вы хотите отказаться от права владения объектом, но хотите избежать ситуации когда объект освобождается сразу (например, когда вы возвращаете объект из метода). Как правило, вам не нужно создавать свои собственные *autorelease* пулы, но есть некоторые ситуации, в которых либо вы должны, или это выгодно сделать [3].

Autorelease пул это экземпляр *NSAutoreleasePool* который "содержит" другие объекты, которые получили *autorelease* сообщение. Когда *autorelease* пул будет освобожден, он посылает *release* сообщение для каждого из этих объектов. Объект может

быть введен в *autorelease* пул в несколько раз; объект получает *release* сообщение для каждого раза, что был введен в пул (пересылаются *autorelease* сообщения).

Autorelease пулы располагаются в стеке, хотя они обычно упоминаются как "вложенные". Если вы создаете новый *autorelease* пул, он добавляется к вершине стека. Когда пулы освобождаются, они удаляются из стека. Когда объекту передается *autorelease* сообщение, оно добавляется к текущей вершине пула для текущего потока.

Cocoa всегда ожидает, что *autorelease* пул доступен. Если пул не доступен, *autoreleased* объекты не освобождаются, и в вашем приложении происходит утечка памяти. Если вы отправляете *autorelease* сообщение, когда пул отсутствует, **Cocoa** сигнализирует соответствующим сообщением об ошибке. **AppKit** и **UIKit framework**-и автоматически создают пул в начале каждой итерации цикла событий, таких как события мыши или клавиатуры и сливает его в конце. Поэтому вам, как правило, не нужно создавать пул, или даже видеть код, который используется для его создания. Есть, однако три случая, когда вы можете использовать ваши собственные *autorelease* пулы:

- Если вы пишете программу, которая не основывается на пользовательском интерфейсу, такая как инструмент командной строки.
- Если вы пишете цикл, который создает много временных объектов.

Вы можете создать *autorelease* пул внутри цикла, чтобы распоряжаться этими объектами до следующей итерации. Использование *autorelease* пула в цикле позволяет снизить затрачиваемый объем памяти для приложения.

- Если вы порождаете вторичный поток.

Вы должны создать свой собственный *autorelease* пул, как только поток начинает выполнение, в противном случае, в вашем приложении будет утечка объектов.

Вы создаете *NSAutoreleasePool* объект с обычным *alloc* и сообщением *init*, и вы распоряжаетесь его сливом (*drain method*). За исключением, если вы посылаете *autorelease* или *retain* в *autorelease* пул. *Autorelease* пул всегда должен быть слит (*drain*) в том же контексте (например, вызов метода или функции, или тела цикла), в котором он был создан.

Autorelease пулы используются "встроенными". Надо взять за правило, что нет такой причины, почему вы должны сделать *autorelease* пул переменной экземпляра объекта.

Многие программы создают временные объекты, которые авто освобождаются. Эти объекты хранятся в памяти программы, пока пул не сольют. Вы можете использовать локальные *autorelease* пулы, чтобы помочь уменьшить пиковый объем памяти. Когда ваш пул сливают, временные объекты освобождаются, что обычно приводит к их уничтожению, тем самым снижая потребление памяти программой.

Следующий пример показывает, как можно использовать локальные *autorelease* пулы в цикле.

```
NSArray *urls = <# Массив файлов URLs #>;

for (NSURL *url in urls) {

    NSAutoreleasePool *loopPool = [[NSAutoreleasePool alloc] init];

    NSError *error = nil;

    NSString *fileContents = [[[NSString alloc] initWithContentsOfURL:url
        encoding:NSUTF8StringEncoding error:&error] autorelease];

    /* создаются и автоуничтожаются множественные объекты. */

    [loopPool drain];
}
```

For цикл обрабатывает один файл за раз. Объект *NSAutoreleasePool* создан в начале этого цикла и осушен в конце. Таким образом, каждый объект послал сообщение *autorelease* внутри цикла (например, *fileContents*) добавляется к *loopPool*, и когда *loopPool* сливают в конце цикла, эти объекты будут освобождены.

После того как *autorelease* пул будет освобожден, вы должны рассматривать любой объект, который был в то время как *autoreleased* и пул был активен, как "утилизированный". Не отправляйте сообщение на этот объект или верните его в вызывающий метод вашего метода. Если вы должны использовать временный объект за пределами *autorelease* контекста, вы можете сделать это, отправив *retain* сообщение для объекта в контексте, а затем отправить его *autorelease* после слива пула, как показано в следующем примере:

```
- (id) findMatchingObject:(id) anObject {
    id match = nil;
    while (match == nil) {
        NSAutoreleasePool *subPool = [[NSAutoreleasePool alloc] init];
        /* Выполните поиск, который создает много временных объектов. */
        match = [self expensiveSearchForObject:anObject];
        if (match != nil) {
            [match retain]; /* Keep match around. */
        }
        [subPool drain];
    }
    return [match autorelease]; /* Верните соответствующий match */
}
```

Отправляя *retain* к *match* в то время как подпул активен, и, отправив *autorelease* к нему после слива подпула, *match* фактически переехал из подпула в пул, который ранее был активен. Это увеличивает срок службы *match* и позволяет ему получать сообщения вне петли и возвращаться вызывающему методу метода *findMatchingObject*:

Каждый поток в **Cocoa** приложении сохраняет свой собственный стек объектов *NSAutoreleasePool*. Когда поток завершается, он автоматически освобождает все *autorelease* пулы, связанные с собой. Если вы пишете *Foundation*- приложение, или если вы отделяете поток, вам нужно создать свой собственный *autorelease* пул.

Если ваше приложение или поток долгожитель и потенциально создает много *autoreleased* объектов, вы должны периодически разрушать и создавать *autorelease* пул (как это делается в основном потоке), иначе *autoreleased* объекты накапливаются, и ваш объем памяти вырастает. Если ваш отдельный поток не делает **Cocoa** вызовы, вам не нужно создавать *autorelease* пул.

Примечание: Если вы создаете вторичные потоки, использующие **POSIX API** вместо *NSThread*, вы не можете использовать **Cocoa**, в том числе *NSAutoreleasePool*, если только **Cocoa** не находится в многопоточном режиме. **Cocoa** входит в многопоточный режим только после отсоединения ее первого объект *NSThread*. Для использования **Cocoa** на вторичных потоках **POSIX**, приложение должно сперва отделить хотя бы один *NSThread* объект, который может сразу завершиться. Вы можете проверить **Cocoa** нахождение в режиме многопоточности методом класса *NSThread* - *isMultiThreaded*.

Это обобщенный разговор о вложенных *autorelease* пулах, но вы можете также думать о вложенных *autorelease* пулах как о стеке, с "внутренним" *autorelease* пулом находящемся на вершине стека. Каждый поток в программе поддерживает стек *autorelease* пулов. При создании *autorelease* пула, он помещается на вершину стека *autorelease* пула текущего потока. Когда объекту передается *autorelease* сообщение или когда он передается в качестве аргумента для метода *AddObject*: он всегда помещается в *autorelease* пул на вершину стека.

Объем *autorelease* пула поэтому определяются его положением в стеке. Верхний пул это пул в который добавляются *autoreleased* объекты. Если другой пул создается, нынешний верхний пул выходит из области видимости, пока новый пул сливается (в этот момент исходный пул вновь становится верхним пулом). Он выходит из области видимости постоянно, когда сам сливается.

Если вы сливаете *autorelease* пул, который не на вершине стека, все (не слитые) *autorelease* пулы над ним на стеке сливаются (и все их объекты отправляют соответствующие *release* сообщения). Если вы пренебрегаете отправкой *drain* на *autorelease* пул, когда вы закончите с ним работать (что не рекомендуется), пул сливается, когда один из *autorelease* пулов, в которых он находится, сливают.

Такое поведение имеет значение для исключительных условий. Если происходит исключение, и поток вдруг переводится из текущего контекста, пулы, связанные с этим потоком сливают. Однако, если пул не верхний на стеке потока, все бассейны выше слитых пулов также сливаются (освобождение всех своих объектов в процессе). Верхним *autorelease* пулом на стеке потока становится пул, ниже ранее слитых пулов, связанный с исключительным условием. Из-за такого поведения, обработчику исключений не нужно, освобождать объекты, в которые были направлены *autorelease*. Для обработчика исключений нет необходимости в отправке *release* его *autorelease* пулу, если обработчик возбужден исключением.

Хотя система сбора мусора не использует *autorelease* пулов, они могут быть полезными в обеспечении намеков коллектору при разработке гибридной структуры (то есть те, которые могут быть использованы в сборщике мусора и со счетчиком ссылок среды). Точки, где *autorelease* пулы спускаются, как правило, также и точки, в которых было бы полезно наметнуть сборщику мусора, что сбор, вероятно, будет оправданным.

В среде сборки мусора, *release* холостая команда (ничего не делающая инструкция). *NSAutoreleasePool*, следовательно, обеспечивает *drain* метод, который в счетчиком ссылок среды ведет себя так же, как вызов *release*, но который в среде мусора инициирует сбор мусора (если памяти, выделенной с момента последней сборки больше текущего порога). Как правило, таким образом, вы должны использовать *drain*, а не *release* для распоряжения *autorelease* пулом.

8.5 Автоматический подсчет ссылок (ARC) Cocoa

Автоматический подсчет ссылок (*Automatic Reference Counting (ARC)*) является компиляторной функцией, которая обеспечивает автоматическое управление памятью в **Objective-C** объектах. Вместо того, чтобы думать о сохранении и освобождении объектов, ARC позволяет сосредоточиться на непосредственном коде Вашего приложения [3].

ARC работает путем добавления кода во время компиляции, чтобы время жизни объекта было ровно столько, сколько необходимо, но не более того. Концептуально, это то же управление памятью, что и ручной подсчет ссылок (описанное в практическом управлении памятью) путем добавления соответствующего кода управления памятью, за вас.

ARC поддерживается начиная с **Xcode 4.2** для **Mac OS X v10.6** и **v10.7** (64-bit applications), а также **iOS 4** и **iOS 5**. Слабые (*weak*) ссылки не поддерживаются в Mac OS X v10.6 и iOS 4 и более ранних.

Кроме того, что **ARC** вставляет в код за Вас *retain/release/autorelease*, он так-же автоматически создает заполненный метод *dealloc*, если Вам необходимо освободить объекты, используемые извне и не использующие **ARC**, вы можете по прежнему написать свой метод *dealloc*, но в конце метода не надо писать *[super dealloc]*. **ARC** это сделает за Вас.

Простой пример кода с использованием **ARC**, позволяющий сосредоточиться на коде, а не на контроле ссылок.

```
- (void)test1
{
    NSNumber *year = [NSNumber numberWithInt:2012];
    NSNumber *day = [NSNumber numberWithInt:1];
    NSString *month = @"май";
    NSArray* arrDate = [NSArray arrayWithObjects:day,month,year, nil];
    NSLog(@"Месяц: %@", [arrDate objectAtIndex:1]);
}
```

Здесь *year*, *day*, *month* - являются строгими ссылками на объекты, которые мы добавляем в массив *arrDate*, соответственно **ARC** вставляет *retain* к каждой добавленной к массиву ссылке, в конце метода нам нет необходимости освобождать выделенную память под массив и объекты, т.к. **ARC** все сделает за нас во время компиляции.

ARC вводит несколько новых критериев жизни объектов, и слабые (*weak*) ссылки. Слабая ссылка не продлевает срок жизни объекта на который она указывает, и автоматически становится нулевой, когда нет более сильных (*strong*) ссылок на объект.

Важно: **ARC** не решает проблему циклических сильных ссылок, описанную в параграфе 8.3.

Квалификаторы переменных:

- *__strong* - по умолчанию. Объект остается "живым", если на него есть сильный указатель.
- *__weak* - указывается (слабая) ссылка, которая не сохраняет указанный объект живым. Слабая ссылка устанавливается в *nil*, когда на объект нет сильных ссылок.
- *__unsafe_unretained* - указывается ссылка, которая не сохраняет указанный объект живым и не устанавливается в *nil*, когда нет сильных ссылок на объект. Если объект, на который она ссылается, будет освобожден, то указатель остается болтаться со значением, которого уже нет.
- *__autoreleasing* - используется для обозначения аргументов, которые передаются по ссылке (*id **) и *autoreleased* по возвращении.

Будьте осторожны с применением *__weak* слабых ссылок, так, если Вы создадите объект только со слабой ссылкой, то нет никакой гарантии, что он доживет до своего применения и не освободится.

9 Работа со строками

9.1 Характеристики строковых объектов в Cocoa

Строковые объекты представляют собой символьные строки в *Cocoa frameworks*. Представление строк в качестве объектов позволяет использовать строки, где бы вы могли использовать другие объекты. Они также обеспечивают преимущества инкапсуляции, таким образом, строковые объекты могут использовать любую кодировку для хранения, необходимую для эффективности, просто представляя массивы символов [3].

Строковый объект реализован в виде массива символов **Юникода** (другими словами, текстовая строка). неизменной строкой является текстовая строка, которая определяется при ее создании и в дальнейшем не может быть изменена. Для создания и управления неизменяемой строкой, используют класс *NSString*. Чтобы создать и управлять строкой, которая может быть изменена после ее создания, используют *NSMutableString*.

Объекты, которые вы создаете, используя *NSString* и *NSMutableString* называются строковыми объектами (или, просто строки). Термин строка **C** относится к стандартным **C** символьным типам `char *`.

Класс *NSString* имеет два примитивных метода `—length` и `characterAtIndex:`, которые обеспечивают основу для всех других методов в его интерфейсе. `length` метод возвращает общее количество **Unicode** символов в строке. `characterAtIndex:` предоставляет доступ к каждому символу в строке по индексу, со значения индекса начинающегося с 0.

NSString объявляет методы для поиска и сравнения строк. Он также объявляет, методы для чтения числовых значений из строк, для различных объединений строк, и для преобразования строки в различных формах (таких, как кодирование и изменение регистра).

9.2 Создание и конвертация строковых объектов

NSString и его подкласс *NSMutableString* обеспечивают несколько способов создания строковых объектов, большинство из них основаны на поддержке различных кодировок. Хотя строковые объекты всегда содержат строки, как **Unicode** символы, они могут конвертировать свое содержимое и во многих других кодировках, например, 7-битные **ASCII**, **ISO Latin 1**, **EUC**, и **Shift-JIS**. Метод класса *availableStringEncodings* возвращает какие кодировки поддерживаются. Вы можете указать кодировку явно при преобразовании строки **C** или из строкового объекта, или использовать строку **C** с кодировкой по умолчанию, которая варьируется от платформы к платформе и возвращается методом класса *defaultCStringEncoding* [3].

Простейший способ создать строку объекта в исходном коде -использовать конструкцию **Objective-C**: `@"....."`:

```
NSString *temp = @"tmp/scratch";
```

Заметим, что при создании строковой константы таким образом, не следует использовать ничего, кроме 7-битных **ASCII** символов. Такой объект создается во время компиляции и существует на протяжении выполнения вашей программы. Компилятор делает такие константы объекта уникальным для каждого модуля, и они никогда не освобождаются, хотя вы можете сохранить и освободить их, как и любой другой объект. Вы также можете отправлять сообщения непосредственно в строковую константу, как и в любую другую строку:

```
BOOL same = [@"comparison" isEqualToString:myString];
```

Чтобы создать объект *NSString* из строки **C**, можно использовать такие методы, как *initWithCString:encoding:*. Очень важно правильно указать кодировку строки **C**. Подобные методы позволяют создавать строковые объекты из символов в различных

кодировках. Метод `initWithData:encoding:` позволяет конвертировать строку данных, хранящихся в объекте `NSData` в объект `NSString`.

```
char *utf8String = /* assume this exists */ ;

NSString *stringFromUTFString = [[NSString alloc]
initWithUTF8String:utf8String];

char *macOSRomanEncodedString = /* принимает существующую строку */ ;

NSString *stringFromMORString =

    [[NSString alloc] initWithCString:macOSRomanEncodedString
                                encoding:NSMacOSRomanStringEncoding];

NSData *shiftJISData = /* принимает существующую строку */ ;

NSString *stringFromShiftJISData =

    [[NSString alloc] initWithData:shiftJISData
                                encoding:NSShiftJISStringEncoding];
```

Следующий пример преобразует объект `NSString` содержащие UTF-8 в данные из символы ASCII затем обратно в объект `NSString`.

```
unichar ellipsis = 0x2026;

NSString *theString =
    [NSString stringWithFormat:@"To be continued%C", ellipsis];

NSData *asciiData = [theString dataUsingEncoding:NSUTF8StringEncoding
allowLossyConversion:YES];

NSString *asciiString = [[NSString alloc] initWithData:asciiData
encoding:NSUTF8StringEncoding];

NSLog(@"Original: %@ (length %d)", theString, [theString length]);

NSLog(@"Converted: %@ (length %d)", asciiString, [asciiString length]);

// output:

// Original: To be continued... (length 16)

// Converted: To be continued... (length 18)
```

Важно: `NSString` предоставляет ряд методов для использования непосредственно строк `C` (например, `stringWithCString:`, `initWithCString:`, `initWithCString:length:`, и `initWithCStringNoCopy:length:freeWhenDone:`). Эти методы используют строку `C` с кодировкой по умолчанию и могут привести к потере информации в процессе кодирования. Вам настоятельно не рекомендуется использовать эти методы так, как они являются устаревшими в Mac OS X v10.4.

Для создания переменной строки, вы обычно используете `stringWithFormat:` или `initWithFormat:` (или локализованных строк, `localizedStringWithFormat:`). Эти методы и их отпрыски используют строку формата в качестве шаблона, в который вы предоставляете значения (строки и другие объекты, числовые значения, и так далее) которые в нее вставляются. Они и поддерживают спецификаторы формата описанные в параграфе "Форматирование строковых объектов".

Вы можете создать строку из существующих строковых объектов, используя методы *stringByAppendingString:* и *stringByAppendingFormat:*, создающих новую строку, добавив одну строку за другой или во втором случае с помощью строки формата.

```
NSString *hString = @"Hello";
```

```
NSString *hwString = [hString stringByAppendingString:@"", world!];
```

При создании строки для представления пользователю, вы должны рассмотреть важность локализации вашего приложения. В общем, следует избегать создания видимых пользователю строк непосредственно в коде. Вместо этого вы должны использовать строки в коде, как ключ к словарю локализации, который будет поставлять видимые пользователю строки на предпочитаемом языке. Как правило, это связано с использованием *NSLocalizedString* и подобных макросов, как показано в следующем примере.

```
NSString *greeting = NSLocalizedStringFromTable
```

```
(@"Hello", @"greeting to present in first launch panel", @"greetings");
```

Вы можете по-разному комбинировать и извлекать строки. Самый простой способ для объединения двух строк это добавить одну к другой. Метод *stringByAppendingString:* возвращает строковый объект сформированный из объекта приемника и заданного аргумента.

```
NSString *beginning = @"beginning";
```

```
NSString *alphaAndOmega = [beginning stringByAppendingString:@" and end"];
```

```
// alphaAndOmega содержит новую строку: @"beginning and end"
```

Вы также можете объединить несколько строк в соответствии с шаблоном при помощи методов *initWithFormat:*, *stringWithFormat:* и *stringByAppendingFormat:*, они описаны более подробно в "Форматирование строковых объектов".

Вы можете извлекать подстроки с начала или с конца строки в пределах индекса, или из определенного диапазона, с помощью методов: *substringToIndex:*, *substringFromIndex:* и *substringWithRange:*. Можно также разделить строку на подстроки (основано на разделителе строки) методом *componentsSeparatedByString:*. Эти методы проиллюстрированы на следующих примерах, обратите внимание, что индекс в методах основанных на индексе, начинается с 0:

```
NSString *source = @"0123456789";
```

```
NSString *firstFour = [source substringToIndex:4];
```

```
// firstFour содержит: @"0123"
```

```
NSString *allButFirstThree = [source substringFromIndex:3];
```

```
// allButFirstThree содержит: @"3456789"
```

```
NSRange twoToSixRange = NSRange(2, 4);
```

```
NSString *twoToSix = [source substringWithRange:twoToSixRange];
```

```
// twoToSix содержит: @"2345"
```

```
NSArray *split = [source componentsSeparatedByString:@"45"];
```

```
// массив split содержит: { @"0123", @"6789" }
```

Если вам нужно извлечь строки, используя сопоставление с образцом, а не по индексу, вы должны использовать сканер.

Чтобы получить C строку от строкового объекта, рекомендуется использовать `UTF8String`. Это возвращает `const char *`, используя кодировку строки UTF8.

```
const char *cString = [@"Hello, world" UTF8String];
```

Полученная Вами строка C принадлежит временному объекту, и станет недействительной, если произойдет автоматическое освобождение. Если вы хотите получить постоянную строку C, необходимо создать буфер и скопировать содержимое `const char *`, возвращенную методом.

Подобные методы позволяют создавать строковые объекты из символов в кодировке Юникод или произвольной кодировке, а также для извлечения данных в этих кодировках. `initWithData:encoding:`, и `dataUsingEncoding:` выполняют эти преобразования из и в NSData объекты.

Важно: `NSString` предоставляет ряд методов для использования C строки непосредственно (например, `cString`, `cStringLength`, `lossyCString`, `getCString:`, `getCString:maxLength:`, `getCString:maxLength:range:remainingRange:`). Эти методы используют строку C с кодировкой по умолчанию и могут привести к потере информации в процессе преобразования в или из этой кодировки. Вам настоятельно рекомендуется не использовать эти методы, поскольку они являются устаревшими с Mac OS X v10.4. и iOS2.0.

В данной таблице приведены наиболее распространенные способы создания и преобразования строковых объектов.

Таблица 1 способы создания и преобразования строковых объектов

Источник	Создающий метод	Извлекающий метод
В коде	@"... " создается компилятором	N/A
UTF8 кодирование	<code>stringWithUTF8String:</code>	<code>UTF8String</code>
Unicode кодирование	<code>stringWithCharacters:length:</code>	<code>getCharacters:</code> <code>getCharacters:range:</code>
Произвольное кодирование	<code>initWithData:encoding:</code>	<code>dataUsingEncoding:</code>
Существующие строки	<code>stringByAppendingString:</code> <code>stringByAppendingFormat:</code>	N/A
Форматирование строк	<code>localizedStringWithFormat:</code> <code>initWithFormat:locale:</code>	Исп. <code>NSScanner</code>
Локализованные строки	<code>NSLocalizedString</code> и аналогичные	N/A

9.3 Форматирование строковых объектов

В данной статье описывается, как создать строку, используя строку формата, как использовать не-ASCII символы в строке формата, а также распространенная ошибка, которую разработчики делают при использовании `NSLog` или `NSLogv` [3].

`NSString` использует формат строки, синтаксис которого похож на используемый другими объектами форматирования. Он поддерживает формат символов, определенных для функции ANSI C, `printf ()`, плюс `% @` для любого объекта. Если объект реагирует на

сообщение `descriptionWithLocale:`, `NSString` отправляет такое сообщение, чтобы получить текстовое представление. В противном случае, он посылает `description` сообщения.

В формате строки, знак `'%'` объявляет заполнитель для значения, с символами, которые определяют в каком формате и виде ожидаются значения. Например, формат строки `"%d houses"` ожидает целое значение, которым будет заменено в формате выражение `'% d'`. `NSString` поддерживает формат символов, определенных для ANSI C function `printf()`, а также `"@"` для любого объекта. Если объект реагирует на сообщение `descriptionWithLocale:`, `NSString` отправляет это сообщение, чтобы получить текстовое представление, в противном случае, он посылает сообщение `description`.

Значение форматирования зависит от текущей локали пользователя, с `NSDictionary` объектом, который определяет числовые, даты, и другие видов форматов. `NSString` использует только определение локали для десятичного разделителя (выдается ключ с именем `NSDecimalSeparator`). Если вы используете метод, который не определяет язык, строка предполагает, язык по умолчанию.

Вы можете использовать метод `stringWithFormat` `NSString`-а и другие связанные с ними методы для создания строки в `printf` стиле и списком аргументов спецификаторов формата, как описано в разделе *Создание и конвертация строковых объектов*. Приведенные ниже примеры иллюстрируют, как можно создать строку, используя различные спецификаторы формата и аргументы.

```
NSString *string1 = [NSString stringWithFormat:@"A string: %@, a float:
%1.2f",

    @"string", 31415.9265];

// string1 содержит: "A string: string, a float: 31415.93"

NSNumber *number = [NSNumber numberWithInt:1234];

NSDictionary *dictionary = [NSDictionary dictionaryWithObject:[NSDate date]
    forKey:@"date"];

NSString *baseString = @"Base string.";

NSString *string2 = [baseString stringByAppendingFormat:

    @" A number: %@, a dictionary: %@", number, dictionary];

// string2 содержит: "Base string. A number: 1234,
a dictionary: {date = 2005-10-17 09:02:01 -0700; }"
```

Вы можете включать не-ASCII символы (включая Unicode) в строки, используя такие методы, как `stringWithFormat:` и `stringWithUTF8String:`.

```
NSString *s = [NSString stringWithFormat:@"Long %C dash", 0x2014];

Так как \xe2\x80\x94 является 3-байт UTF-8 строку для 0x2014, можно написать:

NSString *s = [NSString stringWithUTF8String:"Long \xe2\x80\x94 dash"];

Это не безопасно, т.к. включает высоко-битные символы в исходный код:

NSString *s = [NSString stringWithUTF8String:"Long — dash"];

NSString *s = @"Long — dash";    // Не допускается
```

Служебные функции `NSLog()` и `NSLogv()` используют услуги форматирования строки `NSString` для регистрации сообщений об ошибках. Обратите внимание, что, как

следствие этого, вам следует позаботиться об указании аргумента для этих функций. Распространенной ошибкой является указать строку, содержащую символы форматирования, как показано в следующем примере.

```
NSString *string = @"A contrived string %@";
```

```
NSLog(string);
```

```
// Приложение, вероятно, здесь закончится аварией из-за сигнала 10 (SIGBUS)
```

Здесь лучше (безопаснее) использовать строку формата для вывода в другую строку, как показано в следующем примере.

```
NSString *string = @"A contrived string %@";
```

```
NSLog(@"%@", string);
```

```
// Output: A contrived string %@"
```

Спецификаторы формата поддерживаемые методами и функциями форматирования `NSString` и `CFString` следуют IEEE printf спецификации; спецификаторы приведены в таблице 2. Обратите внимание, что вы можете также использовать "n\$" позиционные спецификаторы, такие как, %1\$s @ %2\$s. Для получения дополнительной информации см. IEEE Printf спецификации. Вы можете также использовать эти спецификаторы формата с `NSLog` функцией.

Таблица 2 Спецификаторы форматов поддерживаемых методами и функциями форматирования `NSString` и `CFString`

Спецификатор	Описание
%@	Objective-C объект, выводится в виде строки, возвращенной <code>descriptionWithLocale</code> : если таковые имеются, или <code>description</code> в противном случае. Также работает с <code>CTypeRef</code> объектами, возвращая результат <code>CFCopyDescription</code> функции.
%%	'%' символ
%d, %D, %i	32-разрядное целое число (int)
%u, %U	Беззнаковое 32-разрядное целое (unsigned int)
%hi	Знаковое 16-разрядное целое (short)
%hu	Беззнаковое 16-разрядное целое (unsigned short)
%qi	Знаковое 64-разрядное целое (long long)
%qu	Беззнаковое 64-разрядное целое (unsigned long long)
%x	Беззнаковое 32-разрядное целое число (unsigned int), напечатанное в шестнадцатеричном виде с использованием цифр 0-9 и нижнего регистра от a-f

%X	Беззнаковое 32-разрядное целое число (unsigned int), напечатанное в шестнадцатеричном виде с использованием цифр 0-9 и верхнего регистра от A-F
%qx	Беззнаковое 64-разрядное целое число (unsigned long long), напечатанное в шестнадцатеричном виде с использованием цифр 0-9 и нижнего регистра от a-f
%qX	Беззнаковое 64-разрядное целое число (unsigned long long), напечатанное в шестнадцатеричном виде с использованием цифр 0-9 и верхнего регистра от A-F
%o, %O	Беззнаковое 32-разрядное целое число (unsigned int), напечатанное в восьмеричном (octal) виде
%f	64-разрядное число с плавающей точкой (double)
%e	64-разрядное число с плавающей точкой (double), печатается в экспоненциальном виде с использованием нижнего регистра для e представления экспоненты
%E	64-разрядное число с плавающей точкой (double), печатается в экспоненциальном виде с использованием верхнего регистра для E представления экспоненты
%g	64-разрядное число с плавающей точкой (double), печатаются в стиле %e, если экспонента меньше -4 или больше или равна точности, иначе в стиле%f
%G	64-разрядное число с плавающей точкой (double), печатаются в стиле %E, если экспонента меньше -4 или больше или равна точности, иначе в стиле%f
%c	8-битный беззнаковый символ (unsigned char), печатается NSLog () как ASCII-символ, или, если не ASCII символ в восьмеричном формате \ddd или Unicode шестнадцатеричном формате \udddd, где d - цифра
%C	16-битный Unicode символ (unichar), печатается NSLog () как ASCII-символ, или, если не ASCII символ в восьмеричном формате \ddd или Unicode шестнадцатеричном формате \udddd, где d - цифра
%s	Null-завершенный массив 8-битных беззнаковых символов. % s. интерпретируется в системе кодирования, вместо, к примеру, UTF-8.
%S	Null-завершенный массив 16-битных символов Unicode
%p	Пустой указатель (void *), напечатанный в шестнадцатеричном виде с

	цифрами 0-9 и нижнем регистре a-f, с ведущими 0x
%L	Модификатор длины уточняющий, что следующий a, A, e, E, f, F, g, или G спецификатор преобразования относится к long double аргументу
%a	64-разрядное число с плавающей точкой (double), напечатанное в экспоненциальном представлении с ведущими 0x и одной шестнадцатеричной цифрой перед десятичной точкой с помощью строчной p для отображения экспоненты
%A	64-разрядное число с плавающей точкой (double), напечатанное в экспоненциальном представлении с ведущими 0x и одной шестнадцатеричной цифрой перед десятичной точкой с помощью заглавной P для отображения экспоненты
%F	64-разрядное число с плавающей точкой (double), печатается в десятичной системе счисления
%z	Модификатор длины уточняющий, что следующий d, i, o, u, x, или X спецификатор преобразования относится к size_t или соответствующему типу целого числа аргументу
%t	Модификатор длины уточняющий, что следующий d, i, o, u, x, или X спецификатор преобразования относится к ptrdiff_t или соответствующему типу целого числа без знака аргументу
%t	Модификатор длины уточняющий, что следующий d, i, o, u, x, или X спецификатор преобразования относится к intmax_t или uintmax_t аргументу

Mac OS X использует несколько типов данных, NSInteger, NSUInteger, CGFloat и CFIndex-обеспечивают последовательное средство представления значений в 32 - и 64-разрядной среде. В 32-битной среде, NSInteger и NSUInteger определяются как int и unsigned int, соответственно. В 64-разрядных средах, NSInteger и NSUInteger определяются как long и unsigned long длиной соответственно. Для того чтобы избежать необходимости использовать разные типы printf стилей спецификаторов в зависимости от платформы, вы можете использовать спецификаторы приведены в таблице 3. Обратите внимание, что в некоторых случаях вам придется привести значение.

Таблица 3 Спецификаторы формата для типов данных

Тип	Формат спецификатора	Соображения
NSInteger	%ld или %lx	Приведение значения к long
NSUInteger	%lu или %lx	Приведение значения к unsigned long

CGFloat	%f или %g	%f работает для float и double при форматировании, но см. ниже предупреждение при сканировании
CFIndex	%ld или %lx	Также как NSInteger
указатель	%p	%p добавляет 0x к началу вывода. Если вы не хотите этого, используйте %lx и приведите к long.
long long	%lld или %llx	long long является 64-разрядным на 32 - и 64-битных платформах
unsigned long long	%llu или %llx	unsigned long long является 64-разрядным на 32 - и 64-битных платформах

Следующий пример иллюстрирует использование %ld в формате и использования приведения NSInteger

```
NSInteger i = 42;

printf("%ld\n", (long)i);
```

В дополнение к соображениям, указанным в таблице 2, есть один дополнительный случай со сканированием: Вы должны различать типы float и double. Вы должны использовать %f для float, %lf для double. Если вам нужно использовать scanf (или ее вариант) с CGFloat, переключитесь на double вместо этого, и скопируйте double в CGFloat.

```
CGFloat imageWidth;

double tmp;

sscanf (str, "%lf", &tmp);

imageWidth = tmp;
```

Важно помнить, что %lf не представляет CGFloat правильно как на 32- или 64-битных платформах. Это не похоже на %ld, который работает с long и во всех случаях.

9.4 Чтение и запись строк в файлы и URL

Чтение файлов или URL-адресов с использованием NSString очень просто, при условии, что вы знаете, какую кодировку использует ресурс, если вы не знаете кодировку, чтение ресурса представляется более сложной задачей. Когда вы пишете в файл или URL, необходимо указать кодировку для последующего использования [3].

NSString обеспечивает различные методы для чтения данных из файлов и URL-адресов. В общем, это намного проще для чтения данных, если вы знаете кодировку. Если у вас есть обычный текст и Вам не известна кодировка, вы уже находитесь в трудном положении. Вы должны избегать этой ситуации, если это вообще возможно, что требует для использования текстовых файлов указывать кодировку (желательно UTF-8 или UTF-16 + BOM).

Для чтения из файла или URL, для которого вы знаете, кодировку, вы используете stringWithContentsOfFile:encoding:error: или stringWithContentsOfURL:encoding:error: или соответствующей init... метод, как показано в следующем примере.

```
NSString *path = ...;

NSError *error;
```

```

NSString *stringFromFileAtPath = [[NSString alloc]
                                   initWithContentsOfFile:path
                                   encoding:NSUTF8StringEncoding
                                   error:&error];

if (stringFromFileAtPath == nil) {
    // произошла ошибка

    NSLog(@"Error reading file at %@\n%@",
          path, [error localizedFailureReason]);

    //продолжение реализации ...

```

Вы также можете инициализировать строку, используя объект данных, как показано на следующих примерах. Опять же, вы должны указать правильную кодировку.

```

NSString *path = ...;

NSData *data = [NSData dataWithContentsOfFile:path];

// предположим данные в UTF8

NSString *string = [NSString stringWithUTF8String:[data bytes]];

// Если данные в другой кодировке, например ISO-8859-1

NSString *string = [[NSString alloc]
                    initWithData:data encoding: NSISOLatin1StringEncoding];

```

Важно: NSString предоставляет методы (такие, как stringWithContentsOfFile: и initWithContentsOfFileURL:) для чтения из файла или URL, без указания кодировки. Поскольку эти методы не указывают кодировку, вы можете потерять информацию или уничтожить данные, используя их. Вам настоятельно не рекомендуется использовать эти методы, поскольку они являются устаревшими.

Если вы окажетесь с текстом в неизвестной кодировке, то лучше сначала убедиться, что существует механизм для исправления неизбежных ошибок. Например, *Apple Mail* и *Safari* имеют меню с выбором кодировки, *TextEdit* позволяет пользователю переоткрыть файл с явно указанной кодировкой.

Если вы вынуждены угадать кодировку (заметим, что в отсутствие явной информации, это будет предположением):

1. Попробуйте [stringWithContentsOfFile:usedEncoding:error:](#) или [initWithContentsOfFile:usedEncoding:error:](#) (или их URL эквиваленты).

Эти методы пытаются определить кодировку ресурса, и в случае успеха возвращают по ссылке используемую кодировку.

2. Если пункт 1 не удался, попробуйте прочитать ресурс, указав кодировку UTF-8.
3. Если и 2-й пункт неудачен, попробуйте соответствующую унаследованную кодировку.

«Соответствующую» здесь немного зависит от обстоятельств, это может быть строка в C кодировке по умолчанию, это может быть ISO или Windows, Latin 1, или нечто другое, в зависимости от того, что за данные и откуда получены.

4. Наконец, вы можете попробовать [NSAttributedString](#) в методах загрузки из [Application Kit](#) (таких, как [initWithURL:options:documentAttributes:error:](#)).

Эти методы попытаются загрузить текстовые файлы и возвращают используемую кодировку. Они могут быть использованы для более или менее произвольных текстовых документов, и которые стоит рассматривать, если ваше приложение не имеет представления о тексте. Они не могут быть использованы в случаях для Foundation-уровня инструментов и документов, которые не являются естественно-языковыми текстами.

По сравнению с чтением данных из файла или URL-адреса, запись в `NSString` просто обеспечивает два удобных метода, `writeToFile:atomically:encoding:error:` и `writeToURL:atomically:encoding:error:`. Вы должны указать кодировку, которая должна использоваться, и выбрать, следует ли писать ресурс атомарно или нет. Если вы не хотите писать атомарно, строка записывается непосредственно на путь, который вы укажете, в противном случае она будет записана во вспомогательный файл, а затем вспомогательный файл переименовывается в путь. Этот параметр гарантирует, что файл, если он вообще существует, не будет поврежден, даже если система совершит аварию во время записи. При записи в URL-адрес, опция атомарность игнорируется, если пункт назначения не того типа, который может быть доступен атомарно.

```
NSString *path = ...;

NSString *string = ...;

NSError *error;

BOOL ok = [string writeToFile:path atomically:YES
                    encoding:NSUTF8StringEncoding error:&error];

if (!ok) {
    // обработка ошибки...

    NSLog(@"Error writing file at %@\n%@",
          path, [error localizedFailureReason]);

    // продолжение реализации ...
}
```

В данной таблице представлены наиболее распространенные способы чтения и записи строковых объектов в и из файлов и URL-адресов:

Таблица 4 Способы чтения и записи строковых объектов

Источник	Метод читающий	Метод записывающий
Файл	<u><code>stringWithContentsOfFile:encoding:error:</code></u> <u><code>stringWithContentsOfFile:usedEncoding:error:</code></u>	<u><code>writeToFile:atomically:encoding:error:</code></u>
URL	<u><code>stringWithContentsOfURL:encoding:error:</code></u> <u><code>stringWithContentsOfURL:usedEncoding:error:</code></u>	<u><code>writeToURL:atomically:encoding:error:</code></u>

9.5 Поиск, сравнение и сортировка строк

Строковые классы предоставляют методы поиска символов и подстрок в строках и сравнения одной строки с другой. Эти методы соответствуют стандарту Юникода, для определения того, что две последовательности символов эквивалентны. Строковые классы предоставляют методы сравнения, которые обрабатывают последовательности символов должным образом, хотя у вас есть возможность указать буквальный поиск, когда эффективность является важнее и вы можете гарантировать некоторую каноническую форму для составления последовательности символов [3].

Каждый метод поиска и сравнения существует в нескольких вариантах. Простейший вариант это поиск по всей строке или сравнение всей строки. Другие варианты позволяют изменять способ сравнения последовательностей символов, указывая определенный диапазон символов в строке для поиска или сравнения, вы также можете найти и сравнить строки в контексте данной локали.

Основные методы поиска и сравнения приведены в таблице 5.

Таблица 5 Методы поиска и сравнения строковых объектов

Метод поиска	Метод сравнения
<u>rangeOfString:</u>	<u>compare:</u>
<u>rangeOfString:options:</u>	<u>compare:options:</u>
<u>rangeOfString:options:range:</u>	<u>compare:options:range:</u>
<u>rangeOfString:options:range:locale:</u>	<u>compare:options:range:locale:</u>
<u>rangeOfCharacterFromSet:</u>	
<u>rangeOfCharacterFromSet:options:</u>	
<u>rangeOfCharacterFromSet:options:range:</u>	

Вы можете использовать методы `rangeOfString:...` для поиска подстроки в объекте приемнике. `rangeOfCharacterFromSet:...` методы поиска для отдельных символов из поставляемого набора символов.

Подстроки будут найдены только если полностью, содержатся в указанном диапазоне. Если задать диапазон для метода поиска или сравнения и не требовать `NSLiteralSearch` (см. ниже), диапазон не должен нарушать границы последовательности символов на любом конце, если это произойдет, вы можете получить неправильный результат. (См. описание метода [rangeOfComposedCharacterSequenceAtIndex:](#) для кода, который регулирует диапазон на границах последовательности символов)

Вы также можете сканировать строковый объект на числовые и строковые значения, используя экземпляр `NSScanner`. Оба `NSString` и `NSScanner` класса кластеров используют `NSCharacterSet` класс кластер для операций поиска.

Если вы просто хотите определить, соответствует ли строка содержимому заданного шаблона, вы можете использовать предикат:

```
BOOL match = [myPredicate evaluateWithObject:myString];
```

Методы `compare:...` возвращают лексический порядок объекта приемника и поставляемой строки. Некоторые другие методы позволяют определить, являются ли две строки равными, или же один префикс или суффикс другой, но у них нет вариантов, которые позволяют задавать параметры или диапазоны поиска.

Простейший метод для сравнения строк, это compare:-это то же самое как вызов compare:options:range: без опций и в полном объеме объекта приемника в качестве диапазона. Если вы хотите задать параметры сравнения (NSCaseInsensitiveSearch, NSLiteralSearch или NSNumericSearch) можно использовать compare:options: а если вы хотите указать язык можно использовать compare:options:range:locale:. NSString также предоставляет различные удобные методы, чтобы позволить вам выполнять общие сравнения, не нужно указывать диапазоны и варианты напрямую, например caseInsensitiveCompare: и localizedCompare:.

Важно: Для показа пользователю отсортированных списков, вы всегда должны использовать локализованные сравнения. Так обычно вместо того, чтобы вызывать compare: или caseInsensitiveCompare: вы должны использовать localizedCompare: или localizedCaseInsensitiveCompare:.

Если вы хотите сравнить строки в том же порядке, как они представлены в Finder, вы должны использовать compare:options:range:locale: пользователя и следующие опции: NSCaseInsensitiveSearch, NSNumericSearch, NSWidthInsensitiveSearch и NSForcedOrderingSearch.

Несколько методов поиска и сравнения принимают "опции" в качестве аргумента. Это битовая маска, которая добавляет дополнительные ограничения на операцию. Для создания маски, комбинируют следующие опции, представленные в таблице 6 (не все опции доступны для каждого метода).

Таблица 6 Опции

Опция поиска	Эффект
NSCaseInsensitiveSearch	Игнорирует различия регистра между символами
NSLiteralSearch	Выполняет сравнение байт-за-байтом. Различные буквально последовательности (например, последовательности символов), которые могли бы считаться эквивалентными считаются не соответствующими. Использование этой опции может резко ускорить некоторые операции.
NSBackwardsSearch	Выполняет поиск с конца к началу диапазона.
NSAnchoredSearch	Выполняет поиск символов только в начале или в конце диапазона. Если нет совпадений в начале или конце, то ничего не найдено, даже если соответствующие последовательности символов присутствуют где-нибудь в строке.
NSNumericSearch	При использовании с методом <u>compare:options:</u> , группы чисел рассматривается как числовое значение для целей сравнения. Например, Filename9.txt < Filename20.txt < Filename100.txt.

Поиск и сравнение в настоящее время осуществляются с опцией NSLiteralSearch по умолчанию.

Примеры.

Поиск префиксов и суффиксов без учета регистра.

NSString обеспечивает методы hasPrefix: и hasSuffix: которые вы можете использовать, чтобы найти точное совпадение префикса или суффикса. Следующий

пример иллюстрирует, как можно использовать *rangeOfString:options:* с сочетанием параметров для выполнения поиска без учета регистра.

```
NSString *searchString = @"age";

NSString *beginsTest = @"Agencies";

NSRange prefixRange = [beginsTest rangeOfString:searchString
                        options:(NSAnchoredSearch | NSCaseInsensitiveSearch)];

// prefixRange = {0, 3}

NSString *endsTest = @"BRICOLAGE";

NSRange suffixRange = [endsTest rangeOfString:searchString
                      options:(NSAnchoredSearch | NSCaseInsensitiveSearch |
                               NSBackwardsSearch)];

// suffixRange = {6, 3}
```

Сравнение строк

Следующие примеры иллюстрируют применение различных методов сравнения строк и связанные с ними опции. Первый показывает простейший метод сравнения.

```
NSString *string1 = @"string1";
NSString *string2 = @"string2";
NSComparisonResult result;
result = [string1 compare:string2];
// result = -1 (NSOrderedAscending)
```

Вы можете сравнивать строки численно с использованием опции *NSNumericSearch*:

```
NSString *string10 = @"string10";

NSString *string2 = @"string2";

NSComparisonResult result;

result = [string10 compare:string2];

// result = -1 (NSOrderedAscending)

result = [string10 compare:string2 options:NSNumericSearch];

// result = 1 (NSOrderedDescending)
```

Вы можете использовать удобные методы (*caseInsensitiveCompare:* и *localizedCaseInsensitiveCompare:*) для выполнения сравнения без учета регистра:

```
NSString *string_a = @"Aardvark";
NSString *string_A = @"AARDVARK";

result = [string_a compare:string_A];
// result = 1 (NSOrderedDescending)

result = [string_a caseInsensitiveCompare:string_A];
// result = 0 (NSOrderedSame)
// эквивалентно [string_a compare:string_A options:NSCaseInsensitiveSearch]
```

Сортировка строк, как в *Finder*.

Следующий пример показывает, как можно сравнивать строки, чтобы выстроить их в так же, как они представлены в *Finder*. Во-первых, необходимо определить функцию сортировки, которая включает соответствующие варианты сравнения (для эффективности, проходит по региональным настройкам пользователя в зависимости от контекста).

```
int finderSortWithLocale(id string1, id string2, void *locale)
{
    static NSStringCompareOptions comparisonOptions =
        NSCaseInsensitiveSearch | NSNumericSearch |
        NSWidthInsensitiveSearch | NSForcedOrderingSearch;

    NSRange string1Range = NSMakeRange(0, [string1 length]);

    return [string1 compare:string2
            options:comparisonOptions
            range:string1Range
            locale:(NSLocale *)locale];
}
```

Вы передаете функции в качестве параметра `sortedArrayUsingFunction:context:` текущую локаль пользователя в зависимости от контекста:

```
NSArray *stringsArray =
    [NSArray arrayWithObjects:
        @"string 1",
        @"String 21",
        @"string 12",
        @"String 11",
        @"String 02", nil];

NSArray *sortedArray = [stringsArray
    sortedArrayUsingFunction:finderSortWithLocale
    context:[NSLocale currentLocale]];

// sortedArray содержит: { "string 1", "String 02", "String 11",
    "string 12", "String 21" }
```

9.6 Параграфы (абзацы) и разрывы строк

Есть несколько способов, в которых строки или абзацы, могут быть представлены. Исторически были использованы `\n`, `\r`, и `\r\n`. Юникод определяет однозначный разделитель пункта, `U+2029` (для которого Сосоа обеспечивает постоянную `NSParagraphSeparatorCharacter`), и однозначную разделительную линию, `U+2028` (для которой Сосоа обеспечивает постоянную `NSLineSeparatorCharacter`) [3].

В системе текста Сосоа `NSParagraphSeparatorCharacter` обрабатывается последовательно, как разрыв абзаца, а `NSLineSeparatorCharacter` обрабатывается последовательно, как разрыв строки, то есть строки внутри абзаца. Однако в других контекстах, есть несколько вариантов относительно того, как эти символы будут восприняты. Например POSIX уровень программного обеспечения, часто признает только `\n`, как перенос. В некоторых старых Macintosh, программное обеспечение распознает только `\r`, а некоторое программное обеспечение для Windows распознает только `\r\n`. Нередко бывает, что нет никакого различия между строками и абзацами.

Какой символ линия или конца абзаца вы должны использовать, зависит от того, как ваши данные могут быть использованы и на каких платформах. Система текста Сосоа признает `\n`, `\r`, или `\r\n` как абзац-эквивалентный `NSParagraphSeparatorCharacter`. Когда Сосоа вставляет абзац, например, с `insertNewline:` она использует `\n`. Обычно `NSLineSeparatorCharacter` используется только для прерывания строк и не прерывают абзац, например, в `insertLineBreak:`, или для представления HTML элементов `<br>`.

Если ваши переносы предназначены для строк и не прерывают пункты, то вы должны как правило, использовать `NSLineSeparatorCharacter`. В противном случае, вы можете использовать `\n`, `\r`, или `\r\n` в зависимости от того, что другое программное обеспечение сможет воспринимать для оформления текста. По умолчанию выбор для Cocoa, как правило, `\n`.

Общий подход к разделению строки "в соответствии с параграфами":

```
NSArray *arr = [myString componentsSeparatedByString:@"\n"];
```

Данный вариант, игнорирует тот факт, что существует множество других способов, в которых абзацы или строки может быть перенесены `-\r`, `\r\n`, или Unicode сепаратором. Вместо этого можно использовать методы, такие как `lineRangeForRange:` или `getParagraphStart:end:contentsEnd:forRange:` -чтобы учесть разнообразие возможных переносов, как показано в следующем примере.

```
NSString *string = /* принимает значение какого либо текста */;
unsigned length = [string length];
unsigned paraStart = 0, paraEnd = 0, contentsEnd = 0;
NSMutableArray *array = [NSMutableArray array];
NSRange currentRange;
while (paraEnd < length) {
    [string getParagraphStart:&paraStart end:&paraEnd
     contentsEnd:&contentsEnd forRange:NSMakeRange(paraEnd, 0)];
    currentRange = NSMakeRange(paraStart, contentsEnd - paraStart);
    [array addObject:[string substringWithRange:currentRange]];
}
```

9.7 Символы и кластеры

Как правило, когда думают о строке то представляют себе последовательность символов, но при работе с `NSString` объектом, или с **Unicode** строкой в целом, в большинстве случаев лучше иметь дело с подстрокой, а не с отдельными символами. Причина этого в том, что пользователь воспринимает как символ в тексте то, что во многих случаях может быть представлено несколькими символами в строке. `NSString` имеет большой перечень методов для правильной обработки **Unicode** строк, которые в целом делают соответствие Unicode легким, но есть некоторые меры предосторожности, которые следует соблюдать [3].

`NSString` объекты концептуально **UTF-16** с прямым порядком байтов. Это не говорит об их внутреннем механизме хранения, но это означает, что `NSString` выражается через **UTF-16** единицы длины, символов, индексов и диапазонов, и что термин "символ" в названиях `NSString` методов относится к 16 -битной платформе с прямым порядком байт **UTF-16** символов. Это общее соглашение для строковых объектов. В большинстве случаев, клиенты не должны быть слишком заинтересованные в этом, до тех пор, пока вы имеете дело с подстроками и точная интерпретация диапазона индексов не обязательно значима.

Подавляющее большинство **Юникода** используется для записи живых языков, представленных одним **UTF-16** символом. Однако некоторые менее распространенные языки представлены в кодировке **UTF-16** с помощью суррогатных пар. Суррогатная пара представляет собой последовательность из двух **UTF-16** символов, взятых из конкретного зарезервированного диапазона, которые в совокупности представляют собой единую точку кода Юникода. `CFString` имеет функции для преобразования между суррогатной парой и **UTF-32** представлением соответствующей Юникоду кодовой точкой. При работе с объектами `NSString`, есть одно ограничение, состоящее в том, что границами подстрок, не следует разделять две половины суррогатной пары. Как правило, это решается автоматически для диапазонов возвращаемых из большинства методов Cocoa, но если вы

создаете диапазоны подстрок сами, вы должны иметь это в виду. Тем не менее, это не единственное ограничение, которое Вы должны рассмотреть.

Во многих системах письменности, один символ может состоять из базового символа плюс иметь ударение или другую декорацию. Число возможных букв и их декораций исключено из **Unicode**, представляющего каждую комбинацию, как единую точку кода, так что в целом такие комбинации представлены базовым символом, за которым следует один или более присоединяемых знаков. Из соображений совместимости, **Unicode** имеет единую точку кода для ряда наиболее распространенных комбинаций, они называются *precomposed forms* и преобразования нормализации **Unicode** могут быть использованы для преобразования между *precomposed* и разложенным представлением. Однако, даже если строка полностью *precomposed*, существует много комбинаций, которые должны быть представлены с использованием базового знака и сочетанием знаков. Для большинства текстов, диапазоны подстрок должны быть устроены так, чтобы их границы не разделяли базовый символ от связанного с ним сочетания знаков.

Кроме того, есть письменности, в которых символы представляют собой комбинацию частей, которые являются более сложными, чем знак ударения. В корейском, например, отдельный слог Hangul может состоять из двух или трех подразделов известный как Jamo. В индийских и письменностях распространенных по всей Южной и Юго-Восточной Азии, один письменный знак часто представляет собой сочетания согласных, гласных и знаков, такие как *viramas* и **Unicode** представления для этих систем письменности часто используют элементы кода для этих отдельных частей, так, что один символ может состоять из нескольких точек кода. Для большинства текстов, диапазоны подстрок также должны быть устроены так, что их границы не разделяют Jamo в одном слоге Hangul или компоненты кластера согласных индийских.

В общем, это комбинация-суррогатных пар, базовый символ плюс сочетание знаков, Hangul Jamo, и индийских согласных, называются графемой кластеров. Для того, чтобы принять их во внимание, вы можете использовать методы `NSString rangeOfComposedCharacterSequencesForRange:` или `rangeOfComposedCharacterSequenceAtIndex:`, или `CFStringGetRangeOfComposedCharactersAtIndex`. Они могут быть использованы для корректировки индексов строки или диапазонов подстрок, так что они падают на границы графем кластера, с учетом всех ограничений, упомянутых выше. Эти методы должны быть выбраны по умолчанию для программного определения границ воспринимаемых пользователем символов.

В некоторых случаях, алгоритмы **Unicode** имеют дело с несколькими символами таким образом, что выходят за рамки даже графема границы кластера. В **Unicode** алгоритмы регистра могут преобразовать один символ в несколько символов при переходе от нижнего регистра в верхний, например, стандартный эквивалент заглавного немецкого символа "ß" является двухбуквенная последовательность "SS". Локализованные алгоритмы сортировки на многих языках рассматривают несколько символов последовательности, как отдельные единицы, например, последовательность "ch" считается одной буквой при сортировке в некоторых европейских языках. Для того, чтобы иметь дело с должным образом обработанным результатом, в подобных случаях важно использовать стандартные методы `NSString` для таких операций, как регистр, сортировка и поиск, и использовать их ко всей строке, к которой они должны применяться. Используйте `NSString` методы, такие как `lowercaseString`, `uppercaseString`, `capitalizedString`, `compare:` и их варианты, `rangeOfString:` и его варианты, и `rangeOfCharacterFromSet:` и его варианты, или их эквивалентов `CFString`. Все это учитывает сложность обработки **Unicode** строк, и поиск и сортировка методами, в частности, есть много вариантов для контроля типов эквивалентности их распознавания.

В некоторых менее распространенных случаях может возникнуть необходимость адаптировать определение графа кластеры особой необходимости. Вопросы, связанные с определением границ и пошив одежды графа кластера подробно описан в стандарте **Unicode**, где дается ряд примеров и некоторых алгоритмов. Стандарт **Unicode** в целом является лучшим источником информации о Unicode алгоритмах и соображениях, связанных с обработкой **Unicode** строк.

9.8 Наборы символов

`NSCharacterSet` объект представляет собой набор символов Юникода. `NSString` и `NSScanner` объекты используют `NSCharacterSet` объект к группе символов для операций поиска, так что они могут найти любой из определенного в наборе символов во время поиска [3].

Объект набора символов представляет собой набор символов Юникода. Наборы символов представлены экземплярами класса кластера. Два кластера классов, `NSCharacterSet` и `NSMutableCharacterSet` объявить программный интерфейс для неизменных и изменяемых наборов символов, соответственно. Набор неизменных символов определяется при его создании и в дальнейшем не может быть изменен. Изменяемый набор символов может быть изменен после его создания.

Объект набора символов не выполняет никаких задач, а просто содержит набор значений символов для ограничения операций над строками. `NSString` и `NSScanner` классы определяют методы, которые принимают `NSCharacterSet` объекты в качестве аргументов, чтобы найти любой из нескольких символов. Например, в этом фрагменте кода находит диапазон первой заглавной буквы в `myString`.

```
NSString *myString = @"some text in an NSString...";
NSCharacterSet *characterSet = [NSCharacterSet uppercaseLetterCharacterSet];
NSRange letterRange;

letterRange = [myString rangeOfCharacterFromSet:characterSet];
```

После этого фрагмента вызывается, `letterRange.location` для определения индекса первого заглавного символа, в данном примере "N" в "NSString" после вызова `rangeOfCharacterFromSet:`. Если первая буква "s" строки была бы "S", то `letterRange.location` вернуло бы 0.

`NSCharacterSet` определяет методы класса, которые возвращают часто используемые наборы символов, такие как буквы (верхнего или нижнего регистра), десятичные цифры, пробелы и так далее. Эти "стандартные" наборы символов всегда неизменны, даже если они созданы, отправкой сообщения для `NSMutableCharacterSet`.

Вы можете использовать стандартный набор символов в качестве отправной точки для создания пользовательского набора, делая изменяемой копию, и изменяя ее. (Вы также можете начать с нуля, создавая изменяемый набор символов с `alloc` и `init` и добавляя к нему символы.) Например, этот фрагмент создает набор символов содержащий буквы, цифры и основные знаки препинания:

```
NSMutableCharacterSet *workingSet;
NSCharacterSet *finalCharSet;

workingSet = [[NSCharacterSet alphanumericCharacterSet] mutableCopy];
[workingSet addCharactersInString:@";,:. "];
finalCharSet = [workingSet copy];
[workingSet release];
```

Для определения пользовательского набора символов с использованием Юникода, используйте код, подобный следующему фрагменту (который создает набор символов, включая прогон страницы и символ разделителя строк):

```
UniChar chars[] = {0x000C, 0x2028};
```

```

NSString *string = [[NSString alloc] initWithCharacters:chars
                    length:sizeof(chars) / sizeof(UniChar)];
NSCharacterSet *chset = [NSCharacterSet
                        characterSetWithCharactersInString:string];
[string release];

```

Поскольку наборы символов, часто принимают участие в выполнении важных операций, вы должны быть осведомлены об аспектах их использования, которые могут повлиять на производительность приложения. Изменяемые наборы символов, как правило, гораздо дороже, чем наборы неизменного характера. Они потребляют больше памяти и являются дорогостоящими, для инвертирования (операция часто выполняется при сканировании строки). Из-за этого вы должны выполнить следующие рекомендации:

1. Создавайте как можно меньше изменяемых наборов символов, насколько это возможно.
2. Кэшируйте наборы символов (в глобальных словарях, к примеру), а не постоянно воссоздайте их.
3. При создании пользовательского набора, который не нужно менять после создания, сделайте неизменяемую копию окончательного набора символов для фактического использования, и утилизируйте рабочий изменяемый набор символов. В качестве альтернативы создайте файл с набором символов, как описано ниже и сохраните его в основной пакет приложения.
4. Аналогично, следует избегать архивации объектов наборов символов, вместо этого храните их в файлах наборов символов. Архивирование может привести к дублированию набора символов в разных архивных файлах, что в результате приведет к впустую использованию дискового пространства и дублированию в памяти каждого отдельно прочитанного архива.

Если ваше приложение часто использует пользовательский набор символов, вы должны сохранить его определение в файле ресурсов и загружать, вместо явного добавления отдельных символов каждый раз как вам нужно создавать набор. Вы можете сохранить набор символов, получив его растровые представления (объект *NSData*) и сохранить этот объект в файле:

```

NSString *filename;    /*Предположим, что уже существует. */
NSString *absolutePath;
NSData *charSetRep;
BOOL result;
absolutePath = [filename stringByStandardizingPath];
charSetRep = [finalCharSet bitmapRepresentation];
result = [charSetRep writeToFile:absolutePath atomically:YES];

```

По соглашению, имена файлов наборов символов используют расширение *.bitmap*. Если вы собираетесь использовать Ваш набор символов другими файлами, вы должны следовать этому соглашению. Для чтения файлов набор символов с *.bitmap* расширением, просто используйте метод *characterSetWithContentsOfFile:*.

Стандартные наборы символов, таких как те, что возвращает *letterCharacterSet*, формально определяются с точки зрения нормативных и информативных категорий установленных стандартом Юникода, такими как буквы верхнего регистра, и так далее. Формальное определение стандартного набора символов, в большинстве случаев дается как одна или несколько категорий, определенных в стандарте. Например, набор, возвращаемый *lowercaseLetterCharacterSet* включает в себя все символы в категории нижнего регистра, в то время как набор, возвращаемый *letterCharacterSet* включает в себя символы во всех категориях.

Обратите внимание, что определение категорий сами по себе могут меняться с новыми версиями Unicode стандарта. Вы можете скачать файлы, которые определяют категории с <http://www.unicode.org/>.

9.9 Сканеры

Объект `NSString` сканирует символы объекта `NSString`, как правило, интерпретируя символы и преобразуя их в числа и строковые значения. Вы назначаете строку сканера при создании, и сканер продвигается по символам этой строки от начала до конца, когда вы запрашиваете элементы [3].

`NSString` класс кластер с одним открытым классом, `NSString`. Как правило, вы реализуете объект сканер, вызвав метод класса `scannerWithString:` или `localizedScannerWithString:`. Любой метод возвращает объект сканера инициализированный строкой, которую вы передаете ему. Вновь созданный сканер начинается в начале своей строки. Вы сканируете компоненты с помощью `scan...` методов, таких как `scanInt:`, `scanDouble:` и `scanString:intoString:`. При сканировании нескольких строк, как правило, создается цикл `while`, который продолжается, пока сканер не достигнет конца строки, как показано в следующем фрагменте кода:

```
float aFloat;

NSString *theScanner = [NSString scannerWithString:aString];

while ([theScanner isAtEnd] == NO) {
    [theScanner scanFloat:&aFloat];
    // продолжение реализации...
}
```

Вы можете настроить сканер, чтобы он рассматривал или игнорировал регистр используя метод `setCaseSensitive:`. По умолчанию сканер не учитывает регистр.

Операции сканирования начинаются с проверки местонахождения сразу за последним символом в отсканированном участке (если таковой имеется). Например, после сканирования чисел в строке "137 small cases of bananas", расположение сканера будет 3, что указывает на пробел сразу после числа. Часто вам нужно переместить место сканирования, чтобы пропустить символы, в которых вы не заинтересованы. Вы можете изменить начальную точку сканирования с помощью метода `setScanLocation:`, чтобы пропустить определенное количество символов (вы также можете использовать этот метод для повторного сканирования части строки после ошибки). Обычно, однако, вы либо хотите пропустить символы из определенного набора символов, прошлое сканирование определенной строки, или сканировать до определенного места.

Вы можете настроить сканер, чтобы пропустить набор символов с помощью метода `setCharactersToBeSkipped:`. Сканер игнорирует символы, которые должны быть пропущены в начале какой-либо операции сканирования. Как только он находит сканируемый символ, он включает в себя все символы, соответствующие запросу. Сканеры пропускают пробелы и символы новой строки по умолчанию. Обратите внимание, что регистр всегда рассматривается с символами, которые будут пропущены. Чтобы пропустить все английские гласные, вы должны установить символы, которые будут пропущены в строке: "AEIOUaeiou".

Если вы хотите прочитать содержимое текущего местоположения до определенной строки, вы можете использовать `scanUpToString:intoString:` (Вы можете передать `NULL` в качестве второго аргумента, если вы просто хотите пропустить промежуточные символы). Например, в следующей строке:

```
137 small cases of bananas
```

Вы можете найти тип контейнера и количество контейнеров, используя `scanUpToString:intoString:` как показано в следующем примере.

```
NSString *bananas = @"137 small cases of bananas";
NSString *separatorString = @" of";
```

```

NSScanner *aScanner = [NSScanner scannerWithString:bananas];

NSInteger anInteger;
[aScanner scanInteger:&anInteger];
NSString *container;
[aScanner scanUpToString:separatorString intoString:&container];

```

Важно отметить, что строка поиска (`separatorString`) содержит " of". По умолчанию сканер игнорирует пробелы, поэтому пробел после целого игнорируется. Как только сканер начинает накапливать символы, все символы добавляются в выходную строку, пока строка поиска не будет достигнута. Таким образом, если строка поиска "of" (без первого пробела), первое значение `container` "small cases " (включает в себя последующие пробелы), если строка поиска " of" (с начальным пробелом), то первое значение `container` "small cases" (без следующего пробела).

После сканирования до заданной строки, расположение сканирования установлено в начале этой строки. Если вы хотите проверить предыдущее содержание этой строки, вы должны выполнить первое сканирование в строке, которую Вы просматривали до этого. Следующий фрагмент кода показывает, как можно пропустить мимо строку поиска из предыдущего примера, и определить тип продукта в контейнере. Обратите внимание на использование `substringFromIndex`: в действие сканирования до конца строки.

```

[aScanner scanString:separatorString intoString:NULL];
NSString *product;
product = [[aScanner string] substringFromIndex:[aScanner scanLocation]];
// Можно также использовать:
// product = [bananas substringFromIndex:[aScanner scanLocation]];

```

Пример:

Предположим, у вас есть строка, содержащая подстроки, такие как:

- Product: Acme Potato Peeler; Cost: 0.98 73
- Product: Chef Pierre Pasta Fork; Cost: 0.75 19
- Product: Chef Pierre Colander; Cost: 1.27 2

Следующий пример использует дублирование операций сканирования для извлечения названия продуктов и расходов (расходы, которые читаются как *float* для простоты), пропуская подстроки "Product:" и "Cost:", а также точки с запятой. Обратите внимание, что сканер пропускает пробелы и переводы строк по умолчанию, цикл не производит специальной обработки для них (в частности, нет необходимости делать дополнительные пробелы обработки для получения окончательного целого числа).

```

NSString *string = @"Product: Acme Potato Peeler; Cost: 0.98 73\n\
Product: Chef Pierre Pasta Fork; Cost: 0.75 19\n\
Product: Chef Pierre Colander; Cost: 1.27 2\n";

NSCharacterSet *semicolonSet;
NSScanner *theScanner;

NSString *PRODUCT = @"Product:";
NSString *COST = @"Cost:";

NSString *productName;
float productCost;
NSInteger productSold;

semicolonSet = [NSCharacterSet characterSetWithCharactersInString:@";"];
theScanner = [NSScanner scannerWithString:string];

while ([theScanner isAtEnd] == NO)
{
    if ([theScanner scanString:PRODUCT intoString:NULL] &&
        [theScanner scanUpToCharactersFromSet:semicolonSet

```

```

        intoString:&productName] &&
        [theScanner scanString:@";" intoString:NULL] &&
        [theScanner scanString:COST intoString:NULL] &&
        [theScanner scanFloat:&productCost] &&
        [theScanner scanInteger:&productSold])
    {
        NSLog(@"Sales of %@: $%1.2f", productName, productCost *
productSold);
    }
}

```

NSScanner использует только определение локали для десятичного разделителя (выдается ключ NSDecimalSeparator). Вы можете создать сканер с локалью пользователя с помощью `localizedScannerWithString:` или установить локаль явно с помощью `setLocale:`. Если вы используете метод, который не определяет язык, сканер принимает значения локали по умолчанию.

9.10 Строковые представления файловых путей

NSString обеспечивает богатый набор методов для манипулирования строками в качестве путей файловой системы. Вы можете извлечь каталог директории, имя файла и расширение, тильду расширения выражения (например, "~me") или создать для домашнего каталога пользователя, а также очистит пути содержащих символические ссылки, избыточные слешы, и ссылки на "." (текущего каталога) и ".." (родительского каталога) [3].

NSString представляет пути с "/" в качестве разделителя пути и "." как сепаратор расширения. Методы, которые принимают строки в качестве аргументов пути преобразовывают эти общие представления к надлежащей системе конкретной формы, по мере необходимости. В системах с неявным корневым каталогом, абсолютные пути начинаются с разделителя пути или с тильды ("~/..." или "~user/..."). Если устройство должно быть указано, вы можете сделать это самостоятельно, внедрением системы зависимостей или разрешить строковому объекту, добавить устройство по умолчанию.

Вы можете создать стандартизированные представления пути, используя *stringByStandardizingPath*. Он выполняет ряд задач, в том числе:

1. Расширение исходного выражения тильды;
2. Сокращение пустых компонентов и ссылок на текущий каталог и ("/" и "/.") с одним сепаратором пути;
3. В абсолютных путях, разрешение ссылки на родительский каталог ("..") к реальному каталогу родителя;

Пример.

```

NSString *path;
NSString *standardizedPath;

path = @"/usr/bin/./grep";
standardizedPath = [path stringByStandardizingPath];
// standardizedPath: /usr/bin/grep

path = @"~me";

standardizedPath = [path stringByStandardizingPath];
// standardizedPath (предположение общепринятой
схеме именования): /Users/Me

path = @"/usr/include/objc/..";

standardizedPath = [path stringByStandardizingPath];
// standardizedPath: /usr/include

path = @"/private/usr/include";

```

```
standardizedPath = [path stringByStandardizingPath];
// standardizedPath: /usr/include
```

Следующие примеры показывают, как можно использовать NSString в пути и другие функции, Сосоа, чтобы получить пользовательские директории.

```
//если предположить, что домашние каталоги пользователей хранятся в /Users
```

```
NSString *meHome = [@"~me" stringByExpandingTildeInPath];
// meHome = @"/Users/me"
```

```
NSString *mePublic = [@"~me/Public" stringByExpandingTildeInPath];
// mePublic = @"/Users/me/Public"
```

Вы можете найти домашний каталог текущего пользователя и для данного пользователя с NSHomeDirectory и NSHomeDirectoryForUser соответственно:

```
NSString *currentUserHomeDirectory = NSHomeDirectory();
NSString *meHomeDirectory = NSHomeDirectoryForUser(@"me");
```

Обратите внимание, что вы должны как правило, использовать функцию NSSearchPathForDirectoriesInDomains, чтобы найти стандартные каталоги для текущего пользователя. Например, вместо:

```
NSString *documentsDirectory = [NSHomeDirectory()
    stringByAppendingPathComponent:@"Documents"];
```

ВЫ ДОЛЖНЫ ИСПОЛЬЗОВАТЬ:

```
NSString *documentsDirectory;
NSArray *paths = NSSearchPathForDirectoriesInDomains (NSDocumentDirectory,
    NSUserDomainMask, YES);
if ([paths count] > 0) {
    documentsDirectory = [paths objectAtIndex:0];
}
```

NSString обеспечивает богатый набор методов для манипулирования строками в качестве путей файловой системы, например:

<u>pathExtension</u>	Интерпретирует путь объекта-приемника и возвращает расширение, если таковое имеется.
<u>stringByDeletingPathExtension</u>	Возвращает новую строку, удалив расширение (если таковые имеются, и только последнее) от объекта приемника.
<u>stringByDeletingLastPathComponent</u>	Возвращает новую строку, удалив последний компонент пути от объекта-приемника, а также любые окончательные разделители пути.

С помощью этих и связанных с ними методов, *описанных в NSString классе*, вы можете извлечь путь к директории, имя файла и расширение, о чем свидетельствуют следующие примеры.

```
NSString *documentPath = @"~me/Public/Demo/readme.txt";
```

```
NSString *documentDirectory = [documentPath
    stringByDeletingLastPathComponent];
// documentDirectory = @"~me/Public/Demo"
```

```
NSString *documentFilename = [documentPath lastPathComponent];
// documentFilename = @"readme.txt"
```



```
NSString *documentExtension = [documentPath pathExtension];
// documentExtension = @"txt"
```

Вы можете найти возможные расширения имен файлов, используя `completePathIntoString:caseSensitive:matchesIntoArray:filterTypes:`. Например, если каталог `~/Demo`, который содержит следующие файлы:

```
ReadMe.txt readme.html readme.rtf recondite.txt test.txt
NSString *partialPath = @"~/Demo/r";
NSString *longestCompletion;
NSArray *outputArray;
```

```
unsigned allMatches = [partialPath completePathIntoString:&longestCompletion
                      caseSensitive:NO
                      matchesIntoArray:&outputArray
                      filterTypes:NULL];
```

```
// allMatches = 3
// longestCompletion = @"~/Demo/re"
// outputArray = (@"~/Demo/readme.html", "~/Demo/readme.rtf",
//               "~/Demo/recondite.txt")
```

Вы можете найти возможные дополнения для пути `~/Demo/r` с расширениями `“.txt”` или `“.rtf”` как показано ниже:

```
NSArray *filterTypes = [NSArray arrayWithObjects:@"txt", @"rtf", nil];
```

```
unsigned textMatches = [partialPath completePathIntoString:&outputName
                      caseSensitive:NO
                      matchesIntoArray:&outputArray
                      filterTypes:filterTypes];
```

```
// allMatches = 2
// longestCompletion = @"~/Demo/re"
// outputArray = (@"~/Demo/readme.rtf",
//               @"~/Demo/recondite.txt")
```

9.11 Отрисовка строк

Вы можете нарисовать строковые объекты непосредственно в имеющем фокус `NSView` с использованием таких методов, как `drawAtPoint:withAttributes:` (рисует строку с несколькими атрибутами, например, несколько шрифтов текста, вы должны использовать `NSAttributedString` объект).

Простые методы, однако, предназначены для нанесения небольших объемов текста или текста, который рисуется редко различными вспомогательными объектами, которые создают и распоряжаются им каждый раз, когда вы их вызываете. Чтобы рисовать строки неоднократно, более эффективно использовать `NSLayoutManager`.

9.12 Класс NSString

Справка по классу **NSString** представлена в таблице 7 [3].

Таблица 7 Справка

Унаследован от:	<u><i>NSObject</i></u>
Соответствует:	<u><i>NSCopying NSMutableCopying NSObject NSCoding</i></u>
Framework:	/System/Library/Frameworks/Foundation.framework

Доступен:	Mac OS X 10.0 и выше, iOS 2.0 и выше
Задекларирован:	NSString.h NSLinguisticTagger.h NSURL.h NSPathUtilities.h

Класс `NSString` объявляет программный интерфейс для объекта, который управляет неизменной строкой. Неизменной строкой является текстовая строка, которая определяется при ее создании и в дальнейшем не может быть изменена. `NSString` осуществляется для представления массива знаков в кодировке Юникод, другими словами, текстовой строки.

Изменяемым подклассом `NSString` является *`NSMutableString`*.

Класс `NSString` имеет два примитивных метода - *`length`* и *`characterAtIndex:`*, которые обеспечивают основу для всех других методов в его интерфейсе. *`length`* метод возвращает общее количество **Unicode** символов в строке. *`characterAtIndex:`* предоставляет доступ к каждому символу в строке по индексу, со значения индекса начинающегося с 0.

`NSString` объявляет методы для поиска и сравнения строк. Он также объявляет, методы для чтения числовых значений из строк, для различных объединений строк, и для преобразования строки в различных формах (таких, как кодирование и изменение регистра).

Приложение также использует комплект `NSParagraphStyle` и его подкласс `NSMutableParagraphStyle` для инкапсуляции параграфов или исправителя атрибутов, используемых `NSAttributedString` классов.

`NSString` является "мостом" с его *Core Foundation* коллегой *`CFString`*. Это означает, что типы *Core Foundation* являются взаимозаменяемыми в функциях или методах вызовов из объектов *Foundation*. Таким образом, в методе, в котором вы видите параметр **`NSString`** *, вы можете передать *`CFStringRef`*, а в функции, где вы видите *`CFStringRef`* параметр, вы можете передать экземпляр **`NSString`** (Вы бросаете один тип в другой, чтобы подавить предупреждения компилятора). Это также относится и к вашим подклассам **`NSString`**.

`NSString` объекты представляют собой строки символов в *framework*-е. Представление строк в качестве объектов позволяет использовать строки, там где бы вы могли использовать и другие объекты. Они также обеспечивают преимущества инкапсуляции, таким образом, чтобы строковые объекты могли использовать любую кодировку и для эффективности, просто появлялись как массивы символов. Два кластера общих классов, `NSString` и `NSMutableString` объявляют, программный интерфейс для неизменяемых и редактируемых строк, соответственно.

Примечание: неизменной строкой является текстовая строка, которая определяется при ее создании и в дальнейшем не может быть изменена. Неизменная строка реализована в виде массива символов кодировки Юникод (другими словами, текстовая строка). Для создания и управления неизменяемыми строками, используется класс `NSString`. Чтобы создать и управлять строкой, которая может быть изменена после ее создания, используют *`NSMutableString`*.

Объекты, которые вы создаете, используя `NSString` и `NSMutableString` называются строковыми объектами (или, если не приведет к путанице, просто строки). Строковый термин **`C`** ссылается на стандартный тип `char *`. Из-за своей классовой природы кластеров, строковые объекты не являются фактическими экземплярами `NSString` или `NSMutableString` классов, а одним из своих частных подклассов. Хотя класс строкового объекта является частным, его интерфейс является публичным, как это заявлено в

абстрактных суперклассах, **NSString** и **NSMutableString**. Строчный класс принимает протоколы **NSCopying** и **NSMutableCopying**, что делает его удобным для преобразования строки из одного типа в другой.

Строковый объект представляет из себя массив знаков Юникод (**Unicode** является зарегистрированной торговой маркой **Unicode, Inc.**) Вы можете определить, сколько содержится символов в строке объекта с помощью метода *length* и может получить конкретный символ с методом *characterAtIndex:*. Эти два "примитивных" метода обеспечивают базовый доступ в строковому объекту.

При создании объекта **NSString** из строки в кодировке **UTF-16** (или потока байт интерпретируемого как **UTF-16**), если в порядке байт не указано иное, **NSString** предполагает, что **UTF-16** символы выстроены от старшего к младшему, если нет BOM (метка порядка следования байтов), и в этом случае спецификация определяет порядок байтов. При создании объекта **NSString** из массива символов **Unicode**, возвращаемая строка всегда имеет нативный порядок байтов, так как массив всегда содержит символы Юникода в нативном (родном) порядке байтов.

За пределы объектов, изменяемые строковые объекты передаются по ссылке, а неизменяемые передаются как копии.

Возможна реализация подкласса **NSString** (и **NSMutableString**), но для этого требуется предоставление места для строки (что не наследуются подклассами) и реализации двух примитивных методов. Абстрактные классы **NSString** и **NSMutableString** предоставляют открытый интерфейс класса кластера, состоящего в основном из частных, конкретных классов, которые создают и возвращают строковый объект подходящий для данной ситуации. Создание Вашего собственного конкретного подкласса этого кластера накладывает определенные требования.

Убедитесь, что ваши причины для наследования **NSString** являются значимыми. Экземпляры подкласса должны представлять строку, а не что-то еще. Таким образом, только атрибуты подкласса должны иметь доступ к длине символьного буфера, а также доступ к отдельным символам в буфере. Уважительными причинами для принятия подкласса **NSString** является предоставление другого способа хранения (возможно, для лучшей производительности) или осуществления каких-либо аспектов поведения объекта, например таких как управление памятью. Если Ваша цель заключается в добавлении несущественных признаков или метаданных к своим подклассам **NSString**, лучшей альтернативой будет составной объект. Сосоа уже предоставляет пример этого с классом **NSAttributedString**.

В любом подклассе **NSString** необходимо переопределить примитивные методы экземпляра *length* и *characterAtIndex:*. Эти методы должны работать на резервном буфере, который вы предоставляете для символов строки. В качестве резервного буфера можно использовать статический массив, динамически выделенный буфер, стандартный объект **NSString**, или какой-либо другой тип данных или механизм. Вы можете также выбирать, как переопределять, частично или полностью, реализацию любого другого метода **NSString**, чтобы обеспечить альтернативную реализацию. Например, для лучшей производительности рекомендуется, чтобы вы переопределили *getCharacters:range:* и дали ему более быструю реализацию.

Обратите внимание, что вам не следует переопределять *hash*-метод.

Создание и инициализация.

- *string*
- *init*
- *initWithBytes:length:encoding:*
- *initWithBytesNoCopy:length:encoding:freeWhenDone:*
- *initWithCharacters:length:*
- *initWithCharactersNoCopy:length:freeWhenDone:*
- *initWithString:*

- initWithCString:encoding:
- initWithUTF8String:
- initWithFormat:
- initWithFormat:arguments:
- initWithFormat:locale:
- initWithFormat:locale:arguments:
- initWithData:encoding:
- stringWithFormat:
- localizedStringWithFormat:
- stringWithCharacters:length:
- stringWithString:
- stringWithCString:encoding:
- stringWithUTF8String:
- stringWithCString:
- stringWithCString:length:
- initWithCString:
- initWithCString:length:
- initWithCStringNoCopy:length:freeWhenDone:
- **Создание и инициализация из файла**
- stringWithContentsOfFile:encoding:error:
- initWithContentsOfFile:encoding:error:
- stringWithContentsOfFile:usedEncoding:error:
- initWithContentsOfFile:usedEncoding:error:
- stringWithContentsOfFile:
- initWithContentsOfFile:
- **Создание и инициализация из URL**
- stringWithContentsOfURL:encoding:error:
- initWithContentsOfURL:encoding:error:
- stringWithContentsOfURL:usedEncoding:error:
- initWithContentsOfURL:usedEncoding:error:
- stringWithContentsOfURL:
- initWithContentsOfURL:
- **Запись в файл или URL**
- writeToFile:atomically:encoding:error:
- writeToURL:atomically:encoding:error:
- writeToFile:atomically:
- writeToURL:atomically:
- **Получение длины строки**
- length
- lengthOfBytesUsingEncoding:
- maximumLengthOfBytesUsingEncoding:
- **Получение символов и байтов**
- characterAtIndex:
- getCharacters:range:
- getBytes:maxLength:usedLength:encoding:options:range: remainingRange:
- getCharacters:
- **Получение C строки**
- cStringUsingEncoding:
- getCString:maxLength:encoding:
- UTF8String
- cString
- cStringLength

- getCString:
- getCString:maxLength:
- getCString:maxLength:range:remainingRange:
- lossyCString
- **Объединение строк**
- stringByAppendingFormat:
- stringByAppendingString:
- stringByPaddingToLength:withString:startingAtIndex:
- **Разделение строк**
- componentsSeparatedByString:
- componentsSeparatedByCharactersInSet:
- stringByTrimmingCharactersInSet:
- substringFromIndex:
- substringWithRange:
- substringToIndex:
- **Поиск символов и подстрок**
- rangeOfCharacterFromSet:
- rangeOfCharacterFromSet:options:
- rangeOfCharacterFromSet:options:range:
- rangeOfString:
- rangeOfString:options:
- rangeOfString:options:range:
- rangeOfString:options:range:locale:
- enumerateLinesUsingBlock:
- enumerateSubstringsInRange:options:usingBlock:
- **Замена подстрок**
- stringByReplacingOccurrencesOfString:withString:
- stringByReplacingOccurrencesOfString:withString:options:range:
- stringByReplacingCharactersInRange:withString:
- **Определение диапазонов строк и абзацев**
- getLineStart:end:contentsEnd:forRange:
- lineRangeForRange:
- getParagraphStart:end:contentsEnd:forRange:
- paragraphRangeForRange:
- **Определение составленной последовательности символов**
- rangeOfComposedCharacterSequenceAtIndex:
- rangeOfComposedCharacterSequencesForRange:
- **Преобразование содержимого строки в список свойств**
- propertyList
- propertyListFromStringsFileFormat
- **Идентификация и сравнение строк**
- caseInsensitiveCompare:
- localizedCaseInsensitiveCompare:
- compare:
- localizedCompare:
- compare:options:
- compare:options:range:
- compare:options:range:locale:
- localizedStandardCompare:
- hasPrefix:
- hasSuffix:
- isEqualToString:

- hash
- **Складные Строки**
- stringByFoldingWithOptions:locale:
- **Получение общих префиксов**
- commonPrefixWithString:options:
- **Изменение регистра**
- capitalizedString
- lowercaseString
- uppercaseString
- **Получение строк с нормализацией**
- decomposedStringWithCanonicalMapping
- decomposedStringWithCompatibilityMapping
- precomposedStringWithCanonicalMapping
- precomposedStringWithCompatibilityMapping
- **Получение числовых значений**
- doubleValue
- floatValue
- intValue
- integerValue
- longLongValue
- boolValue
- **Работа с кодировками**
- availableStringEncodings
- defaultCStringEncoding
- localizedNameOfStringEncoding:
- canBeConvertedToEncoding:
- dataUsingEncoding:
- dataUsingEncoding:allowLossyConversion:
- description
- fastestEncoding
- smallestEncoding
- **Работа с файловыми путями**
- pathWithComponents:
- pathComponents
- completePathIntoString:caseSensitive:matchesIntoArray:filterTypes:
- fileSystemRepresentation
- getFileSystemRepresentation:maxLength:
- isAbsolutePath
- lastPathComponent
- pathExtension
- stringByAbbreviatingWithTildeInPath
- stringByAppendingPathComponent
- stringByAppendingPathExtension:
- stringByDeletingLastPathComponent
- stringByDeletingPathExtension
- stringByExpandingTildeInPath
- stringByResolvingSymlinksInPath
- stringByStandardizingPath
- stringsByAppendingPaths:
- **Работа с URL-ами**
- stringByAddingPercentEscapesUsingEncoding:
- stringByReplacingPercentEscapesUsingEncoding:

Лингвистическое связывание и анализирование

- [enumerateLinguisticTagsInRange:scheme:options:orthography: usingBlock:](#)
- [linguisticTagsInRange:scheme:options:orthography:tokenRanges:](#)

Приложение

- [unichar](#)
- [NSMaximumStringLength](#)
- [NSStringCompareOptions](#)
- [Опции поиска и сравнения](#)
- [NSStringEncodingConversionOptions](#)
- [Параметры преобразования кодировки строки.](#)
- [NSString Имена обработки исключений](#)
- [NSStringEncoding](#)
- [Строковые кодировки](#)
- [Параметры строкового перечисления](#)

9.13 Класс NSMutableString

Справка по классу **NSString** представлена в таблице 8 [3].

Таблица 8 Справка

Унаследован от:	<u>NSObject</u>
Соответствует:	<u>NSCopying NSMutableCopying NSObject NSCoding</u>
Framework:	/System/Library/Frameworks/Foundation.framework
Доступен:	Mac OS X 10.0 и выше, iOS 2.0 и выше
Задекларирован:	NSString.h

NSMutableString класс объявляет программный интерфейс к объекту, который управляет изменяемыми строками, то есть строками, содержимое которых можно редактировать, которые концептуально представляет собой массив символов Юникода. Для построения и управления используются те же методы как и для неизменных строк, которые не могут быть изменены после того, как они были созданы.

В NSMutableString класс добавляется один примитивный метод-`replaceCharactersInRange:withString:` к основным методам обработки строк, унаследованных от NSString. Все остальные методы, которые изменяют строки работают с помощью этого метода. Например, `insertString:atIndex:` просто заменяет символы в диапазоне от 0 длины, в то время как `deleteCharactersInRange:` заменяет символы в заданном диапазоне, без символов.

Создание и инициализация

- [stringWithCapacity:](#)
- [initWithCapacity:](#)

Изменение строк

- [appendFormat:](#)
- [appendString:](#)
- [deleteCharactersInRange:](#)
- [replaceCharactersInRange:withString:](#)
- [replaceOccurrencesOfString:withString:options:range:](#)
- [setString:](#)

10 Коллекции

10.1 Характеристики коллекций в Cocoa

В **Cocoa** и **Cocoa Touch**, коллекции **Foundation framework** используются для хранения и управления группами объектов. Их основная задача состоит в сохранении объектов в форме: массива, словаря или набора [3].

Эти классы облегчают задачу управления группами объектов. **Foundation** коллекции являются эффективным и широко используется **Mac OS X и IOS**.

Коллекции имеют ряд особенностей. Большинство коллекций содержат только объекты и могут быть изменяемыми (*mutable*) и неизменными.

Все коллекции обладают рядом общих задач, которые включают в себя:

- Перечисление объектов в коллекции
- Определение, находится ли объект в коллекции
- Доступ к отдельным элементам в коллекции

Изменяемые (*mutable*) коллекции позволяют рассмотреть также некоторые дополнительные задачи:

- Добавление объектов в коллекцию
- Удаление объектов из коллекции

Коллекции имеют много общих характеристик, но в тоже время есть и важные различия. В результате, вы обнаружите, что некоторые коллекции лучше подходят для конкретной задачи, чем другие. Потому что, насколько хорошо осуществляется коллекция зависит от того, как она используется, вы должны выбрать ту коллекцию, которая лучше всего подходит для конкретной задачи.

Массивы (например, *NSArray* и *NSMutableArray*) упорядоченные коллекции, которые позволяют индексировать доступ к их содержимому. Вы можете использовать массив для хранения информации, которая, к примеру, будет представлена в элементе управления *Table View*.

Словари (например *NSDictionary* и *NSMutableDictionary*) являются неупорядоченными наборами, которые позволяют получить доступ к их содержимому с помощью ключа-значения. Они также позволяют производить быстрые операции вставки и удаления. Словари полезны для хранения значений, которые имеют смысл в зависимости от их ключа. Например, у вас может быть словарь с информацией о России, со "столица" в качестве ключа и "Москва", как соответствующее значение.

Наборы (например, *NSSet*, *NSMutableSet* и *NSCountedSet*) являются неупорядоченными коллекциями объектов. Наборы позволяют операции быстрой вставки и удаления. Они также позволяют Вам быстро увидеть, является ли объект членом коллекции. *NSSet* и *NSMutableSet* хранят коллекции различных объектов, в то время как *NSCountedSet* хранит коллекции однотипных объектов. Например, предположим, у вас есть целый ряд городских объектов, и вы хотите, посетить каждый только один раз. Если Вы храните каждый город, который вы посетите в наборе, вы можете быстро и легко посмотреть посетили ли вы его.

Индексные наборы (например, *NSIndexSet* и *NSMutableIndexSet*) являются вспомогательными объектами, которые расширяют возможности массивов. Они позволяют хранить подмножества массива в массиве для хранения индексов, а не путем создания нового массива. Вы можете использовать множество индексов, чтобы позволить пользователю выбрать несколько элементов из списка записей. Например, предположим, у вас есть таблицы, и вы позволяет пользователю выбрать несколько строк. Поскольку строки хранятся в виде массива, можно сохранить выбранные в качестве набора индексов в этом массиве.

Индекс путей хранить размещение информации в более сложной иерархии коллекции, в частности, вложенных массивов. Cocoa содержит класс *NSIndexPath* для этой цели.

Хотя они не являются коллекциями в самом прямом смысле, индекс путей облегчает задачу управления вложенными массивами. Класс `UITableView` широко используется индекс путей для хранения местоположения в таблице.

Если вам нужны коллекции для хранения произвольных указателей или целых чисел, или нужно использовать обнуляемые слабые (weak) ссылки в среде сборки мусора, есть три класса коллекций указателей: `NSPointerArray`, `NSMapTable` и `NSHashTable`. Они похожи на `NSMutableArray`, `NSMutableDictionary` и `NSMutableSet` соответственно. Три класса коллекции указателей предоставляют дополнительные опции для определения, как коллекция управляет его содержанием. Вы можете, например, использовать указатель равенства вместо вызова `isEqual:` при сравнении. В отличие от всех других классов коллекций, `NSPointerArray` имеет право содержать `NULL` указатель.

В дополнение к классам, определяющим поведение, есть некоторые задачи, которые являются общими в аналогичной форме между классами коллекций. Две из этих задач: копирования коллекции и перечисление ее содержимого.

Если вам необходимо создать новую коллекцию с содержанием друг, вы можете выбрать либо неполное или полное копирование. В неполной копии каждый объект сохраняется, когда он будет добавлен в новую коллекцию и будет в общем владении для двух или более коллекций. В полной копии каждому объекту передается сообщение `copyWithZone:`, как только он добавляется в коллекцию, вместо его сохранения.

Если вам необходимо проверить каждый элемент в коллекции на некоторые условия или выполнить некоторые действия для выборочной записи, вы можете использовать один из предлагаемых способов перечисления содержимого коллекции. Два основных способа перечисления, это быстрые и блочные перечисления.

10.2 Массивы: упорядоченные коллекции

Массивы - упорядоченные коллекции любого вида объектов. Например, объекты, содержащиеся в массиве на рисунке 1 может быть любой комбинацией объектов для яблок и груш, если массив изменяемый, Вы можете добавить больше объектов собак. Коллекция не должна быть однородной [3].

Информация по производительности:

- Доступ к элементу массива занимает постоянное время.
- Присоединение и удаление элементов с обоих концов занимает постоянное время.
- Замена элемента занимает постоянное время.
- Вставка элемента в середину массива занимает линейное время.

Объект `NSArray` управляет неизменным массивом, то есть после того как вы создаете массив, вы не можете добавлять, удалять или заменять объекты. Можно, однако, изменить отдельные элементы сами (если они поддерживают изменения). Изменчивость коллекции не влияет на изменчивость объектов в коллекции. Вы должны использовать массив неизменным, если массив редко изменяется или изменяется массово.

Объект `NSMutableArray` управляет изменяемым массивом, который позволяет добавлять и удалять записи в любое время, автоматически выделяя память по мере необходимости. Например, если `NSMutableArray` объект, который содержит только один объект яблоко, вы можете добавить еще одно яблоко или грушу, или любой другой объект. Вы можете также, как и объект `NSArray`, изменить имя яблока и в общем, все, что можно сделать с помощью `NSArray` объекта вы можете сделать с `NSMutableArray` объектом. Вы должны использовать изменяемый массив, если массив изменяется постепенно и очень большой, так как большие коллекции занимают больше времени для инициализации.

Вы можете легко создать экземпляр одного типа массива из другого, используя инициализатор `initWithArray:` или удобным конструктором `arrayWithArray`. Например, если у вас есть экземпляр `NSArray`, `myArray`, вы можете создать изменяемую копию следующим образом:

```
NSMutableArray *myMutableArray = [NSMutableArray arrayWithArray:myArray];
```

В общем Вашему экземпляру массива посылается сообщение `array...` или `NSArray` или `NSMutableArray` класса. `array...` сообщение возвратит массив, содержащий элементы, переданные в качестве аргумента. Когда вы добавляете объект к `NSMutableArray` объект не копируется (если вы не передали `YES` в аргументе метода `initWithArray:copyItems:`). Скорее всего, объект добавляется непосредственно в массив. В среде управления памятью объект получает сообщение `retain`, когда добавляется к массиву. Когда массив освобождается, каждый элемент получает сообщение `release`.

В `NSArray`, два основных метода: `-count` и `objectAtIndex:` -обеспечивает основу для всех других методов в интерфейсе:

`count` возвращает количество элементов в массиве.

`objectAtIndex:` обеспечивает доступ к элементам массива. Отсчет элементов начинается с индекса 0.

Основные методы в `NSMutableArray`, перечисленные ниже, служат основой для его способности добавлять, заменять и удалять элементы:

```
addObject:
insertObject:atIndex:
removeLastObject
removeObjectAtIndex:
replaceObjectAtIndex:withObject:
```

Если Вам не нужен объект для размещения по указанному индексу или удалить из середины коллекции, Вы должны использовать `AddObject:`, `removeLastObject` методы, потому что добавлять и удалять в конец массива быстрее, чем в середину.

другие методы в `NSMutableArray` обеспечивают удобный способ вставки объекта в слот в массиве и удаление объекта на основе его идентичности или позиции в массиве, как показано ниже:

```
NSMutableArray *array = [NSMutableArray array];
[array addObject:[NSColor blackColor]];
[array insertObject:[NSColor redColor] atIndex:0];
[array insertObject:[NSColor blueColor] atIndex:1];
[array addObject:[NSColor whiteColor]];
[array removeObjectInRange:(NSMakeRange(1, 2))];
// array теперь содержит redColor и whiteColor
```

В среде управления памятью, когда объект удаляется из массива, он получает сообщение `release`. Если этот объект принадлежал только этому массиву, то он будет уничтожен, после удаления из массива. Если вы хотите продолжать использовать этот объект, то вы должны послать ему сообщение `retain` перед удалением из массива.

Вы можете получить доступ к элементу массива по индексу, используя метод `objectAtIndex:`

```
NSString *s=[arrayOfStrings objectAtIndex: 2]; // доступ к третьему элементу массива
```

Методы `objectEnumerator` и `reverseObjectEnumerator` предоставляют последовательный доступ к элементам массива, отличаясь только в направлении движения по элементам. Кроме того, методы `makeObjectsPerformSelector:` и `makeObjectsPerformSelector:withObject:` позволяет отправлять сообщения во все объекты в массиве. В большинстве случаев, следует использовать быстрое перечисление, поскольку это быстрее и гибче, чем при использовании `NSEnumerator` или метода `makeObjectsPerformSelector:.`

Вы можете извлечь подмножество массива (`subarrayWithRange:`) или объединить элементы массива объекта `NSString` в одну строку (`componentsJoinedByString:`). Кроме того, вы можете сравнить два массива использованием методов `isEqualToArray:` и `firstObjectCommonWithArray:.` Наконец, вы можете создать новый массив, который

будет содержать объекты из существующего массива и один или несколько дополнительных объектов `arrayByAddingObject:` или `arrayByAddingObjectsFromArray:`.

Существуют два основных способа, которые можно использовать, чтобы определить, присутствует ли объект в массиве, `indexOfObject:` и `indexOfObjectIdenticalTo:`. Есть также два варианта, `indexOfObject:inRange:` и `indexOfObjectIdenticalTo:inRange:`, которые вы можете использовать для поиска в диапазоне, в пределах массива. Методы `indexOfObject:` для проверки на равенство, отправляют элементам в массиве сообщения `isEqual:`; методы `indexOfObjectIdenticalTo:` испытывают на равенство с помощью указателя сравнения. Разница показана в листинге ниже:

```
NSString *yes0 = @"yes";
NSString *yes1 = @"YES";
NSString *yes2 = [NSString stringWithFormat:@"%@", yes1];

NSArray *yesArray = [NSArray arrayWithObjects: yes0, yes1, yes2, nil];

NSUInteger index;

index = [yesArray indexOfObject:yes2];
// index is 1 (эквивалентен)

index = [yesArray indexOfObjectIdenticalTo:yes2];
// index is 2 (равен)
```

Вам может понадобиться сортировка массива на основе некоторых критериев. Например, вам может потребоваться разместить несколько созданных пользователем строк в алфавитном порядке, либо вам потребуется разместить номера в убыванию или по возрастанию. На рисунке 2 показан массив отсортированный по фамилиям и именам. Cocoa обеспечивает удобные способы для сортировки содержимого массива, такие как своего рода дескрипторы, блоков и селекторов.

Дескрипторы сортировки (экземпляры `NSSortDescriptor`) обеспечивают удобный и абстрактный способ описания порядка сортировки. Дескриптор сортировки предоставляет несколько полезных функций. Вы можете легко выполнять большинство операций сортировки с минимальным пользовательским кодом. Вы также можете использовать дескрипторы сортировки в сочетании с Cocoa bindings для сортировки содержимого, например, таблицы. Вы также можете использовать их с Core Data для сортировки результатов запроса выборки.

Если вы используете методы `sortedArrayUsingDescriptors:` или `sortUsingDescriptors:`, дескрипторы сортировки обеспечивают простой способ сортировки коллекций объектов, используя ряд их свойств. Учитывая множество словарей (пользовательские объекты работают таким же образом), вы можете отсортировать его содержимое по фамилии, имени. Следующий листинг показывает, как создать этот массив, а затем отсортировать с дескрипторами.

```
//Сначала создадим массив из словарей
NSString *LAST = @"lastName";
NSString *FIRST = @"firstName";

NSMutableArray *array = [NSMutableArray array];
NSArray *sortedArray;

NSDictionary *dict;
dict = [NSDictionary dictionaryWithObjectsAndKeys:
        @"Jo", FIRST, @"Smith", LAST, nil];
[array addObject:dict];

dict = [NSDictionary dictionaryWithObjectsAndKeys:
        @"Joe", FIRST, @"Smith", LAST, nil];
```

```

[array addObject:dict];

dict = [NSDictionary dictionaryWithObjectsAndKeys:
        @"Joe", FIRST, @"Smythe", LAST, nil];
[array addObject:dict];

dict = [NSDictionary dictionaryWithObjectsAndKeys:
        @"Joanne", FIRST, @"Smith", LAST, nil];
[array addObject:dict];

dict = [NSDictionary dictionaryWithObjectsAndKeys:
        @"Robert", FIRST, @"Jones", LAST, nil];
[array addObject:dict];

//Далее сортируем содержимое массива по last name затем first name

// Результаты могут быть показаны пользователю
// Обратите внимание на использование localizedCaseInsensitiveCompare:
selector

NSSortDescriptor *lastDescriptor =
    [[[NSSortDescriptor alloc] initWithKey:LAST
        ascending:YES
        selector:@selector(localizedCaseInsensitiveCompare:)]
    autorelease];
NSSortDescriptor *firstDescriptor =
    [[[NSSortDescriptor alloc] initWithKey:FIRST
        ascending:YES
        selector:@selector(localizedCaseInsensitiveCompare:)]
    autorelease];

```

```

NSArray *descriptors = [NSArray arrayWithObjects:lastDescriptor,
        firstDescriptor, nil];
sortedArray = [array sortedArrayUsingDescriptors:descriptors];

```

Направление и первый параметр сортировки легко изменить концептуально и программно, как показано ниже:

```

NSSortDescriptor *lastDescriptor =
    [[[NSSortDescriptor alloc] initWithKey:LAST
        ascending:NO
        selector:@selector(localizedCaseInsensitiveCompare:)]
    autorelease];

NSSortDescriptor *firstDescriptor =
    [[[NSSortDescriptor alloc] initWithKey:FIRST
        ascending:NO
        selector:@selector(localizedCaseInsensitiveCompare:)]
    autorelease];

NSArray *descriptors = [NSArray arrayWithObjects:firstDescriptor,
        lastDescriptor, nil];
sortedArray = [array sortedArrayUsingDescriptors:descriptors];

```

В частности, это несложно для создания подобных дескрипторов от пользовательского ввода.

В отличие от этого, ниже показана первая сортировка с помощью функции. Такой подход значительно менее гибкий.

```

NSInteger lastNameFirstNameSort(id person1, id person2, void *reverse)
{
    NSString *name1 = [person1 valueForKey:LAST];
    NSString *name2 = [person2 valueForKey:LAST];

```

```

        NSComparisonResult comparison = [name1
localizedCaseInsensitiveCompare:name2];
        if (comparison == NSOrderedSame) {
            name1 = [person1 valueForKey:FIRST];
            name2 = [person2 valueForKey:FIRST];
            comparison = [name1 localizedCaseInsensitiveCompare:name2];
        }

        if (*(BOOL *)reverse == YES) {
            return 0 - comparison;
        }
        return comparison;
    }

    BOOL reverseSort = YES;
    sortedArray = [array sortedArrayUsingFunction:lastNameFirstNameSort
context:&reverseSort];

```

Вы можете использовать *блоки*, чтобы отсортировать массив на основе пользовательских критериев. Метод `NSArray sortedArrayUsingComparator:` сортирует массив в новый массив, используя блок для сравнения объектов. Метод `sortUsingComparator:` класса `NSMutableArray` сортирует массив на месте, используя блок для сравнения объектов. Листинг ниже показывает, сортировку с блоком.

```

NSArray *sortedArray = [array sortedArrayUsingComparator: ^(id obj1, id obj2)
{
    if ([obj1 integerValue] > [obj2 integerValue]) {
        return (NSComparisonResult)NSOrderedDescending;
    }

    if ([obj1 integerValue] < [obj2 integerValue]) {
        return (NSComparisonResult)NSOrderedAscending;
    }
    return (NSComparisonResult)NSOrderedSame;
}]];

```

Следующий листинг иллюстрирует использование методов `sortedArrayUsingSelector:`, `sortedArrayUsingFunction:context:`, и `sortedArrayUsingFunction:context:hint:`. Самым сложным из этих методов является `sortedArrayUsingFunction:context:hint:`. Он наиболее эффективен, когда у вас есть большой массив (N записей), которые вам надо отсортировать раз и затем лишь слегка изменить (P добавлений и удалений, где P гораздо меньше, чем N). Вы можете использовать работу, которую вы сделали в оригинальной сортировке, и сделать своего рода слияние между N "старых" предметов и P "новых" предметов. Чтобы получить соответствующую подсказку, вы используете `sortedArrayHint` когда исходный массив был отсортирован, и держите его, пока вам это нужно (если вы хотите, отсортировать массив после того, как он был изменен).

```

NSInteger alphabeticSort(id string1, id string2, void *reverse)
{
    if (*(BOOL *)reverse == YES) {
        return [string2 localizedCaseInsensitiveCompare:string1];
    }
    return [string1 localizedCaseInsensitiveCompare:string2];
}

NSMutableArray *anArray =
[NSMutableArray arrayWithObjects:@"aa", @"ab",
    @"ac", @"ad", @"ae", @"af", @"ag",
    @"ah", @"ai", @"aj", @"ak", @"al", @"am", @"an",
    @"ao", @"ap", @"aq", @"ar", @"as", @"at",
    @"au", @"av", @"aw", @"ax", @"ay", @"az", @"ba",

```

```

        @"bb", @"bc", @"bd", @"bf", @"bg", @"bh",
        @"bi", @"bj", @"bk", @"bl", @"bm", @"bn", @"bo",
        @"bp", @"bq", @"br", @"bs", @"bt", @"bu",
        @"bv", @"bw", @"bx", @"by", @"bz", @"ca", @"cb",
        @"cc", @"cd", @"ce", @"cf", @"cg", @"ch",
        @"ci", @"cj", @"ck", @"cl", @"cm", @"cn", @"co",
        @"cp", @"cq", @"cr", @"cs", @"ct", @"cu",
        @"cv", @"cw", @"cx", @"cy", @"cz", nil];
// внимание: anArray отсортирован
NSData *sortedArrayHint = [anArray sortedArrayHint];

[anArray insertObject:@"be" atIndex:5];

NSArray *sortedArray;

// сортировка используя селектор
sortedArray =
    [anArray
     sortedArrayUsingSelector:@selector(localizedCaseInsensitiveCompare:)];

// сортировка используя функцию
BOOL reverseSort = NO;
sortedArray =
    [anArray sortedArrayUsingFunction:alphabeticSort context:&reverseSort];

// сортировка с подсказкой
sortedArray =
    [anArray sortedArrayUsingFunction:alphabeticSort
              context:&reverseSort
              hint:sortedArrayHint];

```

Важно: Если вы сортируете массив строк, который должен быть показан конечному пользователю, вы должны всегда использовать локализованные сравнения. На других языках или контекстах, правильный порядок в отсортированном массиве может быть различным.

Классы NSArray и NSMutableArray предоставляют методы для фильтрации содержимого массива. NSArray обеспечивает `filteredArrayUsingPredicate:`, который возвращает новый массив, содержащий объекты в приемнике, который соответствует указанному предикату. NSMutableArray добавляет `filterUsingPredicate:`, который оценивает содержимое получателя по отношению к указанному предикату и оставляет только те объекты, которые соответствуют. Эти методы показаны ниже.

```

NSMutableArray *array =
    [NSMutableArray arrayWithObjects:@"Bill", @"Ben", @"Chris", @"Melissa",
     nil];

NSPredicate *bPredicate =
    [NSPredicate predicateWithFormat:@"SELF beginswith[c] 'b'"];
NSArray *beginWithB =
    [array filteredArrayUsingPredicate:bPredicate];
// beginWithB содержит { @"Bill", @"Ben" }.

NSPredicate *sPredicate =
    [NSPredicate predicateWithFormat:@"SELF contains[c] 's'"];
[array filterUsingPredicate:sPredicate];
// array теперь содержит { @"Chris", @"Melissa" }

```

Вы можете также фильтровать массив, используя объект `NSIndexSet`. NSArray обеспечивает `objectsAtIndexes:`, который возвращает новый массив, содержащий объекты на индексах в соответствующем индексном наборе. NSMutableArray добавляет `removeObjectsAtIndexes:`, который позволяет фильтровать массив на месте с помощью набора индексов.

IOS Примечание: класс `NSPointerArray` не доступен в IOS.

Класс `NSPointerArray` по умолчанию настраивается на распоряжение объектами также, как это делает `NSMutableArray`, кроме того, что он может содержать `nil` значения и того, что метод `count` отражает эти нулевые значения. Он также позволяет дополнительные опции по хранению данных, которые вы можете приспособить для конкретных случаев, например, когда вам нужны расширенные возможности управления памятью, или если вы хотите распоряжаться определенным типом указателя. Например, массив указателей на рисунке 3 настроен на распоряжение слабыми (*weak*) ссылками на его содержимое. Вы также можете указать, хотите ли вы скопировать объекты, которые введены в массив.

Вы можете использовать `NSPointerArray` объект, если вы хотите получить упорядоченный набор, который использует слабые (*weak*) ссылки в среде сборки мусора. Например, предположим, у вас есть глобальный массив, который содержит некоторые объекты. Поскольку глобальные объекты не собираются, ни один объект из его содержимого не может быть освобожден, если они проводятся слабо. Массив указателей настроен на слабое распоряжение объектами, не владея их содержимым. Если нет сильных (*strong*) ссылок на объекты в таком массиве указателей, эти объекты могут быть собраны сборщиком мусора. Например, массив указателей на рисунке 3 имеет слабые ссылки на содержимое. В течение следующего сбора, объекты D и E освобождаются (и их указатели устанавливаются в `nil`), а остальные объекты остаются. Если это должным образом не учесть, такая практика может привести к непредсказуемому поведению.

Для создания массива указателей, создайте или инициализируйте его с помощью `pointerArrayWithOptions:` или `initWithOptions:`, и соответствующими параметрами `NSPointerFunctionsOptions`. Альтернативно вы можете инициализировать его с помощью `initWithPointerFunctions:`, и соответствующих экземпляров `NSPointerFunctions`.

Класс `NSPointerArray` также определяет ряд удобных конструкторов для создания массива указателей с сильными или слабыми ссылками на его содержимое. Например, `pointerArrayWithWeakObjects` создает массив указателей, который содержит слабые ссылки на его содержимое. Эти удобства конструкторов должны использоваться только если вы храните объекты.

Для настройки массива указателей использующий произвольные указатели можно инициализировав его с опциями `NSPointerFunctionsOpaqueMemory` и `NSPointerFunctionsOpaquePersonality`. Например, вы можете добавить указатель на целочисленное значение, используя подход, показанный в листинге ниже:

```
NSPointerFunctionOptions options=(NSPointerFunctionsOpaqueMemory |
    NSPointerFunctionsOpaquePersonality);

NSPointerArray *ptrArray=[NSPointerArray pointerArrayWithOptions: options];

[ptrArray addPointer: someIntPtr];
```

Вы можете получить доступ к целым, как показано ниже.

```
NSLog(@" Index 0 contains: %i", *(int *) [ptrArray pointerAtIndex: 0] );
```

Когда используются настройки на использование произвольных указателей, массив указателей получает риски, связанные с использованием указателей. Например, если указатели ссылаются на стековые данные, созданные внутри функции, эти указатели не являются допустимыми за пределами функции, даже если указатели в массиве. Попытка получить доступ к ним приведет к неопределенному поведению.

10.3 Словари: коллекции ключ-значение

Словари управляют парами ключей и значений. Пара ключ-значение в словаре называется записью. Каждая запись состоит из одного объекта, представляющего ключ, а

второй объект, который является значением ключа. В словаре, ключи являются уникальными, то есть в одном словаре нет двух эквивалентных ключей (как определено isEqual:). Ключом может быть любой объект, который принимает NSCopying протокол и реализует hash и isEqual: методы [3].

Информация по производительности:

- Доступ к элементу словаря занимает постоянное время.
- Установка и удаление занимают постоянное время.

Эти показатели действительны при использовании ключей, таких как NSString, которые имеют предопределенную hash-функцию. При использовании пользовательских объектов в качестве ключей, производительность зависит от hash-метода определенного для объектов. При плохой hash-функции, доступ и редактирование выполняется за линейное время.

Объект NSDictionary управляет неизменным словарем, то есть после создания словаря, Вы не можете добавлять, удалять или заменять ключи и значения. Можно, однако, изменить отдельные значения сами по себе (если они поддерживают изменения), но ключи не должны изменяться. Изменчивость коллекции не влияет на изменчивость объектов в коллекции. Вы должны использовать неизменные словари, если в словаре редко происходят изменения или изменения носят массовый характер.

Объект NSMutableDictionary управляет изменяемым словарем, который позволяет добавлять и удалять записи в любое время, автоматически выделяя память по мере необходимости. Вы должны использовать изменяемый словарь, если словарь меняется постепенно, или очень большой, так как большие коллекции занимают больше времени для инициализации.

Вы можете легко создать экземпляр одного типа словаря с другим, используя инициализатор initWithDictionary: или удобный конструктор dictionaryWithDictionary:. Например, если у вас есть экземпляр NSDictionary, myDictionary, вы можете создать изменяемую копию следующим образом:

```
NSMutableDictionary *myMutableDictionary =  
[NSMutableDictionary dictionaryWithDictionary: myDictionary];
```

В общем, вы реализуете словарь, посылая одно из сообщений dictionaryWithDictionary: либо NSDictionary или NSMutableDictionary класса. Сообщение dictionaryWithDictionary: возвращает словарь, содержащий ключи и значения, передаваемые в качестве аргументов. Объекты, добавленные в качестве значений в словарь, не копируются (если вы не передаете YES в качестве аргумента initWithDictionary:copyItems:). Скорее всего, объект добавляется непосредственно в словарь. В управляемой среде памяти, объект получает retain сообщение, когда он добавляется. Когда словарь будет освобожден, среда управления памятью посылает в каждый элемент release сообщение.

Внутренне, словарь использует хэш-таблицу, чтобы организовать его хранение и обеспечить быстрый доступ к значению данных соответствующих ключу. Тем не менее, методы, определенные для словарей ограждают Вас от сложностей работы с хэш-таблицами, функциями хеширования, или хэш-значениями ключей. Методы принимают непосредственно ключи, а не их хэш-формы.

Ключ и значение объектов, которые составляют запись получают release сообщение. Если больше нет ссылок на объекты, они освобождаются. Если ваша программа содержит ссылку на такой объект, и вы не являетесь владельцем объекта, ссылка становится недействительной, если вы хотите использовать ссылку на объект отправьте объекту сообщение retain, прежде чем он будет удален из словаря.

Доступ к объекту после его удаления, **опасно**:

```
id anObject = [aDictionary objectForKey:theKey];  
  
[aDictionary removeObjectForKey:theKey];
```



```
[anObject someMessage]; // возможно падение приложения
```

Чтобы защититься от этой ситуации, вы можете сохранить объект вызовом *retain*, прежде чем удалить его.

```
id anObject = [[aDictionary objectForKey:theKey] retain];
```

```
[aDictionary removeObjectForKey:theKey];
```

```
[anObject someMessage];
```

```
//не забудьте отправить anObject сообщение release, когда вы с ним закончите
```

Процедура добавления объектов в изменяемый словарь относительно проста. Чтобы добавить одну пару ключ-значение, или заменить объект для конкретного ключа, используйте метод экземпляра `setObject:forKey:`.

```
NSString *LAST=@"lastName";
```

```
NSString *FIRST=@"firstName";
```

```
NSMutableDictionary *dict=[NSMutableDictionary dictionaryWithObjectsAndKeys:  
    @"Jo", FIRST, @"Smith", LAST, nil];
```

```
NSString *MIDDLE=@"middleInitial";
```

```
[dict setObject: @"M" forKey:MIDDLE];
```

Вы также можете добавлять записи из другого словаря, используя метод экземпляра `addEntriesFromDictionary:`. Если оба словаря содержат один и тот же ключ, предыдущее значение объекта получателя для этого ключа освобождается, а новый объект занимает его место. Например, после того, как код в листинге ниже выполнится, `dict` будет иметь значение "Jones" для ключа "lastName".

```
NSString *LAST=@"lastName";
```

```
NSString *FIRST=@"firstName";
```

```
NSString *SUFFIX=@"suffix";
```

```
NSString *TITLE=@"title";
```

```
NSMutableDictionary *dict=[NSMutableDictionary dictionaryWithObjectsAndKeys:  
    @"Jo", FIRST, @"Smith", LAST, nil];
```

```
NSDictionary *newDict=[NSDictionary dictionaryWithObjectsAndKeys:  
    @"Jones", LAST, @"Hon.", TITLE, @"J.D.", SUFFIX, nil];
```

```
[dict addEntriesFromDictionary: newDict];
```

`NSDictionary` предоставляет метод `keysSortedByValueUsingSelector:`, который возвращает массив ключей словаря в порядке, как если бы словарь был отсортирован по его значениям:

```
NSDictionary *dict = [NSDictionary dictionaryWithObjectsAndKeys:  
    [NSNumber numberWithInt:63], @"Mathematics",  
    [NSNumber numberWithInt:72], @"English",  
    [NSNumber numberWithInt:55], @"History",  
    [NSNumber numberWithInt:49], @"Geography",  
    nil];
```

```
NSArray *sortedKeysArray =
```

```
    [dict keysSortedByValueUsingSelector:@selector(compare:)];
```

```
// sortedKeysArray содержит: (Geography, History, Mathematics, English)
```

Вы также можете использовать блоки, чтобы с легкостью упорядочить ключи словаря на основе соответствующих значений. Метод `NSDictionary keysSortedByValueUsingComparator:` позволяет использовать блок для сравнения ключей, которые должны быть отсортированы в новый массив.

```
NSArray *blockSortedKeys =
```

```
    [dict keysSortedByValueUsingComparator: ^(id obj1, id obj2) {
```

```

    if ([obj1 integerValue] > [obj2 integerValue]) {
        return (NSComparisonResult)NSOrderedDescending;
    }

    if ([obj1 integerValue] < [obj2 integerValue]) {
        return (NSComparisonResult)NSOrderedAscending;
    }
    return (NSComparisonResult)NSOrderedSame;
}];

```

В большинстве случаев, предоставляемых объектов Сосоа, таких как объекты `NSString` должно быть достаточно для использования в качестве ключей. В некоторых случаях, однако, может возникнуть необходимость использовать пользовательские объекты в качестве ключей в словаре. При использовании пользовательских объектов в качестве ключей, есть несколько важных моментов, которые необходимо держать в голове.

Ключи должны соответствовать `NSCopying` протоколу. Методы, добавления записи в словарь, которые могут выступать как часть инициализации (для всех словарей) или во время модификации (для изменяемых словарей) - не добавляют каждый объект-ключ в словарь прямо. Вместо этого они копируют каждый аргумент-ключ и добавляют копию в словарь. После копирования в словарь, принадлежащие словарию копии ключей не должны быть изменены.

Ключи должны реализовать `hash` и `isEqual:` методы, потому что словарь использует хэш-таблицу, чтобы организовать его хранение и быстрый доступ к содержащимся объектам. Кроме того, производительность в словаре сильно зависит от используемой хэш-функции. При плохой хэш-функции, снижение производительности может быть весьма серьезными. Для получения дополнительной информации по `hash` и `isEqual:` см. методы `NSObject`.

Важно: Поскольку словарь копирует каждый ключ, ключи должны соответствовать `NSCopying` протоколу. Имейте это в виду, при выборе объектов для использования в качестве ключей. Хотя вы можете использовать любой объект, который принимает `NSCopying` протокол и реализует `hash` и `isEqual:` методы, обычно это плохой стиль, использовать крупные объекты, такие как экземпляры `UIImage`, так как это может потребовать неподъемной производительности.

IOS Примечание: класс `NSMutableDictionary` не доступен в IOS.

Класс `NSMutableDictionary` по умолчанию настроен удерживать объекты так же, как `NSMutableDictionary`. Он также предоставляет дополнительные опции хранения данных, которые вы можете приспособить для конкретных случаев, например, когда вам необходимы продвинутые варианты управления памятью, или, если вы хотите сохранить определенный тип указателя. Например, карта таблиц на рисунке 2 настроена для хранения слабых ссылок на объекты своих значений. Вы также можете указать, хотите ли вы скопировать объекты, которые вошли в массив.

Вы можете использовать `NSMutableDictionary` объект, если вы хотите получить коллекцию пар ключ-значение которые используют слабые (weak) ссылки в среде сборки мусора. Например, предположим, у вас есть глобальная карта таблица, которая содержит некоторые объекты. Поскольку глобальные объекты не собираются, ни один из его содержимых объектов не может быть освобожден, если они проводятся слабо. Карта таблиц настроена для хранения объектов слабо не владея их содержимым. Если нет сильных ссылок на объекты в такой таблице карте, эти объекты могут быть собраны. Например, карта таблиц на рисунке 2, имеет слабые ссылки на содержимое. В течение следующего сбора мусора, объекты D и E освобождаются, а остальные объекты остаются. Если это должным образом не учитывать, такая практика может привести к непредсказуемому поведению.

Чтобы создать карту таблицу, создайте или инициализируйте ее с помощью `mapTableWithOptions:valueOptions:` или `initWithKeyOptions:valueOptions:capacity:`, и соответствующим указателем функции вариантов. Кроме того, вы можете инициализировать ее с помощью `initWithKeyPointerFunctions:valuePointerFunctions:capacity:`, и соответствующими экземплярами `NSPointerFunctions`.

`NSMutableDictionary` класс также определяет ряд удобства конструкторов для создания карты таблицу с сильной или слабой ссылки на его содержание. Например, `NSMutableDictionaryWithStrongToWeakObjects` создает карту таблицу, которая содержит сильные ссылки на свои ключи и слабые ссылки на свои значения. Эти удобства конструкторов должны использоваться только если вы храните объекты.

Важно только параметры, перечисленные в `NSMutableDictionaryOptions` гарантируют то, что остальная часть **API** будет работать правильно, в том числе копирование, архивирование и быстрое перечисление. В то время как другие варианты `NSPointerFunctions` используются для определенных конфигураций, таких, как хранение произвольных указателей, не все комбинации параметров являются действительными. С некоторыми комбинациями, карты таблицы могут работать неправильно или не могут даже правильно инициализироваться.

Чтобы настроить `NSMutableDictionary` на использование произвольных указателей, инициализируйте его с опциями `NSPointerFunctionsOpaqueMemory` и `NSPointerFunctionsOpaquePersonality`. Ключевые значения и параметры не должны быть одинаковыми. При использовании карта таблица содержит произвольные указатели, должны быть использованы **API C** функции для указателей `void *`. Например, вы можете добавить указатель на целочисленное значение, используя подход, показанный ниже. Обратите внимание, что карта таблица использует `NSString` объекты в качестве ключей, а также, что ключи будут скопированы в карту таблицу.

```
NSPointerFunctionsOptions keyOptions=NSPointerFunctionsStrongMemory |
    NSPointerFunctionsObjectPersonality | NSPointerFunctionsCopyIn;
NSMutableDictionaryOptions valueOptions=NSPointerFunctionsOpaqueMemory |
    NSPointerFunctionsOpaquePersonality;
```

```
NSMutableDictionary *mapTable=[NSMutableDictionary dictionaryWithKeyOptions: keyOptions
    valueOptions:
        valueOptions];
```

```
NSString *KEY1=@"Key1";
NSMutableDictionaryInsert(mapTable ,KEY1 , someIntPtr);
```

Вы можете получить доступ к этому целому с помощью `NSMutableDictionaryGet`.

```
NSLog(@" Key1 contains: %i", *(int *) NSMutableDictionaryGet(mapTable, @"Key1"));
```

Когда используются настройки на использование произвольных указателей, карта таблица получает риски, связанные с использованием указателей. Например, если указатели ссылаются на стековые данные, созданные внутри функции, эти указатели не являются допустимыми за пределами функции, даже если указатели в массиве. Попытка получить доступ к ним приведет к неопределенному поведению.

10.4 Наборы: неупорядоченные коллекции объектов

Набор представляет собой неупорядоченную коллекцию объектов. Вы можете использовать наборы как альтернативу массивам, когда порядок элементов не имеет значения и выполнение проверки, находится ли объект в наборе очень важно. Хотя массивы упорядочены, проверка их на членство медленнее, чем тестирование набора [3].

Информация по производительности:

- Доступ к элементу набора занимает постоянное время.
- Установка и удаление занимает постоянное время.

Эти показатели предполагают адекватную реализацию для хэш-методов определенных для объектов. При плохой реализации хэш-функций, доступ и редактирование выполняется за линейное время.

Объект `NSSet` управляет неизменным набором различных объектов, то есть после создания набора, вы не можете добавлять, удалять или заменять объекты. Можно, однако, изменить отдельные объекты сами (если они поддерживают изменения). Изменчивость коллекции не влияет на изменчивость объектов в коллекции. Вы должны использовать неизменный набор если он изменяется в редких случаях или изменения носят массовый характер.

`NSMutableSet`, подкласс `NSSet`, является изменяемым набором различных объектов, который позволяет добавлять и удалять записи в любое время, автоматически выделяя память по мере необходимости. Вы должны использовать изменяемые наборы, если они меняются постепенно, или очень большие, так как большие коллекции занимают больше времени для инициализации.

`NSCountedSet`, подкласс `NSMutableSet`, является изменяемым набором, в который можно добавить определенный объект несколько раз, другими словами, элементы набора не обязательно различны. `NSCountedSet` также известен как мешок. Набор хранит счетчик, связанный с каждым отдельным вставленным объектом. Объекты `NSCountedSet` отслеживают, сколько раз объекты вставляются и требует, чтобы объекты были удалены тоже количество раз, чтобы полностью удалить объект из набора. Таким образом, существует только один экземпляр объекта учитываемый набором, даже если объект был добавлен несколько раз. Метод `countForObject:` возвращает количество раз, которое указанный объект был добавлен к этому набору.

Объекты в наборе должны отвечать на методы протокола `NSObject` `hash` и `isEqual:` (см. `NSObject` для получения дополнительной информации). Если изменяемые объекты хранятся в наборе, `hash` метод объекта не должен зависеть от внутреннего состояния изменяемых объектов или изменяемые объекты не должны быть изменены, пока они в наборе. Например, изменяемый словарь может быть положен в набор, но вы не должны его менять, пока он там. (Заметим, что может быть трудно понять, находится ли данный объект в коллекции).

`NSSet` предоставляет ряд методов инициализаторов, такие как `initWithObjects:`, и `initWithArray:`, которые возвращают `NSSet` объект, содержащий элементы (если таковые имеются), которые вы передаете в качестве аргумента. Объекты добавляются в набор и не копируются (если только вы не передаете `YES` в качестве аргумента `initWithSet:copyItems:`). Скорее всего, объект добавляется непосредственно в набор. В среде управления памятью, объект получает `retain` сообщение, когда он добавлен.

Наборы, за исключением `NSCountedSet`, являются предпочтительными коллекциями, если вы хотите быть уверены, что объект не предстанет больше, чем один раз, и нет прямого эффекта при добавлении объекта несколько раз.

Вы можете создать `NSMutableSet` объект с помощью любого из инициализаторов предоставляемых `NSSet`. Вы можете создать `NSMutableSet` объект из экземпляра `NSSet` (или наоборот) с помощью `initWithSet:` или `initWithSet:`.

Класс `NSMutableSet` предоставляет следующие методы для добавления объектов в набор:

- `addObject:` добавляет один объект в набор.
- `addObjectsFromArray:` добавляет все объекты из заданного массива в набор.
- `unionSet:` добавляет все объекты из другого набора которого уже нет.

`NSMutableSet` класс дополнительно предоставляет следующие методы для удаления объектов из набора:

- `intersectSet:` удаляет все объекты, которые не находятся в другом наборе.
- `removeAllObjects` удаляет все объекты из набора.
- `removeObject:` удаляет конкретный объект из множества.

- `minusSet`: удаляет все объекты, которые находятся в другом наборе.

Поскольку `NSCountedSet` является подклассом `NSMutableSet`, он наследует все эти методы. Тем не менее, некоторые из них ведут себя немного по-другому с `NSCountedSet`. Например:

- `unionSet`: добавляет все объекты из другого набора, даже если они уже присутствуют.
- `intersectSet`: удаляет все объекты, которые не находятся в другом наборе. Если имеется несколько экземпляров одного и того же объекта в любом множестве, результирующий набор содержит этот объект столько раз, сколько набор с меньшим количеством экземпляров.
- `minusSet`: удаляет все объекты, которые находятся в другом наборе. Если объект представлен более одного раза в `NSCountedSet`, то удаляется только один экземпляр.

В среде управления памятью, когда объект удаляется из изменяемого набора он получает `release` сообщение. Если этот объект принадлежит только этому набору (стандартные правила управления памятью) объект будет освобожден, после того, как он будет удален. Если вы хотите использовать объект после его удаления, вы должны отправить ему `retain` сообщение, прежде чем он будет удален из набора. Например, если (в листинге 1) `aSet` является единственным владельцем `anObject`, третий результат в заявлении сообщит об ошибке выполнения.

Листинг 1. Доступ к объекту после его удаления **опасен**.

```
id anObject = [aSet anyObject];
[aSet removeObject:anObject];
// если объектом владел только aSet, то следующая строка приведет к ошибке
выполнения
[anObject someMessage];
```

Если вы хотите использовать объект после его удаления из набора, вы должны сначала его сохранить, как показано в листинге 2.

Листинг 2. Сохранение объектов перед удалением дает вам право собственности

```
id anObject = [[aSet anyObject] retain];
[aSet removeObject:anObject];
[anObject someMessage];
// не забудьте послать anObject сообщение release когда закончите с ним
работать
```

Класс `NSSet` предоставляет следующие методы для запроса элементов набора:

- `allObjects` возвращает массив, содержащий объекты в наборе.
- `anyObject` возвращает некоторый объект в наборе. (Объект выбирается преимуществом, а не случайно.)
- `count` возвращает количество объектов в настоящее время в наборе.
- `member:` возвращает объект в наборе, эквивалентный указанному объекту.
- `intersectsSet:` проверяет два набора на разделение по крайней мере одного объекта.
- `isEqualToSet:` проверяет два множества на эквивалентность.
- `isSubsetOfSet:` проверяет все объекты, содержащиеся в наборе на присутствие и в другом наборе.

Метод класса `NSSet` `objectEnumerator` позволяет обходить элементы множества по одному. Методы `makeObjectsPerformSelector:` и `makeObjectsPerformSelector:withObject:` используются для отправки сообщений на отдельные объекты в наборе. В большинстве случаев, следует использовать быстрое перечисление, потому что это быстрее и гибче, чем при использовании `NSEnumerator` или метода `makeObjectsPerformSelector:.`

IOS Примечание: класс `NSHashTable` не доступен в IOS.

NSHashTable класс по умолчанию настроен хранить объекты так же, как это делает NSMutableSet. Он также позволяет дополнительные опции для хранения данных, которые вы можете приспособить для конкретных случаев, например, когда вам нужны продвинутое варианты управления памятью, или, если вы хотите держать определенный тип указателя. Например, карта таблицы на рисунке 2 настроена на хранение слабых ссылок на ее элементы. Вы также можете указать, хотите ли вы скопировать объекты, которые вошли в набор.

Вы можете использовать NSHashTable объект, если вы хотите получить неупорядоченный набор элементов, которые используют слабые ссылки в среде сборки мусора. Например, предположим, у вас есть глобальная хэш-таблица, которая содержит некоторые объекты. Поскольку глобальные объекты не собираются, ни один из его содержимых объектов не может быть освобожден, если они проводятся слабо. Хэш-таблицы настроенные для хранения объектов слабо (weak) не владеют их содержимым. Если нет сильных ссылок на объекты в такой хэш-таблице, эти объекты могут быть освобождены. Например, хэш-таблица на рисунке 2, имеет слабые ссылки на ее содержимое. В следующей коллекции, объекты A, C и Z, освобождены, а остальные объекты остаются. Если это должным образом не учитывать, такая практика может привести к непредсказуемому поведению.

Для создания хэш-таблицы, инициализируйте ее с помощью метода initWithOptions:capacity: и соответствующего указателя функции опций. Кроме того, вы можете инициализировать ее с помощью initWithPointerFunctions:capacity:, и соответствующего экземпляра NSPointerFunctions.

NSHashTable класс также определяет конструктор hashTableWithWeakObjects, удобный для создания хэш-таблиц со слабыми (weak) ссылками на его содержание. Она должна использоваться только если вы храните объекты.

Важно только параметры, перечисленные в NSHashTableOptions гарантируют то, что остальная часть API будет работать правильно, в том числе копирование, архивирование и быстрое перечисление. В то время как другие варианты NSPointerFunctions используются для определенных конфигураций, таких, как хранение произвольных указателей, не все комбинации параметров являются действительными. С некоторыми комбинациями, карты таблицы могут работать неправильно или не могут даже правильно инициализироваться.

Для настройки хэш-таблицы использующей произвольные указатели, инициализируйте ее с параметрами NSPointerFunctionsOpaqueMemory и NSPointerFunctionsOpaquePersonality. При использовании хэш-таблицы содержащих произвольные указатели, должны быть использованы API C функции для указателей void *. Например, вы можете добавить указатель на целочисленное значение, используя подход показанный в листинге 3.

Листинг 3: хэш-таблица настроенная на *nonobject* указатели.

```
NSHashTable *hashTable=[[NSHashTable alloc] initWithOptions:
    NSPointerFunctionsOpaqueMemory |NSPointerFunctionsOpaquePersonality
    capacity: 1];

NSHashTableInsert(hashTable, someIntPtr);

//Не забудьте послать release сообщение к hashTable когда закончите с ним
работать
```

Когда используются настройки на использование произвольных указателей, хэш-таблица получает риски, связанные с использованием указателей. Например, если указатели ссылаются на стековые данные, созданные внутри функции, эти указатели не являются допустимыми за пределами функции, даже если указатели в массиве. Попытка получить доступ к ним приведет к неопределенному поведению.

10.5 Индекс наборы: Хранение индексов в массиве

Вы можете использовать индекс наборы для хранения индексов в другие структуры данных, такие как объект `NSArray`. Каждый индекс в наборе индексов может появляться только один раз, поэтому наборы индексов не подходят для хранения произвольных наборов целых чисел. Поскольку индексные наборы используют диапазоны для хранения индексов, они, как правило, более эффективны, чем хранение коллекций целых чисел, например, в массиве [3].

Объект `NSIndexSet` управляет набором неизменных показателей, то есть после создания набора индексов, вы не можете добавить к нему индексы или удалить индексы из него.

Объект `NSMutableIndexSet` управляет изменяемыми наборами индексов, которые позволяют добавлять и удалять индексы в любое время, автоматически выделяя память по мере необходимости.

Вы можете легко создать экземпляр одного типа набора индексов из другого, используя инициализатор `initWithIndexSet:`. Это особенно полезно, если вы хотите создать неизменный индекс набор, содержащий непересекающиеся наборы индексов, которые обычно создается с помощью изменяемых индексных наборов. Например, если у вас есть объект с именем `NSMutableIndexSet myIndexes`, которому были добавлены индексы, вы можете создать его неизменяемую копию следующим образом:

```
NSIndexSet *myImmutableIndexes=[[NSIndexSet alloc] initWithIndexSet:
myIndexes];
```

Вы также можете инициализировать набор индексов из одного индекса или ряда индексов с помощью методов `initWithIndex:` или `initWithIndexesInRange:`.

Методы `NSMutableIndexSet` класса позволяют добавлять или удалять дополнительные индексы и индексы диапазонов. Можно, например, сохранить непересекающиеся наборы индексов и изменять существующие наборы индексов по мере необходимости. Некоторые из этих методов перечислены ниже:

- `addIndex:` добавляет индекс
- `addIndexesInRange:` добавляет индексы в диапазоне
- `removeIndex:` удаляет индекс
- `removeIndexesInRange:` удаляет индексы в диапазоне

Если у вас есть пустой `NSMutableIndexSet` объект с именем `myDisjointIndexes`, вы можете заполнить его с индексами: 1, 2, 5, 6, 7 и 10, как показано в листинге 1.

Листинг 1 Добавление индексов в изменяемый набор индексов

```
[myDisjointIndexes addIndexesInRange: NSMakeRange(1,2)];
[myDisjointIndexes addIndexesInRange: NSMakeRange(5,3)];
[myDisjointIndexes addIndex: 10];
```

Для доступа ко всем проиндексированным объектам индексного набора может быть удобен последовательный проход через индексный набор. Перебор индекс набора, а не соответствующего массива, является более эффективным, поскольку он позволяет рассматривать только индексы, в которых вы заинтересованы. Если у вас есть объект `NSArray` с именем `anArray` и объект `NSIndexSet` с именем `anIndexSet`, вы можете проходить вперед через индекс набор, как показано в листинге 2.

Листинг 2 Прогрессивная итерация индекс набора

```
NSUInteger index=[anIndexSet firstIndex];

while(index != NSNotFound)
{
    NSLog(@" %@", [anArray objectAtIndex:index]);
    index=[anIndexSet indexGreaterThanIndex: index];
}
```

Иногда может быть необходимо осуществить проход назад через индексный набор, например, если вы хотите, выборочно удалить объекты с индексами из объекта NSMutableArray. Вы можете перемещаться в обратном направлении по индексам, как показано в листинге 3.

Листинг 3 Обратная итерация индекс набора

```
NSUInteger index=[anIndexSet lastIndex];

while(index != NSNotFound)
{
    if([[NSMutableArray objectAtIndex:index] isEqualToString:@"G"]){
        [NSMutableArray removeObjectAtIndex:index];
    }
    index=[anIndexSet indexLessThanIndex:index];
}
```

Такой подход следует использовать, только если вы хотите выборочно удалить объекты, указанные в индекс наборе. Если вы хотите удалить объекты для всех индексов в индекс наборе, используйте `removeObjectsAtIndexes:.`

Индексные наборы особенно эффективны, когда используются в сочетании с блоками. Блоки позволяют создавать наборы индексов, которые указывают на элементы массива, которые проходят несколько тестов. Например, если у вас есть несортированный массив чисел, и вы хотите создать индекс, который содержит набор показателей для всех чисел, меньших, чем 20, можно использовать что-то похожее на Листинг 4.

Листинг 4. Создание индексов из массива с помощью блока

```
NSIndexSet *lessThan20=[someArray indexesOfObjectsPassingTest:^(id obj,
    NSUInteger index, BOOL *stop){
    if ([obj isLessThan:[NSNumber numberWithInt:20]]){
        return YES;
    }
    return NO;
}]];
```

Индексные наборы также могут быть использованы в перечислении массива на основе блока. Чтобы перечислить только индексы массива, содержащиеся в индекс массиве, используйте метод `enumerateObjectsAtIndexes:options:usingBlock:.`

Кроме того, индекс массивы сами могут быть перечислены с помощью блока методом `enumerateIndexesUsingBlock:.` Например, вы можете выполнить задачу для каждого объекта, индекс которого в наборе. Вы даже можете получить доступ к объектам из нескольких массивов при условии, что индекс набор справедлив для использования массивов, как показано в листинге 5.

Листинг 5 Перечисление индекс набора для доступа к нескольким массивам

```
[anIndexSet enumerateIndexesUsingBlock:^(NSUInteger idx, BOOL *stop){
    if([[firstArray objectAtIndex:idx] isEqual:[secondArray objectAtIndex:idx]]){
        NSLog(@"Objects at %i Equal",idx);
    }
}]];
```

10.6 Индекс пути: хранение пути через вложенные массивы

Индекс пути хранить путь через набор вложенных массивов и используется для получения объекта в более сложной иерархии коллекций, как дерево [3].

Если рассматривать иерархию гипотетической компании то, корневой массив состоит из одного входа для генерального директора (СЕО). Массив ниже, состоит из различных вице-президентов. Под каждым вице-президентом стоит массив из директоров, и так далее. Если вы хотите сохранить позиции конкретного сотрудника в команде маркетинга в Европе, простого индекса недостаточно. В этом случае, Bill T. может быть представлен индексом пути 0.0.1.1.2.

Вы можете создать индекс путь, используя один из индексов или массив в С-стиле значений `NSUInteger`. Листинг 1 показывает, как создать индекс путь Bill T.

Листинг 1. Создание индекса пути из массива

```
NSUInteger arrayLength=5;
NSUInteger integerArray[]={0,0,1,1,2};
NSIndexPath *aPath=[[NSIndexPath alloc] initWithIndexes: integerArray
length: arrayLength];
```

Вы также можете создать индекс путь автоматически из многих более сложных иерархий коллекций классов. См. `indexPath` метод класса `NSTreeNode` для примера.

`NSIndexPath` предоставляет методы для запроса элементов в пути. Например, `indexPathAtPosition:` возвращает индекс хранящийся в заданной позиции в индексе пути. Вы также можете создавать новые индекс пути путем добавления нового индекса или удаления последнего индекса. Несколько классов используют индекс пути широко управляя их содержимым. `NSTreeController` является одним из таких примеров.

В iOS, `UITableView`, его представители и источник данных используют индексы пути для управления своим содержанием и для обработки взаимодействия с пользователем. Чтобы помочь в этом, `UIKit` добавляет программные интерфейсы для `NSIndexPath` включая строки и разделы в таблице в индексы путей. Например, индекс пути, используемый для обозначения выбора пользователя, использует метод делегата `tableView:didSelectRowAtIndexPath:.`

10.7 Копирование коллекций

Есть два вида копирования объекта: неполное и полное. Нормальная копия - неполная копия, производящая новую коллекцию, которая разделяет собственность на объекты с оригиналом. Полная копия создать новые объекты из оригиналов и добавить их в новую коллекцию [3].

Есть несколько способов сделать неполную копию коллекции. При создании неполную копии, объектам в оригинальной коллекции направляется сообщение `retain` и указатели копируются в новую коллекцию. Пример ниже, показывает некоторые способы создания новой коллекции, используя неполную копию.

```
NSArray *shallowCopyArray=[someArray copyWithZone:nil];

NSDictionary *shallowCopyDict=[[NSDictionary alloc] initWithDictionary:
someDictionary copyItems: NO];
```

Эти методы не ограничиваются показанными коллекциями. Например, вы можете скопировать набор методами `copyWithZone:` или `mutableCopyWithZone:` или массив `initWithArray:copyItems:.`

Есть два способа сделать полные копии коллекций. Вы можете использовать эквивалентные коллекции из `initWithArray:copyItems:` с `YES` в качестве второго параметра. Если вы создадите полную копию коллекции таким образом, каждому объекту в коллекции передается сообщение `copyWithZone:.` Если объекты в коллекции принимают `NSCopying` протокол, объекты полностью копируются в новую коллекцию, которая затем единолично владеет копируемыми объектами. Если объекты не принимают `NSCopying` протокол, попытка скопировать их таким образом, приведет к ошибке выполнения. Тем не менее, `copyWithZone:` производит поверхностное копирование. Такое копирование

способно производить только один уровень глубокого копирования. Если вам нужен только один уровень полной копии, вы можете явно вызвать инициализатор, как в листинге 2.

Листинг 2. Создание полной копии.

```
NSArray *deepCopyArray=[[NSArray alloc] initWithArray: someArray copyItems: YES];
```

Эта техника также относится и к другим коллекциям. Используйте эквивалент коллекции из `initWithArray:copyItems:` с `YES` в качестве второго параметра.

Если вам необходимо истинно полное копирование, например, когда у вас есть массив массивов, вы можете архивировать и разархивировать коллекции при условии, что содержимое всех массивов соответствует `NSCoding` протоколу. Пример этого метода показан в листинге 3.

Листинг 3 истинно полное копирование

```
NSArray* trueDeepCopyArray = [NSKeyedUnarchiver unarchiveObjectWithData:
    [NSKeyedArchiver archivedDataWithRootObject: oldArray]];
```

При копировании коллекции, изменчивость данной коллекции и объекты в ней могут быть затронуты. Каждый метод копирования имеет несколько различных воздействий на изменчивость объектов в коллекции произвольной глубины:

- `copyWithZone:` делает на уровне поверхностно неизменном. Все более глубокие уровни имеют изменчивость как ранее.
- `initWithArray:copyItems:` с `NO` в качестве второго аргумента дает поверхностный уровень изменчивости выделяемого класса. Все более глубокие уровни имеют изменчивость как ранее.
- `initWithArray:copyItems:` с `YES` в качестве второго параметра дает поверхностный уровень изменчивости выделяемого класса. Следующий уровень является неизменным, и все более глубокие уровни имеют изменчивость как ранее.
- Архивирование и распаковка коллекции оставляет изменчивость на всех уровнях, какой она была раньше.

10.8 Перечисление: обход элементов коллекции

Сосоа содержит три основных способа перечисления содержимого коллекции. К ним относятся быстрое перечисление и блочное перечисление. Существует также `NSEnumerator` класс, хотя в целом был замещен быстрым перечислением [3].

Быстрое перечисление является предпочтительным методом перечисления содержимого коллекции, поскольку оно обеспечивает следующие преимущества:

- Перечисление является более эффективным, чем использование `NSEnumerator` напрямую.
- Синтаксис краток
- Перечислитель вызывает исключение, если вы измените коллекции при перечислении.
- Вы можете выполнять несколько перечислений одновременно.

Поведение для быстрого перечисления немного отличается в зависимости от типа коллекции. Массивы и наборы перечисляют их содержимое, а словари перечисляют свои ключи. `NSIndexSet` и `NSIndexPath` не поддерживают быстрое перечисление. Вы можете использовать быстрое перечисление с коллекцией объектов, как показано в листинге 1.

Листинг 1. Использование быстрого перечисления со словарем.

```
for (NSString *element in someArray) {
    NSLog(@"element: %@", element);
}

NSString *key;
for(key in someDictionary){
    NSLog(@"Key: %@, Value %@", key, [someDictionary objectForKey: key]);
}
```

```
}
```

NSArray, NSDictionary и NSSet разрешают перечисление их содержимого с помощью блоков. Для перечисления с блоком, вызовите соответствующий метод и укажите блок для использования. В листинге 2 показано блочное перечисление объектов NSArray.

Листинг 2: перечисление массива на основе блока.

```
NSArray *anArray=[NSArray arrayWithObjects:@"A", @"B", @"D",@"M",nil];
NSString *string=@"c";

[anArray enumerateObjectsUsingBlock:^(id obj, NSUInteger index, BOOL *stop){
    if([obj localizedCaseInsensitiveCompare:string] == NSOrderedSame){
        NSLog(@"Object Found: %@ at index: %i",obj, index);
        *stop=YES;
    }
}];
```

Для объекта NSSet, вы можете использовать код подобный, показанному в листинге 3.

Листинг 3: перечисления набора на основе блока.

```
NSSet *aSet=[NSSet setWithObjects: @"X", @"Y", @"Z", @"Pi", nil];
NSString *aString=@"z";

[aSet enumerateObjectsUsingBlock:^(id obj, BOOL *stop){
    if([obj localizedCaseInsensitiveCompare:aString]==NSOrderedSame){
        NSLog(@"Object Found: %@", obj);
        *stop=YES;
    }
}];
```

Для перечисления NSArray, параметр *index* полезен для одновременного перечисления. Без этого параметра, единственный способ получить доступ к индексу был бы использованием метода `indexOfObject:`, который является неэффективным. *stop* параметр важен для производительности, так как он позволяет остановить перечисление раньше, на основе некоторого условия, определяемого в пределах блока. Методы перечисления на основе блока в других коллекциях, немного отличаются по названию.

NSEnumerator простой абстрактный класс, подклассы перечисляют коллекции других объектов. Коллекции объектов, таких как массивы, множества, словари, обеспечивают специальные NSEnumerator объекты, с которыми NSEnumerator перечисляет их содержимое. Вы отправляете `nextObject`, чтобы вновь созданный объект NSEnumerator возвращал следующий объект в оригинальной коллекции. Когда коллекция будет исчерпана, то возвращается `nil`. Вы не можете "сбросить" перечислитель после того, как он исчерпал свои коллекции. Для перечисления коллекции, вы должны создать новый перечислитель.

Классы коллекций, такие как NSArray, NSSet и NSDictionary включают методы, которые возвращают Перечислитель соответствующего типа коллекции. Например, NSArray имеет два метода, которые возвращают объект NSEnumerator: `objectEnumerator` и `reverseObjectEnumerator`. Класс NSDictionary также имеет два метода, которые возвращают объект NSEnumerator: `keyEnumerator` и `objectEnumerator`. Эти методы позволяют перечислить содержимое NSDictionary объект по ключу или по значению, соответственно.

Опасно удалять, заменять или добавлять элементы изменяемых коллекций при их перечислении. Если необходимости изменить коллекцию при перечислении, можно либо сделать копию коллекции и перечислять используя копию или сохранить информацию необходимую Вам во время перечисления и применить изменения впоследствии. Вторым вариантом показан в листинге 4.

Листинг 4. Перечисление словаря и удаление объектов.

```
NSMutableDictionary *myMutableDictionary = ... ;
NSMutableArray *keysToDeleteArray =
    [NSMutableArray arrayWithCapacity:[myMutableDictionary count]];
NSString *aKey;
NSEnumerator *keyEnumerator = [myMutableDictionary keyEnumerator];
while (aKey = [keyEnumerator nextObject])
{
    if ( /* Проверка критериев для ключа или значения */ ) {
        [keysToDeleteArray addObject:aKey];
    }
}
[myMutableDictionary removeObjectForKey:aKey];
```

10.9 Параметры указателя на функцию

Указатель классов коллекций (NSPointerArray, NSMapTable и NSHashTable) позволяет дополнительно настроить коллекцию, адаптировать ее к памяти и хранению данных. Опции, указанные для NSPointerFunctionsOptions предоставляют удобный интерфейс для настройки того, как будет коллекция управлять указателями содержащимися в ней [3].

iOS Примечание: Указатель классов коллекций и соответствующие параметры недоступны на iOS.

Указатели коллекции настраиваются с помощью параметров в трех различных категориях: параметры памяти, персональные параметры, поведение копирования. Не все варианты комбинаций памяти, персональные и копирования допустимы.

Параметры памяти определяют поведение при добавлении в коллекцию, удалении из коллекции или копировании. Некоторые из наиболее распространенных вариантов включают в себя:

- NSPointerFunctionsStrongMemory, который используется для коллекции, которая выражает право собственности на свое содержание. В среде сборки мусора это используется для коллекции, которая содержит сильные ссылки на свое содержание.
- NSPointerFunctionsZeroingWeakMemory, часто используется для коллекции, которая содержит слабые ссылки на объекты в среде сборки мусора. Если сбор мусора не включен, коллекция будет содержать, не сохраненные указатели объектов.
- NSPointerFunctionsOpaqueMemory, который используется в тех случаях, когда право собственности на содержимое полностью управляется за пределами коллекции. Он часто используется для коллекций, которые содержат указатели на примитивные типы, такие как целые или С-строки.

Персональные параметры определяют тип указателей, хранящихся в коллекции, такие как указатели на объекты. Они также указывают, что происходит, при хэшировании и сравнении. Некоторые из наиболее распространенных вариантов включают в себя:

- NSPointerFunctionsObjectPersonality, который используется для коллекции, которая содержит объекты и использует isEqual: для определения равенства.
- NSPointerFunctionsObjectPointerPersonality, который часто используется для коллекции, которая содержит объекты и использует прямое сравнение с целью определения равенства.
- NSPointerFunctionsOpaquePersonality, который часто используется для коллекции, которая содержит указатели на примитивные типы, такие как целые или С-строки.

Опции копирования указывают, следует ли коллекции скопировать элементы вошедшие в коллекцию. Если опция NSPointerFunctionsCopyIn определена, коллекция копирует вошедшие элементы, в противном случае нет.

Если вам нужно больше, чем позволяет настройка `NSPointerFunctionsOptions`, вы можете использовать класс `NSPointerFunctions` для определения пользовательских функций для таких операций, как распределение памяти, хеширование и проверка равенства. Например, если у вас есть коллекция структур, вам нужно будет указать размер структуры.

Если вы хотите настроить указатель коллекцию на хранение объектов, есть несколько вариантов. Для объектов, только два персональных варианта имеет смысл:

- `NSPointerFunctionsObjectPersonality`, выбор объекта по умолчанию, используется метод `isEqual:` для определения равенства.
- `NSPointerFunctionsObjectPointerPersonality`, также является жизнеспособным вариантом, использует указатель равенства для определения равенства.

Вы также можете использовать любую сильную (`strong`) память или обнуляемую слабую (`weak`) память. При сборе мусора, сильная память не утилизируется в коллекторе, но объекты проводятся только в слабой памяти. В среде, без сбора мусора, выбор опции `NSPointerFunctionsStrongMemory` обязывает вашу коллекцию использовать `retain` и `release`. Если вы решите использовать сильную память, вы можете выбрать, хотите ли вы иметь объекты, которые будут скопированы, при добавлении в коллекцию.

Например, если вы хотите провести коллекцию со слабыми ссылками на объекты и использовать `isEqual:` для определения равенства, можно указать параметры следующим образом:

```
NSPointerFunctionsOptions collectionOptions=  
NSPointerFunctionsObjectPersonality  
    | NSPointerFunctionsZeroingWeakMemory;
```

После указания параметров, `collectionOptions` затем может быть передан в коллекцию во время инициализации.

Если вы хотите настроить указатель коллекции на содержание произвольных (`nonobject`) указателей, у вас есть гибкие возможности по настройке коллекции в зависимости от типа указателя. Для большей гибкости, вы можете выбрать `NSPointerFunctionsOpaquePersonality`, которая позволяет содержать указатели для самых примитивных типов. Вы также можете выбрать один из вариантов конкретных типов:

- `NSPointerFunctionsIntegerPersonality` держать указатели на целые.
- `NSPointerFunctionsStructPersonality` держать указатели на структуры. Если вы укажете эту опцию, вы должны установить `sizeFunction` свойство у `NSPointerFunctions` объекта, который вы используете.
- `NSPointerFunctionsCStringPersonality` держать указатели на C-строки.

Вы должны как правило, использовать `NSPointerFunctionsOpaqueMemory` при работе с любыми указателями, потому что она совместима со всеми персональными вариантами. Если вы хотите, вы можете использовать `NSPointerFunctionsMallocMemory` или `NSPointerFunctionsMachVirtualMemory` с непрозрачной, C-строкой, и персональной структурой, хотя это обычно не рекомендуется.

Только произвольные конфигурации указателей, поддерживают поведение копирования C-строки и персональной структуры при использовании как `malloc` или `mach` виртуальной памяти.

Если вы хотите держать коллекцию с произвольными указателями с использованием непрозрачных памяти, вы можете указать параметры следующим образом:

```
NSPointerFunctionsOptions collectionOptions=  
NSPointerFunctionsOpaquePersonality  
    | NSPointerFunctionsOpaqueMemory;
```

После указания параметров, `collectionOptions` затем может быть передан в коллекцию во время инициализации.

11 Паттерны iOS

11.1 Паттерн Prototype

Прототип – один из самых простых паттернов, который позволяет нам получить точную копию необходимого объекта. То есть использовать как прототип для нового объекта.

Когда использовать:

1. У нас есть семейство схожих объектов, разница между которыми только в состоянии их полей.
2. Чтобы создать объект вам надо пройти через огонь, воду и медные трубы. Особенно если этот объект состоит из еще одной кучи объектов, многие из которых для заполнения требуют подгрузку данных из базы, веб сервисов и тому подобных источников. Часто, легче скопировать объект и поменять несколько полей.
3. Да и в принципе, нам особо и не важно, как создается объект.
4. Нам просто лень писать иерархию фабрик (читай дальше), которые будут инкапсулировать всю противную работу создания объекта.

Да, и есть еще частое заблуждение (вероятнее всего из названия) – прототип – это архитип, которые никогда не должен использоваться, и служит только для создания себе подобных объектов. Хотя, прототип, как архитип – тоже достаточно популярный кейс. Собственно, нам ничего не мешает делать прототипом любой объект, который у нас в подчинении.

Поверхностное и глубокое копирование.

Тут особенно разницы с .NET нет. Есть указатель, есть значение в куче.

Поверхностное копирование – это просто создание нового указателя на те же самые байты в куче. То есть, в результате мы можем получить два объекта, которые указывают на одно и то же значение.

Например, создадим объект:

```
@interface Person : NSObject <NSCopying>
@property (nonatomic,weak) NSString *name;
@property (nonatomic,weak) NSString *surname;
@property (nonatomic,weak) NSString *age;
@end
```

А теперь давайте просто создадим два объекта и посмотрим, что же получится:

```
Person *firstPerson = [[Person alloc] init]; firstPerson.name = @"Dima"; firstPerson.surname = @"Surname";
Person *secondPerson = firstPerson;
NSLog(@"First Person name = %@ and surname = %@", firstPerson.name, firstPerson.surname);
secondPerson.name = @"Roma";
NSLog(@"Second Person name = %@ and surname = %@", secondPerson.name, secondPerson.surname);
NSLog(@"First Person name = %@ and surname = %@", firstPerson.name, firstPerson.surname);
```

Как видим лог достаточно ожидаемый:

```
2013-01-21 01:31:20.986 PrototypePattern[1961:11303] First Person name = Dima 2013-01-21 01:31:20.987 PrototypePattern[1961:11303] Second Person name = Roma 2013-01-21 01:31:20.987 PrototypePattern[1961:11303] First Person name = Roma
```

Заметьте, что хоть и меняли мы имя для secondPerson, но и у firstPerson имя поменялось. Просто потому что мы создали два указателя на один и тот же объект.

Для таких задач, стоит использовать глубокое копирование, которое в Objective-C сделано в принципе очень похоже, как и в .NET. Для этого надо реализовать протокол NSCopying, и перегрузить:

```
-(id) copyWithZone:(NSZone *) zone;
```

И да, не стоит переживать, что мы не реализуем метод “copy”. Он уже есть у класса NSObject. Если вызвать этот метод, и не реализовать copyWithZone – то мы получим ошибку типа NSInvalidArgumentException.

Потому интерфейс нашего объекта теперь будет выглядеть следующим образом:

```
@interface Person : NSObject <NSCopying>
@property (nonatomic,weak) NSString *name;
@property (nonatomic,weak) NSString *surname;
@property (nonatomic,weak) NSString *age;
-(id) copyWithZone:(NSZone *) zone;
@end
```

а сама реализация:

```
@implementation Person
-(id) copyWithZone:(NSZone *) zone
{
    Person *copy = [[self class] allocWithZone:zone]; copy.name =
    self.name;
    copy.age = self.age; copy.surname = self.surname; return copy;
}
@end
```

Теперь немного изменим код нашего тестового приложения:

```
Person *firstPerson = [[Person alloc] init]; firstPerson.name
= @"Dima"; firstPerson.surname = @"Surname";
Person *secondPerson = firstPerson.copy;
NSLog(@"First Person name = %@ and surname = %@",
firstPerson.name, firstPerson.surname);
secondPerson.name = @"Roma";
NSLog(@"Second Person name = %@ and surname = %@",
secondPerson.name, secondPerson.surname);
NSLog(@"First Person name = %@ and surname = %@",
firstPerson.name, firstPerson.surname);
```

Ну и естественно лог:

2013-01-21 01:48:35.538 PrototypePattern[2090:11303] First Person name = Dima and surname = Surname

2013-01-21 01:48:35.539 PrototypePattern[2090:11303] Second Person name = Roma and surname = Surname

01:48:35.540 PrototypePattern[2090:11303] First Person name = Dima and surname = Surname

Как видим, в результате получились два независимых объекта, второй сделан по подобию первого.

11.2 Паттерн Factory Method

Еще один порождающий паттерн, довольно прост и популярен. Паттерн позволяет переложить создание специфических объектов, на наследников родительского класса, потому можно манипулировать объектами на более высоком уровне, не заморачиваясь объект какого класса будет создан. Частенько этот паттерн называют виртуальный конструктор, что, более выражает его предназначение.

Когда использовать:

1. Мы не до конца уверены, какого типа объект нам необходим.
2. Мы хотим, чтобы не родительский объект решал, какой тип создавать, а его наследники.

Почему хорошо использовать:

Объекты, созданные фабричным методом – похожи, потому как у них один и тот же родительский объект. Потому, если локализовать создание таких объектов, то можно добавлять новые типы, не меняя при этом код, который использует фабричный метод.

Пример:

Давайте представим, что мы такой неправильный магазин, в котором тип товара оценивается по цене. На данный момент товар есть двух типов – «Игрушки» и «Одежда».

В чеке мы получаем только цены, и нам надо сохранить объекты, которые куплены.

Для начала создадим клас Product. Его реализация нас особо не интересует, хотя он может содержать в себе общие для разных типов товаров методы (сделано для примера, мы их особо не используем):

```
@interface Product : NSObject
@property(n nonatomic) int *price;
@property(n nonatomic, strong) NSString *name;
-(NSInteger *)getTotalPrice:(int) sum;
-(void) saveObject;
@end

@implementation Product
-(NSInteger *) getTotalPrice:(int) sum
{
return self.price + sum;}
-(void) saveObject
{
NSLog(@"I am saving an object in to product database");
}
@end
```

Теперь создадим две реализации этого интерфейса.

Игрушка:

```
@interface Toy : Product
@end

@implementation Toy
-(void) saveObject
{
NSLog(@"Saving object into Toys database");
}
@end
```

И одежда:

```
@interface Dress : Product
@end

@implementation Dress
-(void) saveObject
{
NSLog(@"Saving object into Dress database");
}
@end
```

Теперь мы практически подошли вплотную к нашему паттерну. Собственно, теперь надо создать метод, который будет определять по цене, что же за продукт у нас в чеке, и создавать объект необходимого типа.


```

@interface ProductGenerator : NSObject
-(Product *) getProduct:(int)price;
@end
@implementation ProductGenerator
-(Product *) getProduct:(int)price
{
if ( price > 0 && price < 100 )
{
Toy *p = [[Toy alloc] init];
return p;
}

if (price >= 100)
{
Dress *p = [[Dress alloc] init];
return p;
}
return nil;
}
@end

```

Вот собственно и все. Теперь просто создадим метод, который будет считать и записывать расходы:

```

-(void) saveExpenses:(int)aPrice
{
ProductGenerator *pd = [[ProductGenerator alloc] init];
Product *expense = [pd getProduct:aPrice];
[expense saveObject];}

```

Попробуем!

```

[self saveExpenses:50]; [self saveExpenses:56]; [self
saveExpenses:79]; [self saveExpenses:100]; [self
saveExpenses:123]; [self saveExpenses:51];
Лог:

```

2013-01-23 23:27:54.223 FactoryMethodPattern[8833:11303] Saving object into Toys database

2013-01-23 23:27:54.226 FactoryMethodPattern[8833:11303] Saving object into Toys database

2013-01-23 23:27:54.226 FactoryMethodPattern[8833:11303] Saving object into Toys database

2013-01-23 23:27:54.227 FactoryMethodPattern[8833:11303] Saving object into Dress database

2013-01-23 23:27:54.227 FactoryMethodPattern[8833:11303] Saving object into Dress database

23:27:54.228 FactoryMethodPattern[8833:11303] Saving object into Toys database

11.3 Паттерн Abstract Factory

Абстрактная фабрика – еще один очень популярный паттерн, который, как и в названии, так и в реализации слегка похож на фабричный метод.

Итак, что же делает абстрактная фабрика. Абстрактная фабрика дает простой интерфейс для создания объектов, которые принадлежат к тому или иному семейству объектов.

Отличия от фабричного метода:

1. Фабричный метод порождает объекты одного и того же типа, абстрактная

фабрика может создавать независимые объекты.

2. Чтобы добавить новый тип объекта – надо поменять интерфейс фабрики, в фабричном методе легко просто поменять внутренности метода, который ответственный за порождение объектов.

Давайте представим ситуацию: у нас есть две фабрики по производству iPhone и iPad. Одна оригинальная, компании Apple, другая – китайская. И вот, мы хотим производить эти товары: если в страны третьего мира – то товар из Китая, в другие страны – товар любезно предоставлен компанией Apple.

Итак, пускай у нас есть фабрика, которая умеет производить и айпады и айфоны:

```
@interface iPhoneFactory : NSObject
-(GenericiPhone *) getiPhone;
-(GenericIPad *) getIPad;
@end
```

Естественно, нам необходимо реализовать продукты, которые фабрика будет производить:

```
@interface GenericIPad : NSObject
@property(nonatomic,weak) NSString *osName;
@property(nonatomic, weak) NSString *productName;
@property(nonatomic, strong) NSNumber *screenSize;
@end
//-----
@interface GenericiPhone : NSObject
@property (nonatomic, weak) NSString *osName;
@property (nonatomic, weak) NSString *productName;
@end
```

Но, продукты немного отличаются. Пускай у нас есть два типа продуктов, оригинальные Apple и продукты которые произведены в Китае:

```
@interface AppleiPhone : GenericiPhone
@end
@implementation AppleiPhone
-(id) init
{
self = [super init];
self.productName = @"iPhone";
self.osName = @"iOS";
return self;
}
@end

//-----
//Айпад
@interface AppleIPad : GenericIPad
@end
@implementation AppleIPad
-(id) init
{
self = [super init]; self.productName = @"IPad"; self.osName =
@"iOS";
self.screenSize = [[NSNumber alloc] initWithFloat:7.7f];
return self;
}
@end
```

Китайские товары:

```

@interface ChinaPad : GenericIPad
@end
@implementation ChinaPad
-(id) init
{
self = [super init];
self.osName = @"Windows CE";
self.productName = @"Buan Que Ipado Killa";
self.screenSize = [[NSNumber alloc] initWithFloat:12.5f];
return self;
}
@end
//-----
@interface ChinaPhone : GenericIPhone
@end
@implementation ChinaPhone
-(id) init
{
self = [super init];
self.osName = @"Android";
self.productName = @"Chi Huan Hua Phone";

return self;
}
@end

```

Разные телефоны, конечно же, производятся на различных фабриках, потому мы просто обязаны их создать! Приблизительно так должны выглядеть фабрика Apple:

```

@interface AppleFactory : IPhoneFactory
@end
@implementation AppleFactory
-(GenericIPhone *) getIPhone
{
AppleIPhone *iphone = [[AppleIPhone alloc] init];
return iphone;
}
-(GenericIPad *) getIPad
{
AppleIPad *ipad = [[AppleIPad alloc] init];
return ipad;
}
@end

```

Конечно, в Китае тоже есть своя фабрика:

```

@interface ChinaFactory : IPhoneFactory
@end
@implementation ChinaFactory
-(GenericIPad *) getIPad
{
ChinaPad *pad = [[ChinaPad alloc] init];
return pad;
}
-(GenericIPhone *) getIPhone
{
ChinaPhone *phone = [[ChinaPhone alloc] init];
return phone;
}

```

```

}
@end

```

Как видим, фабрики одинаковые, а вот дивайсы у них получатся разные. Вот собственно и все, мы приготовили все, что надо для демонстрации! Теперь, давайте напишем небольшой метод, который будет возвращать нам фабрику, которую мы хотим (кстати, тут будет фабричный метод):

```

-(iPhoneFactory *) getFactory
{
    if (!_isThirdWorld)
        return [[ChinaFactory alloc] init];

    return [[AppleFactory alloc] init];
}

```

Теперь, создадим несколько телефонов:

```

_isThirdWorld = false;
iPhoneFactory *factory = self.getFactory; GenericIPad *ipad =
factory.getIPad; GenericIPhone *iphone = factory.getIPhone;
NSLog(@"IPad named = %@, osname = %@, screensize = %@",
ipad.productName, ipad.osName, ipad.screenSize.stringValue);
NSLog(@"IPhone named = %@, osname = %@", iphone.productName,
iphone.osName);

```

Лог будет выглядеть следующим образом:

```

2013-01-26 20:00:55.663 AbstractFactory[13093:11303] IPad named = Buan Que Ipado
Killa, osname = Windows CE, screensize = 12.5
2013-01-26 20:00:55.665 AbstractFactory [13093:11303] IPhone named = Chi Huan
Hua Phone, osname = Android

```

Теперь, просто поменяем значение переменной `_isThirdWorld` на `false`, и лог будет совсем другой:

```

2013-01-26 20:02:21.745 AbstractFactory [13115:11303] IPad named = IPad, osname
= iOS, screensize = 7.7
21.747 AbstractFactory [13115:11303] IPhone named = IPhone, osname = iOS

```

11.4 Паттерн Builder

Представьте, что у нас есть фабрика. В отличии от фабрики из предыдущего раздела, она умеет создавать только телефоны на базе андроида, и еще при этом различной конфигурации. То есть, есть один объект, но при этом его состояние может быть совершенно разным, а еще представьте, если его очень трудно создавать, и во время создания этого объекта еще и создается много дочерних объектов. Именно в такие моменты, нам очень помогает такой паттерн, как строитель.

Когда использовать:

1. При создании сложного объекта.
2. Если процесс создания объекта не тривиальный — к примеру получение данных из базы и манипуляция ими.

Сам паттерн состоит из двух компонент — `Bulilder` и `Director`. `Builder` занимается именно построение объекта, а `Director` знает какой использовать `Builder`, чтобы выдать необходимый продукт. Приступим.

Пусть у нас есть телефон, который обладает следующими свойствами:

```

@interface AndroidPhone : NSObject
@property (nonatomic, weak) NSString *osVersion;
@property (nonatomic, weak) NSString *name;

```

```

@property (nonatomic, weak) NSString *cpuCodeName;
@property (nonatomic, strong) NSNumber *RAMsize;
@property (nonatomic, strong) NSNumber *osVersionCode;
@property (nonatomic, weak) NSString *launcher;
@end

```

Давайте создадим дженерик строителя, от которого будут наследоваться конкретные строители:

```

@interface BPAndroidPhoneBuilder : NSObject
@property (nonatomic, strong) AndroidPhone* _phone;
-(void) setOSVersion;
-(void) setName;
-(void) setCPUCodeName;
-(void) setRAMSize;
-(void) setOSVersionCode;
-(void) setLauncher;
-(AndroidPhone *) getPhone;
@end
//---
@implementation BPAndroidPhoneBuilder
-(id) init
{
self = [super init];
self._phone = [[AndroidPhone alloc] init];
return self;
}
-(AndroidPhone *) getPhone
{
return self._phone;
}
@end

```

Ну а теперь напомним код для конкретных строителей. Наприер, так бы выглядел строитель для дешевого телефона:

```

@interface LowPricePhoneBuilder : BPAndroidPhoneBuilder
@end
//----
@implementation LowPricePhoneBuilder
-(void) setOSVersion
{
self._phone.osVersion = @"Android 2.3";
}
-(void) setName
{
self._phone.name = @"Low price phone!";
}
-(void) setCPUCodeName
{
self._phone.cpuCodeName = @"Some shitty CPU";
}
-(void) setRAMSize
{
self._phone.RAMsize = [[NSNumber alloc] initWithInt:256];
}
-(void) setOSVersionCode
{
self._phone.osVersionCode = [[NSNumber alloc] initWithFloat:3.0f];
}

```

```

}
-(void) setLauncher
{
self._phone.launcher = @"Hia Tsung!";
}
@end

```

И так будет выглядеть строительство дорогого телефона:

```

@interface HighPricePhoneBuilder : BPAndroidPhoneBuilder
@end
//----
@implementation HighPricePhoneBuilder
-(void) setOSVersion
{
self._phone.osVersion = @"Android 4.1";
}
-(void) setName
{
self._phone.name = @"High price phone!";
}
-(void) setCPUCodeName
{
self._phone.cpuCodeName = @"Some shitty but expensive CPU";
}
-(void) setRAMSize
{
self._phone.RAMsize = [[NSNumber alloc] initWithInt:1024];
}
-(void) setOSVersionCode
{
self._phone.osVersionCode = [[NSNumber alloc] initWithFloat:4.1f];
}
-(void) setLauncher
{
self._phone.launcher = @"Samsung Launcher";
}
@end

```

Кто-то же должен использовать строителей, поэтому давайте создадим объект, который будет с помощью строителей создавать дешевые или дорогие телефоны:

```

@interface FactorySalesMan : NSObject
@property (nonatomic, strong) BPAndroidPhoneBuilder *_builder;
-(void) setBulider:(BPAndroidPhoneBuilder *)aBuilder;
-(AndroidPhone *) getPhone;
-(void) constructPhone;
@end
//----
@implementation FactorySalesMan
-(void)setBulider:(BPAndroidPhoneBuilder *)aBuilder
{
self._builder = aBuilder;
}
-(AndroidPhone *) getPhone
{
return self._builder.getPhone;
}

```

```

}
-(void) constructPhone
{
[self._builder setOSVersion]; [self._builder setName];
[self._builder setCPUCodeName]; [self._builder setRAMSize];
[self._builder setOSVersionCode]; [self._builder setLauncher];
}
@end

```

А теперь тест и код:

```

LowPricePhoneBuilder *_cheapPhoneBuilder =
[[LowPricePhoneBuilder alloc] init]; HighPricePhoneBuilder
*_expensivePhoneBuilder =
[[HighPricePhoneBuilder alloc] init];
FactorySalesMan *_salesMan = [[FactorySalesMan alloc] init];
[_salesMan setBuilder:_cheapPhoneBuilder];
[_salesMan constructPhone];
AndroidPhone *_phone = [_salesMan getPhone];
NSLog(@"Phone Name = %@, osVersion = %@, cpu code name = %@,
ram size = %@, os
version code = %@, launcher = %@",
_phone.name, _phone.osVersion, _phone.cpuCodeName,
_phone.RAMsize,
_phone.osVersionCode, _phone.launcher);
[_salesMan setBuilder:_expensivePhoneBuilder]; [_salesMan
constructPhone];
_phone = [_salesMan getPhone];
NSLog(@"Phone Name = %@, osVersion = %@, cpu code name = %@,
ram size = %@, os version code = %@, launcher = %@",
_phone.name, _phone.osVersion, _phone.cpuCodeName,
_phone.RAMsize,
_phone.osVersionCode, _phone.launcher);

```

Как видим, мы создали различных строителей, и, сообщив директору (FactorySalesMan) какой строитель мы хотим использовать, мы получаем тот дивайс, который нам необходим:

Традиционный лог:

```

2013-01-28 00:38:51.863 BuilderPattern[708:11303] Phone Name = Low price phone!,
osVersion = Android 2.3, cpu code name = Some shitty CPU, ram size = 256, os version code =
3, launcher = Hia Tsung!
00:38:51.867 BuilderPattern[708:11303] Phone Name = High price phone!, osVersion =
Android 4.1, cpu code name = Some shitty but expensive CPU, ram size = 1024, os version code
= 4.1, launcher = Samsung Launcher

```

11.5 Паттерн Singleton

Реализация паттерна, которая будет приведена тут, подразумевает использование GDC и ARC.

Кто мог подумать, что Singleton такой не самый просто паттерн в iOS? Вернее, что есть столько версий. Собственно, в .NET, всё точно также, но там в основном были просто апдейты к самой простой версии паттерна.

Итак, начнем с простого – с описания.

Singleton - это такой объект, который существует в системе только в единственном экземпляре. Очень часто используется для хранения каких-то глобальных переменных, например настроек приложения.

Итак, как и все в Obj-C начнем мы естественно с создания интерфейса:

```
@interface SingletonObject : NSObject
@property (nonatomic, weak) NSString *tempProperty;
+(SingletonObject *) singleton;
@end
```

Как видим, обычный объект с одним свойством и класс методом. Естественно, просто от интерфейса мы не получим всего чего ожидаем:

```
@implementation SingletonObject
+(SingletonObject *) singleton
{
static SingletonObject *singletonObject = nil;
static dispatch_once_t onceToken;
dispatch_once(&onceToken, ^{ singletonObject = [[self alloc]
init];
});
return singletonObject;
}
@end
```

Собственно вот и все. iOS сам за нас позаботится о том, чтобы создан был только один экземпляр нашего объекта. Тут стоит сделать шаг назад и описать проблему, которая является головной болью любого кто более-менее близко работал с потоками:

1. Представьте, что есть два потока.
2. Каждый из них, одновременно создает singleton. Вроде бы и должен создаваться только один объект, но потому что все происходит в один момент – бывают случаи когда создается два объекта.

«Но ведь можно сделать проверку на nil!» – скажете Вы. А теперь, представьте более сложную ситуацию: объекта singleton не существует. Два потока хотят его создать одновременно:

1. Поток 1 делает проверку или объект существует. Видит что его нет, и проходит этап проверки.
2. Поток 2 делает проверку на существование объекта, и хотя и поток 1 проверку УЖЕ прошел, но объект ЕЩЕ не существует.

Для решения таких проблем, в .NET использовали locks – блокирование кода, для других потоков, пока он выполняется в каком – либо потоке. Собственно dispatch_once делает тоже самое – он просто синхронный. Потому, ни один поток не может зайти в этот код, пока он занят.

Собственно, без GCD такое создать тоже можно, тогда наш код бы выглядел следующим образом:

```
+(SingletonObject *) singleton
{
static SingletonObject *singletonObject = nil;
@synchronized(self)
{
if (singletonObject == nil )
{
singletonObject = [[self alloc] init];
}
}

return singletonObject;
}
```

Получится тоже самое. Единственное, что dispatch_once() по документации

быстрее и семантически более правильный.

Можно вообще создать макрос:

```
#define DEFINE_SHARED_INSTANCE_USING_BLOCK(block) \ static  
dispatch_once_t pred = 0; \  
    strong static id _sharedObject = nil; \ dispatch_once(&pred, \  
    ^{ \  
        _sharedObject = block(); \  
    }); \  
return _sharedObject; \
```

Тогда сама реализация создания объекта будет выглядеть следующим образом:

```
+(SingletonObject *) singleton  
{  
    DEFINE_SHARED_INSTANCE_USING_BLOCK(^{  
        return [[self alloc] init];  
    });  
}
```

Ну, а использование простое:

```
[[SingletonObject singleton] setTempProperty:@"Hello 2 You!"];  
NSLog(@"%@", [[SingletonObject singleton] tempProperty]);
```

11.6 Паттерн Adapter

Тяжело найти более красочно описание паттерна «Адаптер», чем пример из жизни каждого, кто покупал технику из США. В США розетка с квадратными отверстиями, в Европе с круглыми, а в некоторых странах вообще с треугольными. Следовательно – поэтому вилки на зарядных устройствах, и других устройствах питания тоже различные.

Представьте, что Вы едете в командировку в США. У Вас есть, допустим, ноутбук купленный в Европе – следовательно вилка на проводе от блока питания имеет круглые окончания. Что делать? Покупать зарядку для американского типа розетки? А когда вы вернетесь домой – она будет лежать у Вас мертвым грузом?

Потому, вероятнее всего, Вы приобретете один из адаптеров, которые надеваются на вилку, и которая позволяет Вам использовать старую зарядку и заряжаться от совершенно другой розетки.

Так и с «Адаптером» – он конвертирует интерфейс класса – на такой, который ожидается.

Сам паттерн состоит из трех частей: «Цели» (target), «Адаптера» (adapter), и «Адаплируемого» (adaptee).

В нашей с вами проблеме:

1. Target – ноутбук со старой зарядкой
2. Adapter – переходник.
3. Adaptee – розетка с квадратными отверстиями.

Имплементации паттерна «Адаптер» в Objective-C может быть две. Итак, первая – это простенькая имплементация. Пускай у нас есть объект Bird, который реализует протокол BirdProtocol:

```
@protocol BirdProtocol  
-(void) sing;  
-(void) fly;  
@end  
  
@interface Bird : NSObject <BirdProtocol>  
@end  
  
//реализация  
@implementation Bird
```

```

-(void) sing
{
NSLog(@"Tew-tew-tew");
}
-(void) fly
{
NSLog(@"OMG! I am flying!");
}
@end

```

И пускай у нас есть объект Raven, у которого есть свой интерфейс:

```

@interface Raven : NSObject
-(void) flySearchAndDestroy;
-(void) voice;
@end
@implementation Raven
-(void) flySearchAndDestroy
{
NSLog(@"I am flying and seek for killing!");
}
-(void) voice
{
NSLog(@"Kaaaar-kaaaaar-kaaaaaaar!");
}
@end

```

Чтобы использовать Raven в методах, которые ждут птицу, стоит создать так называемый адаптер:

```

@interface RavenAdapter : NSObject <BirdProtocol>
{
@private Raven *_raven;
}
@property (nonatomic, strong) Raven *raven;
-(id) initWithRaven:(Raven*) adaptee;
@end
//ну и сама реализация. Как видим, мы завернули ворона в
обертку протоколом
BirdProtocol
@implementation RavenAdapter
@synthesize raven = _raven;
-(id) initWithRaven:(Raven*) adaptee
{
self = [super self];
_raven = adaptee;
return self;
}
-(void) sing
{
[_raven voice ];
}
-(void) fly
{
[_raven flySearchAndDestroy];
}
@end

```

Ну и конечно же тест:

```

-(void) makeTheBirdTest:(id<BirdProtocol>)aBird
{
[aBird fly]; [aBird sing];}
//и тестовый код:

Bird *simpleBird = [[Bird alloc] init]; Raven *simpleRaven =
[[Raven alloc] init];
RavenAdapter *ravenAdapter = [[RavenAdapter alloc]
initWithRaven:simpleRaven];

[self makeTheBirdTest:simpleBird]; [self
makeTheBirdTest:ravenAdapter];

```

Результат можно легко увидеть в логе:

```

2013-02-03 15:43:14.447 AdapterPattern[5985:11303] OMG! I am flying! 2013-02-03
15:43:14.449 AdapterPattern[5985:11303] Tew-tew-tew
2013-02-03 15:43:14.449 AdapterPattern[5985:11303] I am flying and seek for killing! 2013-02-
03 15:43:14.450 AdapterPattern[5985:11303] Kaaaaar-kaaaaaar-kaaaaaaar!

```

Теперь более сложная реализация, которая все еще зависит от протоколов, но уже использует делегацию. Вернемся к нашему ноутбуку и зарядке. Допустим, у нас есть базовый класс Charger:

```

@interface Charger : NSObject
-(void) charge;
@end
@implementation Charger
-(void) charge
{
NSLog(@"C'mon I am charging");
}
@end

```

И есть протокол для европейской зарядки:

```

@protocol EuropeanNotebookChargerDelegate
-(void) chargeNotebookRoundHoles:(Charger *)charger;
@end

```

Если сделать просто реализацию – то получится тоже самое, что и в прошлом примере. Потому, давайте добавим делегат:

```

@interface EuropeanNotebookCharger : Charger
<EuropeanNotebookChargerDelegate>
{
@private id<EuropeanNotebookChargerDelegate> _delegate;
}

@property (nonatomic, strong) id<EuropeanNotebookChargerDelegate> delegate;
@end

//реализация
@implementation EuropeanNotebookCharger

@synthesize delegate = _delegate;

-(id) init
{

```

```

self = [super self]; self.delegate = self; return self;
}
-(void) charge
{
[_delegate chargeNotebookRoundHoles:self]; [super charge];
}
-(void) chargeNotebookRoundHoles:(Charger *)charger
{
//and yeah you can do smth with charger.
NSLog(@"Charging with 220 and round holes!");
}

@end

```

Как видим, у нашего класса есть свойство, которое реализует тип `EuropeanNotebookChargerDelegate`. Так как, наш класс этот протокол реализует, поэтому он может присвоить себя свойству, когда происходит вызов:

```
[_delegate chargeNotebookRoundHoles:self];
```

Просто вызывается свой же метод. Вы увидите дальше, для чего это сделано. Теперь, давайте посмотрим, что из себя представляет американская зарядка:

```

@interface USANotebookCharger : NSObject
-(void) chargeNotebookRectHoles:(Charger *) charger;
@end
@implementation USANotebookCharger
-(void) chargeNotebookRectHoles:(Charger *) charger;
{
NSLog(@"Charge Notebook Rect Holes");
}
@end

```

Как видим, в американской зарядке совсем другой метод и мировоззрение. Давайте, создадим адаптер для зарядки:

```

@interface USANotebookEuropeanAdapter : Charger
<EuropeanNotebookChargerDelegate>
@property (nonatomic, strong) USANotebookCharger *usaCharger;
-(id) initWithUSANotebookCharger:(USANotebookCharger *)
charger;
-(void) charge;
@end
@implementation USANotebookEuropeanAdapter
-(id) initWithUSANotebookCharger:(USANotebookCharger *)
charger
{
self = [super init];
self.usaCharger = charger;
return self;
}
-(void) chargeNotebookRoundHoles:(Charger *) charger
{
[self.usaCharger chargeNotebookRectHoles:charger];
}
-(void) charge

```

```

{
EuropeanNotebookCharger          *euroCharge          =
[[EuropeanNotebookCharger alloc] init]; euroCharge.delegate =
self;
[euroCharge charge];
}
@end

```

Как видим, наш адаптер реализует интерфейс `EuropeanNotebookChargerDelegate` и его метод `chargeNotebookRoundHoles`. Поэтому, когда вызывается метод `charge` – на самом деле создается тип европейской зарядки, ей присваивается наш адаптер как делегат, и вызывается ее метод `charge`. Так как делегатом присвоен наш адаптер, при вызове метода `chargeNotebookRoundHoles`, будет вызван этот метод нашего адаптера, который в свою очередь вызывает метод американкой зарядки.

Давайте посмотрим тест код и вывод лога:

```

//сам метод
-(void) makeTheNotebookCharge:(Charger *) aCharger
{
[aCharger charge];
}

//тест код
EuropeanNotebookCharger          *euroCharger          =
[[EuropeanNotebookCharger          alloc]          init];          [self
makeTheNotebookCharge:euroCharger];
USANotebookCharger *charger = [[USANotebookCharger alloc]
init]; USANotebookEuropeanAdapter *adapter =
[[USANotebookEuropeanAdapter          alloc]
initWithUSANotebookCharger:charger];          [self
makeTheNotebookCharge:adapter];

```

Лог нам выведет:

```

2013-02-03 15:57:42.624 AdapterPattern[6179:11303] Charging with 220 and round holes!
2013-02-03 15:57:42.626 AdapterPattern[6179:11303] C'mon I am charging
15:57:42.626 AdapterPattern[6179:11303] Charge Notebook Rect Holes 2013-02-03
15:57:42.627 AdapterPattern[6179:11303] C'mon I am charging

```

11.7 Паттерн Bridge

Представьте себе, что у нас есть что-то однотипное, например, у нас есть телефон и куча наушников. Если бы у каждого телефона был свой разъем, то мы могли бы пользоваться только одним типом наушников. Собственно та же ситуация и с наушникам. Они могут выдавать различный звук, иметь различные дополнительные функции, но основная их цель – просто звучание. И хорошо, что во многих случаях штекер у них одинаковый. Собственно, «Мост» (Bridge) позволяет разделить абстракцию от реализации, так чтобы реализация в любой момент могла быть изменена, без изменения абстракции.

Когда использовать?

1. Вам совершенно не нужна связь между абстракцией и реализацией.
2. Собственно, как абстракцию, так и имплементацию могут наследовать независимо.
3. Вы не хотите, чтобы изменения в реализации имело влияния на клиентский код.

Давайте создадим теперь базовую абстракцию наушников:

```
@interface BaseHeadphones : NSObject
-(void) playSimpleSound;
-(void) playBassSound;
@end
```

И теперь два элемента – дорогие наушники и дешевые:

```
//Наушники обычные – китайские
@interface CheapHeadphones : BaseHeadphones
@end
@implementation CheapHeadphones
-(void) playSimpleSound
{
NSLog(@"beep - beep - bhhhrhrhrep");
}
-(void) playBassSound
{
NSLog(@"puf - puf - pufhrrr");
}
@end
//наушники дорогие, тоже китайские
@interface ExpensiveHeadphones : BaseHeadphones
@end
@implementation ExpensiveHeadphones
-(void) playSimpleSound
{
NSLog(@"Beep-Beep-Beep Taram - Rararam");
}
-(void) playBassSound
{
NSLog(@"Bam-Bam-Bam");
}
@end
```

И собственно плеер, через который мы будем слушать музыку:

```
@interface MusicPlayer : NSObject
@property (nonatomic, strong) BaseHeadphones *headPhones;
-(void) playMusic;
@end
@implementation MusicPlayer
-(void) playMusic
{
[self.headPhones playBassSound]; [self.headPhones
playBassSound]; [self.headPhones playSimpleSound];
[self.headPhones playSimpleSound];
}
@end
```

Как видите, одно из свойств нашего плеера – наушники. Их можно подменять в любой момент, так как свойство того же типа, от которого наши дешевые и дорогие наушники наследуются.

А теперь тест.

```
MusicPlayer *p = [[MusicPlayer alloc] init]; CheapHeadphones
*ch = [[CheapHeadphones alloc] init];
```

```
ExpensiveHeadphones *ep = [[ExpensiveHeadphones alloc] init];
p.headPhones = ch;
[p playMusic];
p.headPhones = ep; [p playMusic];
```

И конечно же лог:

```
2013-02-06 23:03:52.378 BridgePattern[3397:c07] puf – puf – pufhrrr 2013-02-06 23:03:52.379
BridgePattern[3397:c07] puf – puf – pufhrrr
2013-02-06 23:03:52.380 BridgePattern[3397:c07] beep – beep – bhhhrhrhrep
2013-02-06 23:03:52.380 BridgePattern[3397:c07] beep – beep – bhhhrhrhrep 2013-02-06
23:03:52.380 BridgePattern[3397:c07] Bam-Bam-Bam
2013-02-06 23:03:52.381 BridgePattern[3397:c07] Bam-Bam-Bam
23:03:52.381 BridgePattern[3397:c07] Beep-Beep-Beep Taram – Rararam 2013-02-06
23:03:52.381 BridgePattern[3397:c07] Beep-Beep-Beep Taram – Rararam
```

11.8 Паттерн Facade

Многие сложные системы состоят из множества компонент. Так же и в жизни, очень часто для совершения одного основного действия, мы должны выполнить много маленьких.

К примеру, чтобы пойти в кино нам надо:

Посмотреть расписание фильмов, выбрать фильм, посмотреть расписание сеансов, выбрать время.

Необходимо купить билет, для этого ввести номер карточки, секретный код, дожидаться снятия денег, распечатать билет.

Приехать в кинотеатр, запарковать машину, купить попкорн, найти места и смотреть.

И все это для того, чтобы просто посмотреть фильм, который нам, очень вероятно, не понравится. Или же возьмем пример сервис «Amazon» – покупка с одного клика – как много систем задействовано в операции покупки? И проверка Вашей карточки, и проверка Вашего адреса, проверка товара на складе, проверка или возможная доставка данного товара в данную точку мира... В результате очень много действий которые происходят всего по одному клику.

Для таких вот процессов был изобретен паттерн – «Фасад» (Facade), который предоставляет унифицированный интерфейс для большого числа интерфейсов системы, вследствие чего система станет гораздо проще в использовании.

Давайте, попробуем создать систему, которая нас переносит в другую точку мира с одного нажатия кнопки! С начала нам нужна система, которая проложит путь от нашего места пребывания в место назначения:

```
@interface Pathfinder : NSObject
-(void) findCurrentLocation;
-(void) findLocationToTravel:(NSString *) location;
-(void) makeARoute;
@end
@implementation Pathfinder
-(void) findCurrentLocation
{
NSLog(@"Finding your location. Hmmm, here you are!");
}
-(void) findLocationToTravel:(NSString *)location
{
NSLog(@"So you wanna travell to %@", location);
}
```

```

-(void) makeARoute
{
NSLog(@"Okay, to travell to this location we are using google
maps....");
//looking for path in google maps.
}
@end

```

Естественно нам необходима сама система заказа транспорта и собственно путешествия:

```

@interface TravellEngine : NSObject
-(void) findTransport;
-(void) orderTransport;
-(void) travel;
@end
@implementation TravellEngine
-(void) findTransport
{
NSLog(@"Okay, to travell there you will probabply need
dragon!Arghhhhh");
}
-(void) orderTransport
{
NSLog(@"Maaaam, can I order a dragon?... Yes... Yes, green
one... Yes, with fire!... No, not a dragon of death... Thank
you!");
}
-(void) travel
{
NSLog(@"Maaan, you are flying on dragon!");
}
@end

```

Ну и какие же путешествия без билетика:

```

@interface TicketPrinitingSystem : NSObject
-(void) createTicket;
-(void) printingTicket;
@end
@implementation TicketPrinitingSystem
-(void) createTicket
{
NSLog(@"Connecting to our ticketing system...");
}
-(void) printingTicket
{
NSLog(@"Hmmm, ticket for travelling on the green
dragon.Interesting...");
}
@end

```

А теперь, давайте создадим единый доступ ко всем этим системам:

```

@interface TravellSystemFacade : NSObject
-(void) travellTo:(NSString *)location;
@end
@implementation TravellSystemFacade
-(void) travellTo:(NSString *)location
{

```



```

PathFinder *pf = [[PathFinder alloc] init]; TravellEngine *te
= [[TravellEngine alloc] init];
TicketPrinitingSystem *tp = [[TicketPrinitingSystem alloc]
init];
[pf findCurrentLocation];
[pf findLocationToTravel:location]; [pf makeARoute];
[te findTransport]; [te orderTransport]; [tp createTicket];
[tp printingTicket]; [te travel];
}
@end

```

Как видим, наш фасад знает все про все системы, потому в одном методе он берет и транспортирует нас куда следует. Код теста элементарен:

```

TravellSystemFacade *facade = [[TravellSystemFacade alloc]
init]; [facade travellTo:@"Lviv"];

```

Давайте посмотрим лог:

2013-02-09 17:46:28.442 FacadePattern[2410:c07] Finding your location. Hmmm, here you are!

2013-02-09 17:46:28.444 FacadePattern[2410:c07] So you wanna travell to Lviv 2013-02-09 17:46:28.445 FacadePattern[2410:c07] Okay, to travell to this location we are using google maps....

2013-02-09 17:46:28.446 FacadePattern[2410:c07] Okay, to travell there you will probabply need dragon!Arghhhhh

2013-02-09 17:46:28.446 FacadePattern[2410:c07] Maaaam, can I order a dragon?...

Yes... Yes, green one... Yes, with fire!... No, not a dragon of death... Thank you! 2013-02-09 17:46:28.447 FacadePattern[2410:c07] Connecting to our ticketing system...

2013-02-09 17:46:28.447 FacadePattern[2410:c07] Hmmm, ticket for travelling on the green dragon.Interesting...

17:46:28.448 FacadePattern[2410:c07] Maaan, you are flying on dragon!

11.9 Паттерн Mediator

Медиатор – паттерн которые определяет внутри себя объект, в котором реализуется взаимодействие между некоторым количеством объектов. При этом эти объекты, могут даже не знать про существования друг друга, потому взаимодействий реализованных в медиаторе может быть огромное количество.

Когда стоит использовать:

Когда у вас есть некоторое количество объектов, и очень трудно реализовать взаимодействие между ними. Яркий пример – умный дом. Онозначно есть несколько датчиков, и несколько устройств. К примеру, датчик температуры следит за тем, какая на данный момент температура, а кондиционер умеет охлаждать воздух. Причем кондиционер, не обязательно знает про существования датчика температуры. Есть центральный компьютер, который получает сигналы от каждого из устройств и понимает, что делать в том или ином случае.

Трудно использовать объект ещё раз, так как он взаимодействует и коммуницирует с огромным количеством других объектов.

Логика взаимодействия должна легко настраиваться и расширяться. Собственно, пример медиатора даже писать бессмысленно, потому как это любой контроллер который мы используем во время нашей разработки. Посудите сами – на view есть очень много контролов, и все правила взаимодействия мы прописываем в контроллере. И все же пример не будет лишним Давайте все же создадим пример который показывает создание медиатора для умного дома.

Пусть у нас есть оборудование, которое может взаимодействовать с нашим умным домом:

```

@class CentrallProcessor;
@interface SmartHousePart : NSObject
{
    @private CentrallProcessor *_processor;
}
-(id) initWithCore:(CentrallProcessor *) processor;
-(void) numbersChanged;
@end
@implementation SmartHousePart
-(id) initWithCore:(CentrallProcessor *)processor
{
    self = [super init];
    _processor = processor;
    return self;
}
-(void) numbersChanged
{
    [_processor valueChanged:self];
}
@end

```

Теперь, создадим сердце нашего умного дома:

```

@interface CentrallProcessor : NSObject
@property (nonatomic, weak) Thermometer *_thermometer;
@property (nonatomic, weak) ConditioningSystem *_condSystem;
-(void) valueChanged:(SmartHousePart *) aPart;
@end
@implementation CentrallProcessor
-(void) valueChanged:(SmartHousePart *) aPart
{
    NSLog(@"Value changed! We need to do smth!");
    //detecting that changes are done by thermometer
    if ( [aPart isKindOfClass:[Thermometer class]])
    {
        NSLog(@"Oh, the change is temperature");
        [[self _condSystem] startCondition];
    }
}
@end

```

Тут очень интересный момент – класс CentrallProcessor должен знать про существование SmartHousePart, собственно и SmartHousePart должен знать про существование CentrallProcessor. Если все сделать простым import – то проект не скомпилируется. Поэтому, в SmartHousePart.h мы добавили @class CentrallProcessor; для того чтобы XCode знал что за объект будет использован, и при этом не импортил файл заголовков CentrallProcessor.

Заголовки же мы импортируем в SmartHousePart.m. Далее в классе CentrallPart в методе valueChanged мы определяем с какой деталью и что произошло, чтобы адекватно среагировать. В нашем примере – изменение температуры приводит к тому, что мы включаем кондиционер.

А вот, и код термометра и кондиционера:

```

@interface Thermometer : SmartHousePart
@property (nonatomic) int temperature;
-(void) temperatureChanged:(int) temperature;
@end
@implementation Thermometer

```

```

-(void) temperatureChanged: (int) temperature
{
    self.temperature = temperature; [self numbersChanged];
}
@end
@interface ConditioningSystem : SmartHousePart
-(void) startCondition;
@end
@implementation ConditioningSystem
-(void) startCondition
{
    NSLog(@"Conditioning...");
}
@end

```

Как видим в результате у нас есть два объекта, которые друг про друга не в курсе, и все таки они взаимодействуют друг с другом посредством нашего медиатора CentrallProcessor.

Код для теста:

```

CentrallProcessor *processor = [[CentrallProcessor alloc]
init]; Thermometer *therm = [[Thermometer alloc]
initWithCore:processor];
ConditioningSystem *condSystem =
[[ConditioningSystem alloc] initWithCore:processor];
processor._condSystem = condSystem; processor._thermometer =
therm; [therm temperatureChanged:45];

```

И конечно же лог:

2013-02-12 18:45:05.790 MediatorPattern[8809:c07] Value changed! We need to do smth!

2013-02-12 18:45:05.793 MediatorPattern[8809:c07] Oh, the change is temperature

05.793 MediatorPattern[8809:c07] Conditioning...

11.10 Паттерн Observer

Что такое паттерн Observer? Вот вы, когда-нибудь подписывались на газету? Вы подписываетесь, и каждый раз, когда выходит новый номер газеты, вы получаете его. Вы никуда не ходите, просто даете информацию про себя, и организация, которая выпускает газету, сама знает, куда и какую газету отнести. Второе название этого паттерна – Publish – Subscriber.

Как описывает этот паттерн наша любимая GoF книга – Observer определяет одно-ко-многим отношение между объектами, и если изменения происходят в объекте – все подписанные на него объекты тут же узнают про это изменение.

Идея проста, объект, который мы называем Subject – дает возможность другим объектам, которые реализуют интерфейс Observer, подписываться и отписываться от изменений происходящих в Subject. Когда изменение происходит – всем заинтересованным объектам высылается сообщение, что изменение произошло. В нашем случае – Subject – это издатель газеты, Observer это мы с вами – те, кто подписывается на газету, ну и собственно изменение – это выход новой газеты, а оповещение – отправка газеты всем кто подписался.

Когда используется паттерн:

1. Когда Вам необходимо сообщить всей объектам, подписанным на изменения, что изменение произошло, при этом вы не знаете типы этих объектов.
2. Изменения в одном объекте, требуют, чтобы состояние изменилось в других объектах, причем количество объектов может быть разное.

Реализация этого паттерна возможно двумя способами:

1. Notification

Notification – механизм использования возможностей NotificationCenter самой операционной системы. Использование NotificationCenter позволяет объектам коммуницировать, даже не зная друг про друга. Это очень удобно использовать когда у вас в параллельном потоке пришел push-notification, или же обновилась база, и вы хотите дать об этом знать активному на данный момент View.

Чтобы послать такое сообщение стоит использовать конструкцию типа:

```
NSNotification *broadcastMessage = [NSNotification
notificationWithName:@"broadcastMessage" object:self];
NSNotificationCenter *notificationCenter =
[NSNotificationCenter defaultCenter];
```

Как видим мы создали объект типа NSNotification, в котором мы указали имя нашего оповещения: “broadcastMessage”, и собственно сообщили о нем через NotificationCenter.

Чтобы подписаться на событие в объекте, который заинтересован в изменении, стоит использовать следующую конструкцию:

```
NSNotificationCenter *notificationCenter
=[NSNotificationCenter defaultCenter];
[notificationCenter
addObserver:self
selector:@selector(update:)
name:@"broadcastMessage"
object:nil];
```

Собственно, из кода все более-менее понятно: мы подписываемся на событие, и вызывается метод, который задан в свойстве selector.

2. Стандартный метод.

Стандартный метод, это реализация этого паттерна тогда, когда Subject знает про всех подписчиков, но при этом не знает их типа. Давайте начнем с того, что создадим протоколы для Subject и Observer:

```
@protocol StandardObserver <NSObject>
-(void) valueChanged:(NSString *)valueName newValue:(NSString *) newValue;
@end

@protocol StandardSubject <NSObject>
-(void) addObserver:(id<StandardObserver>) observer;
-(void) removeObserver:(id<StandardObserver>) observer;
-(void) notifyObjects;
@end
```

Теперь, давайте создадим реализацию Subject:

```
@interface StandardSubjectImplementation : NSObject
<StandardSubject>
{
    @private NSString *_valueName;
    @private NSString *_newValue;
}
@property (nonatomic, strong) NSMutableSet
*observerCollection;
-(void)changeValue:(NSString *)valueName andValue:(NSString *)
newValue;
@end

@implementation StandardSubjectImplementation
-(NSMutableSet *) observerCollection
{
    if (_observerCollection == nil)
        _observerCollection = [[NSMutableSet alloc] init];

    return _observerCollection;
}
```

```

}
-(void) addObserver:(id<StandardObserver>)observer
{
[self.observerCollection addObject:observer];
}
-(void) removeObserver:(id<StandardObserver>)observer
{
[self.observerCollection removeObject:observer];
}
-(void) notifyObjects
{
for (id<StandardObserver> observer in self.observerCollection)
{ [observer valueChanged: _valueName newValue:_newValue];
}
}
-(void)changeValue:(NSString *)valueName andValue:(NSString *)
newValue
{
_newValue = newValue;
_valueName = valueName; [self notifyObjects];
}
@end

```

Ну и куда же без обсерверов:

```

@interface SomeSubscriber : NSObject <StandardObserver>
@end
@implementation SomeSubscriber
-(void) valueChanged:(NSString *)valueName newValue:(NSString
*)newValue
{
NSLog(@"And some subscriber tells: Hmm, value %@ changed to
%@", valueName, newValue);
}
@end
//и второй подписчик:
@interface OtherSubscriber : NSObject <StandardObserver>
@end
@implementation OtherSubscriber
-(void) valueChanged:(NSString *)valueName newValue:(NSString
*)newValue
{
NSLog(@"And some another subscriber tells: Hmm, value %@
changed to %@", valueName, newValue);
}
@end

```

Собственно – все:) теперь демо-код:

```

StandardSubjectImplementation *subj =
[[StandardSubjectImplementation alloc] init];
SomeSubscriber *someSubscriber = [[SomeSubscriber alloc]
init]; OtherSubscriber *otherSubscriber = [[OtherSubscriber
alloc] init];
[subj addObserver:someSubscriber]; [subj addObserver:
otherSubscriber];
[subj changeValue:@"strange value" andValue:@"newValue"];

```

И естественно лог:

2013-02-16 17:31:43.176 ObserverPattern[24332:c07] And some subscriber tells: Hmm, value strange value changed to newValue
17:31:43.177 ObserverPattern[24332:c07] And some another subscriber tells: Hmm, value strange value changed to newValue

Конечно без использования KVO, описание паттерна выглядило бы неполным. Одна из моих самых любимых особенностей Obj-C – это key-value coding. Про него очень хорошо написано в официальной документации, но если объяснять на пальцах – то это возможность изменять значения свойств объекта с помощью строк, которые указывают название свойства. Например, такие две конструкции идентичны:

```
kvoSubj.changeableProperty = @"new value";  
[kvoSubj setValue:@"new value" forKey:@"changeableProperty"];
```

Такая гибкость дает нам доступ к еще одной замечательной возможности, которая называется key-value observing. Опять же, все хорошо описанно в документации, но если объяснять на пальцах то это возможность подписаться на изменение любого свойства, у любого объекта который KV compliant, любым объектом. На самом деле легче объяснить на примере.

Давайте создадим клас с одним свойством, которое мы будем менять:

```
@interface KVOSubject : NSObject  
@property (nonatomic, strong) NSString *changeableProperty;  
@end  
@implementation KVOSubject  
@end
```

И создадим объект, который будет слушать изменение свойства changeableProperty:

```
@interface KVOObserver : NSObject  
@end  
@implementation KVOObserver  
-(void) observeValueForKeyPath:(NSString *)keyPath  
ofObject:(id)object change:(NSDictionary *)change  
context:(void *)context  
{  
    NSLog(@"KVO: Value changed;");  
}  
@end
```

Как видим, этот класс реализует только один метод: observeValueForKeyPath. Этот метод будет вызван, когда поменяется свойство объекта, за которым мы наблюдаем.

Теперь тест:

```
KVOSubject *kvoSubj = [[KVOSubject alloc] init]; KVOObserver  
*kvoObserver = [[KVOObserver alloc] init];  
[kvoSubj addObserver:kvoObserver  
forKeyPath:@"changeableProperty"  
options:NSKeyValueObservingOptionNew context:nil];  
[kvoSubj setValue:@"new value" forKey:@"changeableProperty"];  
//because kvoSubj will be deallocated after this functions  
ends we need to remove observer information.  
[kvoSubj removeObserver:kvoObserver  
forKeyPath:@"changeableProperty"];
```

Как видно из примера, мы для объекта за которым мы наблюдаем, выполняем функцию addObserver – где устанавливаем кто будет наблюдать за изменениями, за изменениями какого свойства мы будем наблюдать и остальные опции. Дальше меняем значение свойства, и так как мы все это проделываем на нажатие кнопки – в конце мы удаляем наблюдателя с нашего объекта, что бы память не текла. Лог говорит сам за себя:

11:41:58.051 ObserverPattern[26689:c07] KVO: Value changed;

11:41:58.052 ObserverPattern[26689:c07] KVO: Value changed;

11.11 Паттерн Composite

Вы задумывались как много в нашей жизни древовидных структур? Начиная собственно от самих деревьев, и заканчивая структурами компаний. Да даже не компаний – целые страны используют древовидные структуры, чтобы построить власть.

Во главе компании или страны частенько стоит один человек, у него есть несколько помощников. У них тоже есть помощники, и так далее... Если нарисовать их отношения на листе бумаги – увидим дерево.

Очень часто, и мы используем такие типы данных, которые лучше всего хранятся в древовидной структуре. Возьмите, например, стандартный UI: сначала у нас есть View, в нем находятся Subview, в которых могут быть или другие View, или все такие компоненты. Та же самая структура.

Именно для хранения таких типов данных, а вернее их организации, используется паттерн – «Композит».

Когда использовать такой паттерн?

Собственно когда вы работаете с древовидными типами данных, или хотите отобразить иерархию данных таким образом. Давайте разберем более детально структуру.

Сначала всегда есть контейнер, в котором находятся все остальные объекты. Контейнер может хранить как другие контейнеры – ветки нашего дерева, так и объекты которые контейнерами не являются – листья нашего дерева. Не сложно представить, что контейнеры второго уровня могут хранить как другие контейнеры, так и листья.

Пример. Начнем с создания протокола для наших объектов:

```
@protocol CompositeObjectProtocol <NSObject>
-(NSString *) getData;
-(void) addComponent:(id<CompositeObjectProtocol>) aComponent;
@end
```

Создадим объект листа:

```
@interface LeafObject : NSObject <CompositeObjectProtocol>
@property (nonatomic, strong) NSString *leafValue;
@end
@implementation LeafObject
-(NSString *) getData
{
    return [[NSString alloc] initWithFormat:@"%d", self.leafValue ];
}
-(void) addComponent:(id) aComponent
{
    NSLog(@"Can't add component. Sorry, man");
}
@end
```

Как видим наш объект не может добавлять себе детей (ну он же не контейнер), и может возвращать свое значение с помощью метода getData. Теперь нам необходим контейнер:

```
@interface Container : NSObject <CompositeObjectProtocol>
@property (nonatomic, strong) NSMutableArray *components;
@end
@implementation Container
-(NSMutableArray *) components
```

```

{
if (_components == nil)
_components = [[NSMutableArray alloc] init];
return _components;
}
-(void) addComponent:(id<CompositeObjectProtocol>) aComponent
{
[self.components addObject:aComponent];
}
-(NSString *) getData
{
NSMutableString *valueToReturn = [[NSMutableString alloc]
init]; [valueToReturn appendString:@"<ContainerValues>"];
for ( id<CompositeObjectProtocol> object in _components)
{
[valueToReturn appendString:[object getData]];
}
[valueToReturn appendString:@"</ContainerValues>"];
return valueToReturn;
}
@end

```

Как видим, наш контейнер может добавлять в себя детей, которые могут быть как типа Container, так и типа LeafObject. Метод `getData` используется всеми объектами в массиве `components`, и вызывает тот же самый метод в детях. Вот собственно и все.

Теперь, пример:

```

Container *rootContainer = [[Container alloc] init];
LeafObject *object = [[LeafObject alloc] init];
object.leafValue = @"level1 value";
[rootContainer addComponent:object];
Container *firstLevelContainer1 = [[Container alloc] init];
LeafObject *object2 = [[LeafObject alloc] init];
object2.leafValue = @"level2 value"; [firstLevelContainer1
addComponent:object2];
[firstLevelContainer1 addComponent:firstLevelContainer1];
Container *firstLevelContainer2 = [[Container alloc] init];
LeafObject *object3 = [[LeafObject alloc] init];
object3.leafValue = @"level2 value 2"; [firstLevelContainer2
addComponent:object3];
[firstLevelContainer2 addComponent:firstLevelContainer2];
NSLog(@"%@", rootContainer.getData);

```

И конечно лог:

2013-02-17 13:04:09.470 CompositePattern[27392:c07]

```

<ContainerValues>
<level1 value/>
<ContainerValues>
<level2 value/>
</ContainerValues>
<ContainerValues>
<level2 value 2/>
</ContainerValues>
</ContainerValues>

```

11.12 Паттерн Iterator

Я задумался о том, какой бы пример из жизни привести, чтобы показать пример как

работает паттерн итератор, и оказалось что это не такой простое задание. И как показывает практика, самый простой пример – это обычная вендинг машина. У нас есть контейнер, который разделен на секции, каждая из которых содержит определенный вид товара, например, набор бутылок Coca-Cola. Когда мы заказываем товар, то нам выпадет следующий из коллекции. Образно говоря, команда `cocaColaCollection.next`. Две независимые части – контейнер и итератор.

Паттерн итератор позволяет последовательно обращаться к коллекции объектов, не вникая в то, что же это за коллекция.

Разделяют два вида итераторов – внутренний и внешний. Как видно из названия, внешний итератор – итератор, про который знает клиент, и собственно он сам (клиент) создает коллекцию, к которой обращается итератор. Внутренний итератор – это внутренняя кухня самой коллекции, которая предоставляет интерфейс клиенту для итерирования.

При внешнем итераторе, клиенту надо:

1. Вообще знать про существование итератора, хоть это и дает больше контроля.
2. Создавать и управлять итератором.
3. Можно использовать различные итераторы, для различных алгоритмов итерации.

При внутреннем:

1. Клиенту совершенно не известно существование итератора. Он просто использует интерфейс коллекции.
2. Коллекция сама создает и управляет итератором.
3. Коллекция может менять различные итераторы, при этом не трогая код клиента.

Когда использовать итератор:

1. Вам необходимо достучаться к объектам коллекции, без того чтобы щупать внутренности коллекции.
2. Вам надо обходить объекты коллекции различными способами (вспомните «Композит» – у вас коллекция может быть древовидной).
3. Вам необходимо дать унифицированный интерфейс для различных подходов итерации.

Самый простой пример внешнего итератора, это использование класса.

`NSEnumerator:`

```
NSEnumerator *enumerator = [internalArrayCollection
objectEnumerator]; NSString* element;
while (element = [enumerator nextObject]) {
    NSLog(@"%@", element);
}
```

Как видим, мы просто вызываем у объекта `internalArrayCollection` метод `objectEnumerator` – и получаем необходимый нам итератор. Вообще можно не заморачиваться, и использовать обычный цикл `for`:

```
for (NSString *element in internalArrayCollection)
{
    NSLog(@"%@", element);
}
```

Одним из примеров реализации внешнего итератора – может быть итерация с помощью блоков:

```
[internalArrayCollection enumerateObjectsUsingBlock:
^(id obj, NSUInteger idx, BOOL *stop) {
    if([obj localizedCaseInsensitiveCompare:@"Dima"] ==
    NSOrderedSame)
    {
    }
}
```

```

    }];
    NSLog(@"Dima has been found!");
    *stop = YES;

```

Радость этого метода в том, что сам алгоритм итерации может написать другой программист, вам же необходимо будет только использовать блок написанный этим программистом. Все выглядит приблизительно так:

```

//создание блока поиска Dima в массиве строчек
void (^simpleDimaSearchBlock)(id, NSUInteger, BOOL*) =
^(id obj, NSUInteger idx, BOOL *stop){
    if([obj localizedCaseInsensitiveCompare:@"Dima"] ==
    NSOrderedSame)
    {
        NSLog(@"Dima has been found!");
        *stop = YES;
    }
};
[interallArrayCollection
enumerateObjectsUsingBlock:simpleDimaSearchBlock];

```

Теперь давайте создадим свой итератор, а то и два. Пускай у нас будет коллекция товаров, одни из них будут сломаны, другие же – целые. Создадим два итератора, которые будут бегать по разным типам товаров. Итак, для начала сам класс товаров:

```

@interface ItemInShop : NSObject
@property (nonatomic, strong) NSString *name;
@property (nonatomic) BOOL isBroken;
-(id) initWithArgs:(NSString *)aName
andQaulity:(BOOL)isBroken;
@end
@implementation ItemInShop
-(id) initWithArgs:(NSString *)aName andQaulity:(BOOL)isBroken
{
    self = [super init]; self.name = aName; self.isBroken =
    isBroken; return self;
}
@end

```

Как видим не густо – два свойства, и инициализатор.

Теперь давайте создадим склад, в котором собственно товары то и будут:

```

@interface ShopWarehouse : NSObject
{
    @private NSMutableArray *goods;
    @private GoodItemsEnumerator *goodItemsEnumerator;
    @private BadItemsEnumerator *badItemsEnumerator;
}
-(void) addItem:(ItemInShop *)anItem;
-(NSEnumerator *) getBrokenItemsEnumerator;
-(NSEnumerator *) getGoodItemsEnumerator;
@end
@implementation ShopWarehouse
-(id) init
{
    self = [super init];
    goods = [[NSMutableArray alloc] init];
    return self;
}
-(void) addItem:(ItemInShop *)anItem

```

```

{
[goods addObject:anItem];
}
-(NSEnumerator *) getBrokenItemsEnumerator
{
badItemsEnumerator      =      [[BadItemsEnumerator      alloc]
initWithItems:goods];
return badItemsEnumerator;
}
-(NSEnumerator *) getGoodItemsEnumerator
{
goodItemsEnumerator      =      [[GoodItemsEnumerator      alloc]
initWithItems:goods];
return goodItemsEnumerator;
}
@end

```

Как видим, наш склад умеет добавлять товары, а также возвращать два таинственных объекта под названием GoodItemsEnumerator и BadItemsIterator. Собственно их назначение очевидно, давайте посмотрим на реализацию. Для начала создадим базовый класс для обоих:

```

@interface BasicEnumerator : NSEnumerator
-(id)initWithItems:(NSMutableArray *)anItems;
-(NSArray *)allObjects;
-(id) nextObject;
@end

```

Как видим, это просто интерфейс, который предполагает реализацию 3х методов: инициализация, вернуть все объекты, и вернуть следующий объект. Давайте создадим два итератора, как и задумывалось:

```

@interface BadItemsEnumerator : BasicEnumerator
{
@private NSMutableArray *itemsArray;
@private NSEnumerator *internalEnumerator;
}
@end
@implementation BadItemsEnumerator
-(id) initWithItems:(NSMutableArray *)anItems
{
self = [super init];
itemsArray = [[NSMutableArray alloc] init];
for ( ItemInShop *item in anItems)
{
if (item.isBroken)
[itemsArray addObject:item];
}
internalEnumerator = [itemsArray objectEnumerator];
return self;
}
-(NSArray *)allObjects
{
return itemsArray;
}
-(id) nextObject
{

```

```

return [internalEnumerator nextObject];
}
@end

```

Мы не приводим код `GoodItemsIterator`, потому как отличаться они будут только в одной строчке:

```

if (!item.isBroken)

```

Как видим во время инициализации, мы создаем свою копию данных, в которых только плохие товары. Так же создаем свой внутренний итератор, из стандартных Cocoa.

Тест:

```

//создание тестовых данных
shopWarehouse = [[ShopWarehouse alloc] init];
[shopWarehouse addItem:[ItemInShop alloc] initWithArgs:@"Item1" andQuality:NO]]; [shopWarehouse
addItem:[ItemInShop alloc] initWithArgs:@"Item2"
andQuality:NO]];
[shopWarehouse addItem:[ItemInShop alloc] initWithArgs:@"Item3" andQuality:YES]]; [shopWarehouse
addItem:[ItemInShop alloc] initWithArgs:@"Item4"
andQuality:YES]]; [shopWarehouse addItem:[ItemInShop
alloc] initWithArgs:@"Item5" andQuality:NO]];

```

Сам тест:

```

GoodItemsEnumerator *goodIterator = [shopWarehouse
getGoodItemsEnumerator]; BadItemsEnumerator *badIterator =
[shopWarehouse getBrokenItemsEnumerator];
ItemInShop *element;
while (element = [goodIterator nextObject]) {
NSLog(@"Good Item = %@", element.name);
}
while (element = [badIterator nextObject]) {
NSLog(@"Bad Item = %@", element.name);
}

```

И конечно лог:

```

01:18:10.401 IteratorPattern[5000:c07] Good Item = Item1 2013-02-25 01:18:10.403
IteratorPattern[5000:c07] Good Item = Item2 2013-02-25 01:18:10.403
IteratorPattern[5000:c07] Good Item = Item5 2013-02-25 01:18:10.404
IteratorPattern[5000:c07] Bad Item = Item3 2013-02-25 01:18:10.405
IteratorPattern[5000:c07] Bad Item = Item4

```

11.13 Паттерн Visitor

Вот у каждого дома вероятнее всего есть холодильник. Что будет если холодильник сломается? Некоторые пойдут почитают в интернете как чинить холодильник, какая модель, попробуют поколдовать над ним, и разочаровавшись вызовут мастера по ремонту холодильников. Заметьте – холодильник ваш, но функцию “Чинить Холодильник” выполняет совершенно другой человек, про которого вы ничего не знаете, а попросту – обычный визитер.

Паттерн визитер – позволяет вынести из наших объектов логику, которая относится к этим объектам, в отдельный класс, что позволяет нам легко изменять/добавлять алгоритмы, при этом не меняя логику самого класса.

Когда мы захотим использовать этот паттерн:

1. Когда у нас есть сложный объект, в котором содержится большое количество различных элементов, и вы хотите выполнять различные операции в зависимости от типа этих элементов.
2. Вам необходимо выполнять различные операции над классами, и при этом Вы не хотите писать много кода внутри реализации этих классов.
3. В конце – концов, вам нужно добавлять различные операции над элементами, и вы не хотите постоянно обновлять классы этих элементов.

Давайте вернемся к примеру из предыдущего раздела, только теперь сложнее – у нас есть несколько складов, в каждом складе может храниться товар. Один визитер будет смотреть склады, другой визитер будет называть цену товара в складе.

Итак, для начала сам товар:

```
@interface WarehouseItem : NSObject
@property (nonatomic, strong) NSString *name;
@property (nonatomic) BOOL isBroken;
@property (nonatomic) int price;
-(id) initWithArgs:(NSString *) aName andQuality:(BOOL)
isBrokenState andPrice:(int) aPrice;
@end
@implementation WarehouseItem
-(id) initWithArgs:(NSString *) aName
andQuality:(BOOL) isBrokenState andPrice:(int) aPrice
{
self = [super init];
self.name = aName; self.isBroken = isBrokenState; self.price =
aPrice;
return self;
}
@end
```

И естественно сам склад:

```
@interface Warehouse : NSObject
{
private NSMutableArray *_itemsArray;
}
-(void) addItem:(WarehouseItem *) anItem;
-(void) accept:(id<BasicVisitor>) visitor;
@end
@implementation Warehouse
-(void) addItem:(WarehouseItem *) anItem
{
if (!_itemsArray)
_itemsArray = [[NSMutableArray alloc] init];
[_itemsArray addObject:anItem];
}
-(void) accept:(id<BasicVisitor>) visitor
{
[visitor visit:self];
for (WarehouseItem *item in _itemsArray) [visitor visit:item];
}
@end
```

Как видим, наш склад умеет хранить и добавлять товар, но также обладает неизвестным методом accept, который принимает в себя визитер и вызывает его метод visit. Чтобы картинка сложилась, давайте создадим протокол BasicVisitor и различных визитеров:

```
@protocol BasicVisitor <NSObject>
-(void) visit:(id)anObject;
@end
```

Как видим, протокол требует реализацию только одного метода. Теперь давайте перейдем к самим визитерам:

```
@interface QualityCheckerVisitor : NSObject <BasicVisitor>
@end
@implementation QualityCheckerVisitor
-(void) visit:(id)anObject
{
    if ([anObject isKindOfClass:[WarehouseItem class]])
    {
        if ([anObject isBroken])
        {
            NSLog(@"Item: %@ is broken", [anObject name]);
        }
        else
        {
            NSLog(@"Item: %@ is pretty cool!", [anObject name]);
        }
    }
    if ([anObject isKindOfClass:[Warehouse class]])
    {
        NSLog(@"Hmmm, nice warehouse!");
        return;
    }
}
@end
```

Если почитать код, то сразу видно, что визитер при вызове своего метода visit определяет тип объекта, который ему передан, и выполняет определенные функции в зависимости от этого типа. Данный объект просто говорит, что вещи на складе поломаны, а так же что ему нравится склад.

```
@interface PriceCheckerVisitor : NSObject <BasicVisitor>
@end
@implementation PriceCheckerVisitor
-(void) visit:(id)anObject
{
    if ([anObject isKindOfClass:[WarehouseItem class]])
    {
        NSLog(@"Item: %@ have price = %i", [anObject name], [anObject price]);
    }
    if ([anObject isKindOfClass:[Warehouse class]])
    {
        NSLog(@"Hmmm, I don't know how much Warehouse costs!");
        return;
    }
}
@end
```

В принципе этот визитер делает то же самое, только в случае склада он признается, что растерян, а в случае товара говорит цену товара.

Теперь давайте запустим то, что у нас получилось. Код генерации тестовых данных:

```

_localWarehouse = [[Warehouse alloc] init];
[_localWarehouse addItem:[WarehouseItem alloc]
initWithArgs:@"Item1" andQuality:NO andPrice:25]];
[_localWarehouse addItem:[WarehouseItem alloc]
initWithArgs:@"Item2" andQuality:NO andPrice:32]];
[_localWarehouse addItem:[WarehouseItem alloc]
initWithArgs:@"Item3"
andQuality:YES andPrice:45]];
[_localWarehouse addItem:[WarehouseItem alloc]
initWithArgs:@"Item4" andQuality:NO andPrice:33]];
[_localWarehouse addItem:[WarehouseItem alloc]
initWithArgs:@"Item5" andQuality:NO andPrice:12]];
[_localWarehouse addItem:[WarehouseItem alloc]
initWithArgs:@"Item6"
andQuality:YES andPrice:78]];
[_localWarehouse addItem:[WarehouseItem alloc]
initWithArgs:@"Item7" andQuality:YES andPrice:34]];
[_localWarehouse addItem:[WarehouseItem alloc]
initWithArgs:@"Item8" andQuality:NO andPrice:51]];
[_localWarehouse addItem:[WarehouseItem alloc]
initWithArgs:@"Item9" andQuality:NO andPrice:25]];

```

И собственно сам тестовый код:

```

PriceCheckerVisitor *visitor = [[PriceCheckerVisitor alloc]
init];
QualityCheckerVisitor *qualityVisitor =
[[QualityCheckerVisitor alloc] init];
[_localWarehouse accept:visitor];
[_localWarehouse accept:qualityVisitor];

```

Итак, при вызове метода accept нашего склада, визитер сначала проверяет наш склад, а потом проверяет каждый товар на этом складе. При этом мы можем менять как визитера, так и алгоритм, и это не повлечет изменения в коде клиента.

Традиционный лог:

```

00:19:47.756 VisitorPattern[8748:c07] Hmmm, I don't know how much Warehouse costs!
2013-02-26 00:19:47.759 VisitorPattern[8748:c07] Item: Item1 have price = 25
2013-02-26 00:19:47.759 VisitorPattern[8748:c07] Item: Item2 have price = 32
2013-02-26 00:19:47.760 VisitorPattern[8748:c07] Item: Item3 have price = 45
2013-02-26 00:19:47.761 VisitorPattern[8748:c07] Item: Item4 have price = 33
2013-02-26 00:19:47.762 VisitorPattern[8748:c07] Item: Item5 have price = 12
2013-02-26 00:19:47.763 VisitorPattern[8748:c07] Item: Item6 have price = 78
2013-02-26 00:19:47.763 VisitorPattern[8748:c07] Item: Item7 have price = 34
2013-02-26 00:19:47.764 VisitorPattern[8748:c07] Item: Item8 have price = 51
2013-02-26 00:19:47.765 VisitorPattern[8748:c07] Item: Item9 have price = 25
00:19:47.765 VisitorPattern[8748:c07] Hmmm, nice warehouse! 2013-02-26 00:19:47.766
VisitorPattern[8748:c07] Item: Item1 is pretty cool! 2013-02-26 00:19:47.767
VisitorPattern[8748:c07] Item: Item2 is pretty cool! 2013-02-26 00:19:47.767
VisitorPattern[8748:c07] Item: Item3 is broken 2013-02-26 00:19:47.768
VisitorPattern[8748:c07] Item: Item4 is pretty cool! 2013-02-26 00:19:47.769
VisitorPattern[8748:c07] Item: Item5 is pretty cool! 2013-02-26 00:19:47.837
VisitorPattern[8748:c07] Item: Item6 is broken 2013-02-26 00:19:47.837
VisitorPattern[8748:c07] Item: Item7 is broken 2013-02-26 00:19:47.837
VisitorPattern[8748:c07] Item: Item8 is pretty cool! 2013-02-26 00:19:47.838
VisitorPattern[8748:c07] Item: Item9 is pretty cool!

```

11. 14 Паттерн Decorator

Пример для данного паттерна использует категории, хотя в классическом исполнении сам декоратор должен быть отдельным объектом.

Класный пример декоратора – различные корпуса для новых телефонов. Для начала у нас есть телефон. Но так как телефон дорогой, мы будем счастливы, если он не разобьется при падении – поэтому мы покупаем чехол для него. То есть, к уже существующему предмету мы добавили функционал защиты от падения. Мы купили стильный чехол – теперь наш телефон еще и выглядит отлично. А потом мы докупили съемный объектив, с помощью которого можно делать фотографии с эффектом “рыбьего глаза”. Декорировали наш телефон дополнительным функционалом.

Вот, приблизительно так выглядит реально описание паттерна декоратор. Теперь описание GoF. Декоратор добавляет некий функционал уже существующему объекту.

Когда использовать этот паттерн:

1. Вы хотите добавить определенному объекту дополнительные возможности, при этом, не задевая и не меняя других объектов.
2. Дополнительные возможности класса – опциональны.

Радость Objective-C в данном случае – это использование категорий. Здесь мы не будем детально описывать категории, но следует знать, что категории – это возможность расширить любой объект дополнительными методами без наследования от него. Давайте возьмем простой пример – декорирование Cocoa классов. Например, добавим новый метод для объекта NSDate:.

Например, нам нужно иметь возможность отформатировать любую дату в нашем приложении и получить её в виде строки. Для начала создадим категорию:

```
@interface NSDate (StringDate)
- (NSString *) convertDateToString;
@end
@implementation NSDate (StringDate)
- (NSString *) convertDateToString
{
    NSDateFormatter *formatter = [[NSDateFormatter alloc] init];
    [formatter setDateFormat:@"%yyyy/dd/MM"];
    return [formatter stringFromDate:self];
}
```

Как видим, наша категория определяет только один метод “convertDateToString”, который форматирует дату в какой-то совсем странный формат, но у нас такая задача.

Теперь надо сделать эту категорию используемой для всех объектов NSDate в нашем приложении. Для этого в файле appName-Prefix, добавляем строку:

```
#ifdef OBJC
#import <UIKit/UIKit.h>
#import <Foundation/Foundation.h>
#import "NSDate+StringDate.h"
#endif
```

Строчка выделенна жирным шрифтом, чтобы не потерялась.

В принципе это все. Примерный код тестирования выглядит следующим образом:

```
NSDate *dateNow = [NSDate date];
NSLog(@"Date is %@", [dateNow convertDateToString]);
```

Традиционный лог:

00:30:18.328 DecoratorPattern[11731:c07] Date is 2013/01/03

11.15 Паттерн Chain of responsibility

Представьте себе очередь людей которые пришли за посылками. Выдающий посылки человек, дает первую посылку первому в очереди человеку, он смотрит на имя-фамилию на коробке, видит что посылка не для него, и передает посылку дальше. Второй человек делает собственно то же самое, и так пока не найдется получатель.

Цепочка ответственности (chain of responsibility) – позволяет вам передавать объекте по цепочке объектов-обработчиков, пока не будет найден необходимый объект обработчик.

Когда использовать этот паттерн:

1. У вас более чем один объект-обработчик.
2. У вас есть несколько объектов обработчика, при этом вы не хотите специфицировать который объект должен обрабатывать в данный момент времени.

Как всегда – пример:

Представим что у нас есть конвейер, который обрабатывает различные предметы на нем: игрушки, электронику и другие.

Для начала создадим классы объектов, которые могут быть обработаны нашими обработчиками:

```
//базовый объект
@interface BasicItem : NSObject
@end
@implementation BasicItem
@end
//игрушка
@interface Toy : BasicItem
@end
@implementation Toy
@end
//электроника
@interface Electronics : BasicItem
@end
@implementation Electronics
@end
//и мусор
@interface Trash : BasicItem
@end
@implementation Trash
@end
```

Теперь создадим обработчики:

```
@interface BasicHandler : NSObject
{
@private BasicHandler *_nextHandler;
}
@property (nonatomic, strong) BasicHandler *nextHandler;
-(void) handleItem:(BasicItem *) item;
@end
@implementation BasicHandler
@end
```

Как видим, наш базовый обработчик, умеет обрабатывать объекты типа BasicItem. И самое важное – он имеет ссылку на следующий обработчик (как в нашей очереди, про людей передающие посылку).

Давайте создадим код обработчика игрушки:

```
@interface ToysHandler : BasicHandler
@end
@implementation ToysHandler
-(void) handleItem:(BasicItem *)item
{
    if ([item isKindOfClass:[Toy class]])
    {
        NSLog(@"Toy found. Handling");
    }
    else
    {
        NSLog(@"Toy not found. Handling using next handler");
        [self.nextHandler handleItem:item];
    }
}
@end
```

Как видим, если обработчик получает объект класса Toy – то он его обрабатывает, если нет – то обработчик передает объект следующему обработчику.

По аналогии создадим два следующих обработчика: для электроники, и мусора:

```
//хэндлер электроники
@interface ElectronicsHandler : BasicHandler
@end
@implementation ElectronicsHandler
-(void) handleItem:(BasicItem *)item
{
    if ([item isKindOfClass:[Electronics class]])
    {
        NSLog(@"Electronics found. Handling");
    }
    else
    {
        NSLog(@"Electronics not found. Handling using next handler");
        [self.nextHandler handleItem:item];
    }
}
@end
//хэндлер мусора
@interface OtherItemsHandler : BasicHandler
@implementation OtherItemsHandler
-(void) handleItem:(BasicItem *)item
{
    NSLog(@"Found undefined item. Destroying");
}
}
@end
```

Как видим OtherItemsHandler в случае, когда до него дошло дело – объект уничтожает, и не пробует дергать следующий обработчик.

Давайте тестировать:

```
BasicHandler *toysHandler = [[ToysHandler alloc] init];
BasicHandler *electronicsHandler = [[ElectronicsHandler alloc]
init]; BasicHandler *otherItemHandler = [[OtherItemsHandler
alloc]init];
```

```
electronicsHandler.nextHandler = otherItemHandler;
toysHandler.nextHandler = electronicsHandler;
BasicItem *toy = [[Toy alloc] init];
BasicItem *electronic = [[Electronics alloc] init]; BasicItem
*trash = [[Trash alloc] init];
[toysHandler handleItem:toy]; [toysHandler
handleItem:electronic]; [toysHandler handleItem:trash];
```

Как видим мы в начале создаем обработчики, потом скрепляем их в цепь, и пытаемся обработать различные элементы. Традиционно лог:

```
2013-03-02 15:35:35.668 ChainOfResponsibility[16777:c07] Toy found. Handling
2013-03-02 15:35:35.671 ChainOfResponsibility[16777:c07] Toy not found. Handling using
next handler
2013-03-02 15:35:35.672 ChainOfResponsibility[16777:c07] Electronics found.
Handling
2013-03-02 15:35:35.673 ChainOfResponsibility[16777:c07] Toy not found. Handling using
next handler
2013-03-02 15:35:35.673 ChainOfResponsibility[16777:c07] Electronics not found. Handling
using next handler
15:35:35.674 ChainOfResponsibility[16777:c07] Found undefined item. Destroying
```

11.16 Паттерн Template Method

Вы заметили как много в нашей жизни шаблонов? Например, наше поведение когда мы приходим в незнакомый дом:

1. Зайти.
2. Поздороваться с хозяевами.
3. Разуаться.
4. Пройти и удивляться какая большая/уютная/забавная квартира.

Или же когда мы приходим в квартиру, где происходит ремонт:

1. Зайти.
2. Поздороваться с хозяевами.
3. Не разуваться, потому как грязно.
4. Поудивляться, когда хозяин квартиры поведает нам смелость его архитектурной мысли.

В целом, все происходит практически одинаково, но с изюминкой в каждом различном случае. Наверное, поэтому, это и называется шаблоном поведения. Шаботонный метод задает алгоритму пошаговую инструкцию. Элементы алгоритма же, определяются в наследующих классах.

Сам паттерн достаточно интуитивный, и многие давно уже использовали его. Поэтому давайте попробуем сделать пример. Вернемся к старой практике, писать примеры по созданию телефонов.

Итак, напишем наш шаботонный клас, с помощью которого будем создавать телефон:

```
@interface AnyPhoneTemplate : NSObject
//it will be template method
-(void) makePhone;
-(void) takeBox;
-(void) takeMicrophone;
-(void) takeCamera;
-(void) assemble;
@end
@implementation AnyPhoneTemplate
```

```
//our template method
-(void) makePhone
{
[self takeBox]; [self takeCamera];
[self takeMicrophone];
[self assemble];
}
-(void) takeBox
{
NSLog(@"Taking a box");
}
-(void) takeCamera
{
NSLog(@"Taking a camera");
}
-(void) takeMicrophone
{
NSLog(@"Taking a microphone");
}
-(void) assemble
{
NSLog(@"Assembling everythig");
}
@end
```

Как вы уже наверное догадались – сам шаблонный метод, это метод makePhone, который задает последовательность вызовов методов, необходимых для складывания телефонов. Давайте теперь научим нашу программу создавать айфоны:

```
@interface iPhoneMaker : AnyPhoneTemplate
-(void) design;
@end
@implementation iPhoneMaker
-(void) takeBox
{
[self design]; [super takeBox];
}
-(void) design
{
NSLog(@"Putting label 'Designed in California'");
}
@end
```

Как видим, у сборщика телефонов Apple есть один дополнительный метод – design, а также перегруженный метод takeBox, в котором дополнительно вызывается метод design и после этого вызывается родительский метод takeBox.

На очереди сборка Android:

```
@interface AndroidMaker : AnyPhoneTemplate
-(void) addRam;
-(void) addCPU;
@end
@implementation AndroidMaker
-(void) assemble
{
[self addCPU]; [self addRam]; [super assemble];
}
```

```

-(void) addCPU
{
    NSLog(@"Installing 4 more CPUs");
}
-(void) addRam
{
    NSLog(@"Installing 2Gigs of RAM");
}
@end

```

Как видим у сборщика андроида целых два дополнительных метода, и перегруженный метод `assemble`.

Тест здесь – элементарный:

```

AndroidMaker *android = [[AndroidMaker alloc] init];
iPhoneMaker *iphone = [[iPhoneMaker alloc] init];
[android makePhone]; [iphone makePhone];

```

Традиционный лог:

```

2013-03-03 22:56:28.996 TemplateMethod[21040:c07] Taking a box
2013-03-03 22:56:28.998 TemplateMethod[21040:c07] Taking a camera
2013-03-03 22:56:28.999 TemplateMethod[21040:c07] Taking a microphone
2013-03-03 22:56:29.000 TemplateMethod[21040:c07] Installing 4 more CPUs
2013-03-03 22:56:29.000 TemplateMethod[21040:c07] Installing 2Gigs of RAM
2013-03-03 22:56:29.001 TemplateMethod[21040:c07] Assembling everythig
2013-03-03 22:56:29.001 TemplateMethod[21040:c07] Putting label 'Designed in California'
2013-03-03 22:56:29.002 TemplateMethod[21040:c07] Taking a box
2013-03-03 22:56:29.003 TemplateMethod[21040:c07] Taking a camera
2013-03-03 22:56:29.003 TemplateMethod[21040:c07] Taking a microphone
2013-03-03 22:56:29.003 TemplateMethod[21040:c07] Assembling everything

```

11.17 Паттерн Strategy

Если Ваша девушка злая, вы скорее всего будете общаться с ней осторожно. Если на вашем проекте завал, то вероятнее всего вы не будете предлагать в команде вечером выпить пива или поиграть в компьютерные игры. В различных ситуациях, у нас могут быть очень разные стратегии поведения. К примеру, в приложении вы можете использовать различные алгоритмы сжатия, в зависимости от того с каким форматом изображения вы работаете, или же куда вы хотите после этого изображение деть. Вот мы и добрались до паттерна «Стратегия».

Также отличным примером может быть MVC паттерн – в разных случаях мы можем использовать разные контроллеры для одного и того же View (к примеру авторизованный и не авторизованный пользователь).

Паттерн «Стратегия» определяет семейство алгоритмов, которые могут взаимозаменяться.

Когда использовать паттерн:

1. Вам необходимы различные алгоритмы.
2. Вы очень не хотите использовать много вложенных If-ов.
3. В различных случаях ваш класс работает по разному.

Давайте напишем пример – RPG игра, в которой у вас есть различные стратегии нападения Вашими персонажами. Каждый раз, когда вы делаете ход, ваши персонажи делают определенное действие. Итак, для начала управление персонажами.

Создадим базовую стратегию:

```
@interface BasicStrategy : NSObject
-(void) actionCharacter1;
-(void) actionCharacter2;
-(void) actionCharacter3;
@end
```

Как видно из кода стратегии – у нас есть 3 персонажа, каждый из которых может совершать одно действие! Давайте научим персонажей нападать!

```
@interface AttackStrategy : BasicStrategy
@end
@implementation AttackStrategy
-(void) actionCharacter1
{
NSLog(@"Character 1: Attack all enemies!");
}
-(void) actionCharacter2
{
NSLog(@"Character 2: Attack all enemies!");
}
-(void) actionCharacter3
{
NSLog(@"Character 3: Attack all enemies!");
}
@end
```

Как видим, при использовании такой стратегии наши персонажи нападают на все что движется. Давайте научим их защищаться:

```
@interface DefenceStrategy : BasicStrategy
@end
@implementation DefenceStrategy
-(void) actionCharacter1
{
NSLog(@"Character 1: Attack all enemies!");
}
-(void) actionCharacter2
{
NSLog(@"Character 2: Healing Character 1!");
}
-(void) actionCharacter3
{
NSLog(@"Character 3: Protecting Character 2!");
}
@end
```

Как видим, во время защитной стратегии, наши персонажи действуют по-другому – кто атакует, кто лечит, а некоторый даже защищают. А теперь как-то надо это все использовать. Давайте создадим нашего игрока:

```
@interface Player : NSObject
@property (nonatomic, strong) BasicStrategy *_strategy;
-(void) makeAction;
-(void) changeStrategy:(BasicStrategy *) strategy;
@end
@implementation Player
-(void) makeAction
```

```

{
[self._strategy      actionCharacter1];      [self._strategy
actionCharacter2]; [self._strategy actionCharacter3];
}
-(void) changeStrategy:(BasicStrategy *)strategy
{
self._strategy = strategy;
}
@end

```

Как видим, наш игрок может только менять стратегию и действовать в зависимости от этой стратегии.

Код для тестирования:

```

Player *p = [[Player alloc] init];
AttackStrategy *a = [[AttackStrategy alloc] init];
DefenceStrategy *d = [[DefenceStrategy alloc] init];
[p changeStrategy:a]; [p makeAction];
[p changeStrategy:d];
[p makeAction];

```

Собственно все предельно ясно. В первом случае наши персонажи будут активно атаковать, а после смены стратегии уйдут в глухую оборону.

Традиционный лог:

```

23:57:44.797 StrategyPatterns[22420:c07] Character 1: Attack all enemies!
2013-03-04 23:57:44.799 StrategyPatterns[22420:c07] Character 2: Attack all enemies!
2013-03-04 23:57:44.800 StrategyPatterns[22420:c07] Character 3: Attack all enemies!
2013-03-04 23:57:44.800 StrategyPatterns[22420:c07] Character 1: Attack all enemies!
2013-03-04 23:57:44.801 StrategyPatterns[22420:c07] Character 2: Healing Character
1!
2013-03-04 23:57:44.801 StrategyPatterns[22420:c07] Character 3: Protecting
Character 2!

```

11.18 Паттерн Command

Стоять, лежать, сидеть – все это команды, которые нам очень часто давали учителя физкультуры. Так как это очень часто происходит в нашей жизни, глупо было бы предполагать, что кто-нибудь не придумает шаблон с одноименным названием.

Итак, паттерн – команда – позволяет инкапсулировать всю информацию, необходимую для выполнения определенных операций, которые могут быть выполнены потом, используя объект команды.

Образно говоря, если взять пример учителя физкультуры, родители давным давно инкапсулировали в нас команду "Сидеть", потому учитель использует ее чтобы мы сели, не объясняя при этом как это сделать.

Когда использовать паттерн:

Ответ один, и выходит он из описания, когда вы хотите инкапсулировать определенную логику в отдельный класс команду. Отличный пример – do/undo операции. У вас, вероятнее всего, будет так называемый CommandManager, который будет запоминать что делает команда, и при желании отменять предыдущее действие если выполнить команду undo (кстати, это может быть и просто метод).

Собственно, есть два пути реализации этого паттерна: Для начала создадим базовую команду:

```

@interface BaseCommand : NSObject
-(void) execute;

```

```
-(void) undo;
@end
```

Как видим у нашей команды два метода – сделать, и вернуть обратно изменения.

Теперь реализации наших команд:

```
@interface FirstCommand : BaseCommand
{
@private NSString *_originalString;
@private NSString *_currentString;
}
-(id) initWithArguments: (NSString *) anArgument;
-(void) printString;
@end
@implementation FirstCommand
-(id) initWithArguments: (NSString *) anArgument
{
self = [super init];
_originalString = anArgument;
_currentString = anArgument;
return self;
}
-(void) execute
{
_currentString = @"This is a new string"; [self printString];
NSLog(@"Execute command called");
}
-(void) undo
{
_currentString = _originalString; [self printString];
NSLog(@"Undo of execute command called");
}
-(void) printString
{
NSLog(@"Current string is equal to %@", _currentString);
}
@end
```

Как видим, наша первая команда просто умеет менять одну строчку. При этом всегда хранит оригинал, чтобы можно было отменить изменение.

Вторая наша команда:

```
@interface SecondCommand : BaseCommand
{
@private int _originalNumber;
@private int _currentNumber;
}
-(id) initWithArgs: (int) aNumber;
-(void) printNumber;
@end
@implementation SecondCommand
-(id) initWithArgs: (int) aNumber
{
self = [super init];
_originalNumber = aNumber;
_currentNumber = aNumber;
return self;
}
```



```

-(void) execute
{
    _currentNumber++;
    [self printNumber];
}
-(void) undo
{
    if (_currentNumber > _originalNumber)
        _currentNumber--; [self printNumber];
}
-(void) printNumber
{
    NSLog(@"current number is %i", _currentNumber);
}
@end

```

Вторая команда делает то же самое, но с числом.

Давайте теперь создадим объект который будет получать команду и выполнять ее:

```

@interface CommandExecutor : NSObject
{
    @private NSMutableArray *_arrayOfCommands;
}
-(void) addCommand:(BaseCommand *) aCommand;
-(void) executeCommands;
-(void) undoAll;
@end
@implementation CommandExecutor
-(id) init
{
    self = [super init];
    _arrayOfCommands = [[NSMutableArray alloc] init];
    return self;
}
-(void) addCommand:(BaseCommand *) aCommand
{
    //id<CommandProtocol> item = aCommand;
    [_arrayOfCommands addObject:aCommand];
}
-(void) executeCommands
{
    for (BaseCommand *command in _arrayOfCommands)
    {
        [command execute];
    }
}
-(void) undoAll
{
    for (BaseCommand *command in _arrayOfCommands)
    {
        [command undo];
    }
}
@end

```

Как видим, наш менеджер может получать очередь команд, и выполнять их все, или даже отменять все действия (пример простой и с багами). Итак, наш тестовый код:

```

CommandExecutor *commandE = [[CommandExecutor alloc] init];

```

```
BaseCommand *cmdF = [[FirstCommand alloc]
initWithArguments:@"This is a test string"];
BaseCommand *cmdS = [[SecondCommand alloc] initWithArgs:3];
[commandE addCommand:cmdF]; [commandE addCommand:cmdS];
[commandE executeCommands]; [commandE undoAll];
```

И конечно же лог:

```
2013-03-06 22:40:47.392 CommandPattern[9871:c07] Current string is equal to This is a new
string
2013-03-06 22:40:47.393 CommandPattern[9871:c07] Execute command called
2013-03-06 22:40:47.393 CommandPattern[9871:c07] current number is 4
2013-03-06 22:40:47.394 CommandPattern[9871:c07] Current string is equal to This is a test
string
2013-03-06 22:40:47.394 CommandPattern[9871:c07] Undo of execute command called
2013-03-06 22:40:47.395 CommandPattern[9871:c07] current number is 3
```

Второй метод реализации паттерна – это уже использование внутренностей самой Cocoa – NSInvocation:.

NSInvocation – это объект, который можно использовать чтобы передать возможно вызова метода одного класса – другому, и при это передать в него несколько аргументов, а так же объект который вызывал этот метод.

Давайте напишем в наш CommandExecutor добавим два метода и одно приватное поле:

```
//private field
@private NSInvocation *_specificCommand;
//two methods:
-(void) setSpecificCommand:(NSInvocation *)aCommand
{
    _specificCommand = aCommand;
}
-(void) executeSpecificCommand
{
    [_specificCommand invoke];
}
```

Как видим, теперь наш объект сохраняет объект типа NSInvocation и может его запустить, когда требуется. Давайте теперь в нашем основном контроллере напишем функцию, которую мы будем вызывать:

```
-(void) methodInMainController:(int) aFirstArgument
andString:(NSString
*)aStringArgument
{
    NSLog(@"Method called with first argument = %i and second
argument = %@",
aFirstArgument, aStringArgument);
}
```

А теперь создадим объект типа NSInvocation который и будет в результате нашей командой:

```
NSMethodSignature *signature = [self
methodSignatureForSelector:
@selector(methodInMainController:andString:)];
NSInvocation *invocationToPass = [NSInvocation
```

```

invocationWithMethodSignature:signature];
[invocationToPass setTarget:self];
[invocationToPass
setSelector:@selector(methodInMainController:andString:)];
int intArgument = 3;
NSString *stringArgument = @"This is a string argument";
[invocationToPass setArgument:&intArgument atIndex:2];
[invocationToPass setArgument:&stringArgument atIndex:3];
CommandExecutor *executor = [[CommandExecutor alloc] init];
[executor setSpecificCommand:invocationToPass];
[executor executeSpecificCommand];

```

Как видим, в самом начале мы создаем объект, который хранит сигнатуры нашего метода, и создаем объект `NSInvocation`. После этого мы передаем в него аргументы – не стоит удивляться, что индекс начинается с цифры 2 – `index 0` и `index 1` зарезервированы под `target` и `selector`.

Конечно, лог:

23:18:25.624 CommandPattern[10479:c07] Method called with first argument = 3 and second argument = This is a string argument

11.19 Паттерн Flyweight

Flyweight – паттерн который помогает нам отделять определенную информацию, для того, чтобы в будущем делиться этой информацией с многими объектами.

Как пример, возьмем любую стратегическую игру – представьте что у вас одна тысяча солдат одного типа и если каждый будет обращаться к диску и пробовать подгрузить картинку с диска, то вероятнее всего у вас или память закончится или производительность будет желать лучшего.

Когда использовать этот паттерн?

1. У вас очень много однотипных объектов в приложении.
2. Много объектов сохранены в памяти, от чего производительность вашего приложения страдает.
3. Вы видите, что несколько объектов, которые могут быть размножены – спасут вас от создания многих других объектов

Пусть мы пишем игру, где есть два типа персонажей – гоблины и драконы. Для начала создадим базовый класс для всех юнитов:

```

@interface BasicUnit : NSObject
@property (nonatomic, strong) NSString *name;
@property (nonatomic) int health;
@property (nonatomic, strong) UIImage *image;
@end

```

Как видим у каждого юнита есть свойство `image`, которое является типом `UIImage` и может потребовать подгрузки картинки для каждого юнита. Как же сделать загрузку только однажды? С этим и справится наш паттерн.

```

@interface FlyweightImageFactory : NSObject
+(UIImage *) getImage:(NSString *)imageName;
@end
@implementation FlyweightImageFactory
NSMutableDictionary *_imageDictionary;
+(UIImage *)getImage:(NSString *)imageName
{
if (!_imageDictionary)

```

```

_imageDictionary = [[NSMutableDictionary alloc] init];
if (![_imageDictionary objectForKey:imageName])
{
    [_imageDictionary setObject:[UIImage imageNamed:[NSString
alloc]
initWithFormat:@"%@.jpeg",imageName]] forKey:imageName];
NSLog(@"Loading image of the %@", imageName);
}
return [_imageDictionary objectForKey:imageName];
}
@end

```

Как видим, наш flyweight имеет только один класс метод, который возвращает картинку по имени. Если картинки под таким именем нет в его словаре – то она загружается из бандла, если же есть – то просто передается ссылка на нее. Каждый раз когда картинка загружается из бандла мы логируем сообщение, это сделано для того чтобы увидеть сколько раз происходит подгрузка изображения из бандла.

Теперь нам просто нужно в конструкторе наших юнитов загружать картинку не на прямую, а через наш паттерн:

```

@implementation Dragon
-(id) init
{
    self = [super init];
    self.name = @"Dragon";
    self.health = 150;
    self.image = [FlyweightImageFactory getImage:@"dragon"];
    return self;
}
@end
@implementation Goblin
-(id) init
{
    self = [super init];
    self.name = @"goblin";
    self.health = 20;
    self.image = [FlyweightImageFactory getImage:@"goblin"];
    return self;
}
@end

```

Ну и конечно же тест:

```

NSMutableArray *units = [[NSMutableArray alloc] init];
for ( int i = 0 ; i < 500; i++)
{
    [units addObject:[Dragon alloc] init]];
}
for ( int i = 0 ; i < 500; i++)
{
    [units addObject:[Goblin alloc] init]];
}

```

И как ожидается, хоть мы и создаем 1 тысячу юнитов, лог срабатывает только два раза:

```

11:08:45.002 FlyweightPattern[5595:c07] Loading image of the dragon 2013-03-09
11:08:45.006 FlyweightPattern[5595:c07] Loading image of the goblin

```

11.20 Паттерн Proxy

Что делает прокси-сервер? Многие из нас уверены, что в основном он режет скорость интернета, хотя вероятнее всего он делает еще очень много положительных вещей:

1. Логирует кто куда обращается.
2. Смотрит, чтобы не обращались, куда не следует.
3. Смотрит, чтобы по нашему соединению к нам не обращались.

Все эти активности взяты из головы, но они показывают использование прокси в реальной жизни – давать стандартный доступ к чему-либо, при этом обращая стандартные вызовы к прокси и добавляя свою логику.

Паттерн прокси – подменяет реальный объект, и шлет ему запросы через свои интерфейсы. При этом может добавлять дополнительную логику, или создавать реальный объект, если тот еще не создан.

Как пример – вы можете иметь обычные приложения и приложения премиум пользователей. Например, премиум пользователи могут скачивать файлы на большей скорости, чем обычные пользователи. Потому что, объекту, который отвечает за скачивание файлов в вашем приложении, не обязательно знать про существование разных типов пользователей, вы можете обернуть этот объект в прокси, который в свою очередь знает про таких пользователей, и говорит объекту скачивания, на какой скорости пользователь должен получить файл.

Когда использовать паттерн:

1. Возможно, у вас есть два сервера – тестовый и продуктовый. Когда Вы отлаживаете приложение – скорее всего вы будете пользоваться тестовым сервером, ну а когда компилируете приложение для реализации – скорее всего реальный. Эту логику можно реализовать в прокси.
2. Добавление различных валидаций, и проверок безопасности.
3. Множество других возможных ситуаций.

Давайте создадим пример. Пускай у нас есть объект, который отвечает за скачивание файлов:

```
@interface FileDownloader : NSObject
-(void) slowDownload;
-(void) fastDownload;
@end
@implementation FileDownloader
-(id) init
{
self = [super init]; NSLog(@"Downloader created"); return
self;
}
-(void) slowDownload
{
NSLog(@"Sloooooowly downloading...");
}
-(void) fastDownload
{
NSLog(@"Shuuuuuh, already downloaded...");
}
@end
```

Как видим, наш объект умеет скачивать быстро и медленно. При этом, ему все равно, какой пользователь и есть ли связь с интернетом.

Давайте создадим наш прокси:

```

@interface FileDownloaderProxy : NSObject
{
    @private FileDownloader *_downloader;
}
@property (nonatomic) bool isPremiumUser;
-(void) slowDownload;
-(void) fastDownload;
@end
@implementation FileDownloaderProxy
-(void) fastDownload
{
    if (!_isPremiumUser)
    {
        [self slowDownload];
        return;
    }
    if (!_downloader)
        _downloader = [[FileDownloader alloc] init];    [self
checkNetworkConnectivity];
    [_downloader fastDownload];
}
-(void) slowDownload
{
    if (!_downloader)
        _downloader = [[FileDownloader alloc] init];    [self
checkNetworkConnectivity];
    [_downloader slowDownload];
}
-(void) checkNetworkConnectivity
{
    NSLog(@"Checking network connectivity...");
}
@end

```

Как видим проки незначительно умнее: Он знает про тип пользователя, и даже если вызвали метод `fastDownload`, но пользователь не премиум – будет вызван метод `slowDownload`.

1. Он умеет проверять доступ к интернету (пусть это и просто выписка лога).
2. Он проверяет, или тип объекта `FileDownloader` создан, и если нет – создает его.

Протестируем:

```

FileDownloaderProxy *proxy = [[FileDownloaderProxy alloc]
init]; [proxy setIsPremiumUser:NO];

[proxy fastDownload];

[proxy setIsPremiumUser:YES]; [proxy fastDownload];

```

Традиционный лог:

```

13:27:50.312 ProxyPattern[10775:c07] Downloader created
2013-03-10 13:27:50.313 ProxyPattern[10775:c07] Checking network connectivity...
2013-03-10 13:27:50.313 ProxyPattern[10775:c07] Sloooooowly downloading...
2013-03-10 13:27:50.314 ProxyPattern[10775:c07] Checking network connectivity...
13:27:50.314 ProxyPattern[10775:c07] Shuuuuuh, already downloaded...

```

11.21 Паттерн Memento

Ах, как же не хватает в жизни таких штук, как Quick Save и Quick Load. На худой конец Ctrl + Z. Частенько, такой функционал очень полезен для реализации в приложении. Очень правильно также защитить наше записанное состояние от других классов, чтобы в них не смогли внести изменения.

Итак, что же за паттерн такой? Memento – паттерн, который позволяет, не нарушая инкапсуляцию, зафиксировать и сохранить внутреннее состояние объекта, чтобы позже восстановить его состояние.

Состояние, как таковое, может сохраняться как в файловую систему, так и в базу данных. Яркий пример использования: можно сворачивать и выключать ваше приложение. Во время выключения приложения вы можете сохранить все данные с формы, или настройки, или еще что вам угодно в базу данных через CoreData, чтобы восстановить при включении.

Когда использовать паттерн:

1. Вам необходимо сохранять состояние объекта как слепок (snapshot) за определенный период
2. Вы хотите скрыть интерфейс получения состояния объекта.

В данном паттерне используется три ключевые объекта: Caretaker (объект который скрывает реализацию сохранения состояния объекта), originator (собственно сам объект) и конечно же сам Memento (объект который сохраняет состояние originator).

Небольшой пример:

```
@interface OriginatorState : NSObject
@property (nonatomic) int intValue;
@property (nonatomic) NSString *stringValue;
@end
@implementation OriginatorState
@end
```

Допустим, у нас есть состояние, в котором всего лишь два значения – целочисленное и строка.

```
@interface Originator : NSObject
{
    @private OriginatorState *_localState;
}
-(void) changeValues;
-(OriginatorState *) getState;
-(void) setState:(OriginatorState *)oldState;
@end
@implementation Originator
-(id) init
{
    self = [super init];
    _localState = [[OriginatorState alloc] init];
    _localState.intValue = 100;
    _localState.stringValue = @"Hello World!";
    return self;
}
-(void) changeValues
{
    _localState.intValue++;
    _localState.stringValue = [NSString stringWithFormat:@"%d %@",
    _localState.intValue, @"!"];
```

```

NSLog(@"Current state int = %i, string = %@",
    _localState.intValue,
    _localState.stringValue);
}
-(OriginatorState *) getState
{
return _localState;
}
-(void) setState:(OriginatorState *)oldState
{
    _localState = oldState;
    NSLog(@"Load completed. Current state: int = %i, string = %@",
        _localState.intValue, _localState.stringValue);
}
@end

```

Как видим, мы можем изменять состояние объекта состояния, а так же получить состояние и загрузить состояние.

Пусть у нас есть Memento – объект, который будет заведовать состояние нашего объекта:

```

@interface Memento : NSObject
{
    @private OriginatorState *_localState;
}
-(id) initWithState:(OriginatorState *)state;
-(OriginatorState*) getState;
@end
@implementation Memento
-(id) initWithState:(OriginatorState *)state
{
    self = [super init];
    _localState = [[OriginatorState alloc] init];
    [_localState
    setIntValue:state.intValue];
    [_localState
    setStringValue:state.stringValue];
    return self;
}
-(OriginatorState *) getState
{
    return _localState; }
@end

```

То есть наш объект Memento – умеет хранить состояние, и конечно же отдавать состояние:)

Ну и теперь, соединим все это в единое целое, создавая Caretaker:

```

@interface Caretaker : NSObject
{
    @private Originator *_originator;
    @private Memento *_memento;
}
-(void) changeValue;
-(void) saveState;
-(void) loadState;
@end
@implementation Caretaker
-(void) changeValue
{

```



```

if (!_originator)
    _originator = [[Originator alloc] init];
[_originator changeValues];
}
-(void) saveState
{
    _memento = [[Memento alloc] initWithState:[_originator
getState]]; NSLog(@"Saved state. State int = %i and string =
%@", [[_memento getState]
intValue ], [[_memento getState] stringValue ]]);
}
-(void) loadState
{
    [_originator setState:[_memento getState]];
}
@end

```

Как видим Caretaker умеет держать в себе сохраненное состояние (для примера, оно все очень просто, но здесь может быть и стек состояний, и так далее), а так же загружать его.

Давайте протестируем:

```

Caretaker *crtaker = [[Caretaker alloc] init]; [crtaker
changeValue];
[crtaker saveState]; [crtaker changeValue]; [crtaker
changeValue]; [crtaker changeValue]; [crtaker loadState];

```

Лог как пример работы паттерна:

```

23:23:30.711 MementoPattern[14985:c07] Current state int = 101, string = Hello World! !
2013-03-11 23:23:30.712 MementoPattern[14985:c07] Saved state. State int = 101 and
string = Hello World! ! 2013-03-11 23:23:30.712 MementoPattern[14985:c07] Current state int
= 102, string = Hello World! !!
2013-03-11 23:23:30.713 MementoPattern[14985:c07] Current state int = 103, string =
Hello World! !!!
2013-03-11 23:23:30.713 MementoPattern[14985:c07] Current state int = 104, string = Hello
World! !!!!
2013-03-11 23:23:30.713 MementoPattern[14985:c07] Load completed. Current state:
int = 101, string = Hello World! !

```

11.22 Архитектурный шаблон проектирования MVC

В главе описывается популярный архитектурный шаблон MVC (модель-представление-контроллер) с точки зрения программирования для интернета. Статья ориентирована на начинающих девелоперов [6].

Впервые услышав такие слова, как **Model View Controller**, немного не ясно, что это такое, поскольку, не зная ничего о шаблонах проектирования, разобраться в конкретном их случае, достаточно сложно.

Новичку, пожалуй, сразу начинать с рассмотрения вопросов шаблонов не следует, но по мере продвижения в области разработки ПО, уделить внимание этой области, так или иначе, необходимо.

Представьте MP3 проигрыватель, например iTunes. Интерфейс программы используется для добавления новых песен, управления списками воспроизведения и т.д. Проигрыватель ведет базу данных с названиями и информацией о песнях и воспроизводит их, причем в процессе воспроизведения пользовательский интерфейс постоянно обновляется: в нём выводится название текущей песни, позиция воспроизведения и т.д. В основе этой модели заложен паттерн Модель-Представление-Контроллер. Вы видите как

обновляется информация (название и т.д.) и слышите, как она воспроизводится. Представление обновляется автоматически. Вы хотите воспроизвести песню и работаете с интерфейсом, а ваши действия передаются контроллеру. Контроллер выполняет операции с моделью, «приказывает» модели начать воспроизведение песни. Модель содержит всю информацию состояния, данные и логику приложения, необходимые для ведения базы данных и воспроизведения MP3 файлов. Модель оповещает представление об изменении состояния.

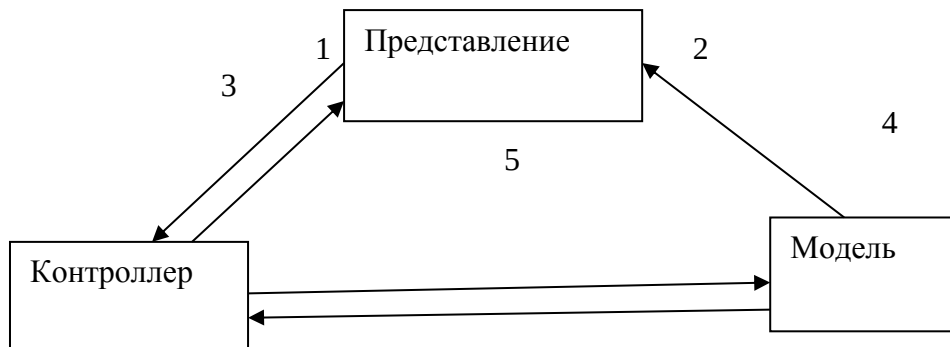


Рисунок 11.1

Контроллер. Получает данные, вводимые пользователем и определяет их смысл для модели.

Модель. Хранит все данные, информацию состояния и логику приложения. Она не знает о существовании представления и контроллера, хотя и предоставляет интерфейс для получения/изменения состояния, а также может отправлять оповещения об изменениях состояния наблюдателям.

Представление. Определяет представление модели (пользовательский интерфейс). Как правило, представление получает состояние и данные для отображения непосредственно от модели.

1 – Пользователь взаимодействует с моделью. Представление – «окно», через которое пользователь воспринимает модель. Когда вы делаете что-то с представлением, например, щелкаете по кнопке воспроизведения, представление сообщает контроллеру, какая операция была выполнена. Контроллер должен обработать это действие.

2 – Контроллер обращается к модели с запросами об изменении состояния. Контроллер получает действия пользователя и интерпретирует их. Если вы щелкаете на кнопке, контроллер должен разобраться, что это значит и какие операции с моделью должны быть выполнены при данном действии.

3 – Контроллер также может обратиться к представлению с запросом об изменении. Когда контроллер получает действие от представления, в результате его обработки он может обратиться к представлению с запросом на изменение, например, заблокировать некоторые кнопки.

4 – Модель оповещает представление об изменении состояния. Когда в модели что-то изменяется, вследствие действий пользователя или других внутренних изменений, например, переходу к следующей песне в списке, модель оповещает представление об изменении состояния.

5 – Представление запрашивает у модели информацию состояния. Представление получает отображаемую информацию состояния непосредственно от модели. Например, когда модель оповещает представление о начале воспроизведения новой песни, представление запрашивает название песни и отображает его. Представление также может запросить у модели информацию состояния в результате запроса на изменение состояния со стороны контроллера.

Контроллер является наблюдателем для модели. В некоторых архитектурах контроллер регистрируется у модели и оповещается о её изменениях, например, если

какие-либо аспекты модели напрямую влияют на пользовательский интерфейс (в некоторых состояниях модели отдельные элементы интерфейса могут блокироваться).

Функции контроллера не сводятся к передаче данных модели. Контроллер отвечает за интерпретацию ввода и выполнение соответствующих операций с моделью. Почему нельзя сделать всё в коде представления? Можно, но нежелательно по двум причинам. Во-первых, это усложнит код представления – у него появятся две обязанности: управление пользовательским интерфейсом и логика управления моделью. Во-вторых, между представлением и моделью формируется жёсткая привязка. О повторном использовании кода представления с другой моделью можно забыть. Логическая изоляция представления и контроллера способствуют формированию более гибкой и расширяемой архитектуры, которая лучше адаптируется к возможным изменениям.

Итак, условимся:

Представление. Модуль вывода информации. Это может быть шаблонизатор или что-либо подобное, цель которого является только в выводе информации в виде HTML на основе каких-либо *готовых* данных.

Контроллер. Модуль управления вводом и выводом данных. Данный модуль должен следить за переданными в систему данными (через форму, строку запроса, cookie или любым другим способом) и на основе введённых данных решить:

- Передавать ли их в модель
- Вывести сообщение об ошибке и запросить повторный ввод (заставить модуль представления обновить страницу с учётом изменившихся условий)

Кроме того, контроллер обязан определять тип данных, полученных от модели (есть ли это готовый результат, отсутствие одного, либо сообщение об ошибке) и передавать информацию в модуль представления.

Модель. Модуль, отвечающий за непосредственный расчёт чего-либо на основе полученных от пользователя данных. Результат, полученный этим модулем, должен быть передан в контроллер, и не должен содержать ничего, относящегося к непосредственному выводу (то есть должен быть представлен во внутреннем формате приложения).

Достаточно сложно с первого раза разобраться и понять. На это требуется время и подходящий проект. Но на самом деле ничего сверхсложного в этом нет.

Представим себе в качестве представления какой-либо класс, который с помощью шаблонизатора выводит результат или сообщение об ошибке. На его вход подаётся либо массив с данными (объект или что-либо иное), либо переменная, содержащая текст с ошибкой.

В качестве контроллера будет выступать класс, производящий все необходимые проверки корректности данных и генерирующий сообщения об ошибках. Проверку данных целесообразно поместить именно в класс контроллера, их используют достаточно часто. Как вариант, можно просто наследовать класс контроллера от более общего класса, реализующего проверку входных данных по заданным правилам. Или, если так будет удобнее, включить в класс контроллера класс или серию функций проверки данных.

Этот же класс должен передать данные, полученные в результате работы модели, в класс представления для вывода.

Контроллер имеет не только методы, но и поля. С полями всё просто: это ошибка и результаты вычислений. По умолчанию им задаётся значение *false*, что свидетельствует о том, что пока нет ни ошибки, ни результатов.

В модели может содержаться любое количество полей и методов. Однако два метода должны быть обязательными (или даже один. Как удобнее будет). Разделение функций модели скорее смысловое. Вполне достаточно создать один метод, который будет и считать, и возвращать результат.

Итак, собственно, у нас есть три класса и алгоритм взаимодействия между ними. Суть архитектурного шаблона MVC состоит в том, чтобы чётко разделить представление,

управление и модель системы. Это очень удобно, ведь если что-либо поменяется в одной из частей системы, других частей эти изменения не коснутся.

Или, например, в модели изменить пару расчетных формул. Или в контроллере убрать пару ограничений, или изменить метод приёма-передачи данных. Если же взять в расчёт принципы наследования в ООП, то архитектура **MVC** станет ещё удобнее. Скажем, когда есть две формы, выглядящие одинаково, но с несколько отличающимися алгоритмами расчёта.

Выделим положительные и отрицательные стороны:

К минусам можно отнести:

- Увеличение объема кода.
- Необходимость соблюдения заранее заданного интерфейса.
- Для поддержки разработки требуются более квалифицированные специалисты.

Последнее требование к нашему примеру не относится, но для реальных систем оно весьма актуально.

К плюсам отнесём следующее:

- Несомненно более гибкий код.
- Возможность повторного использования каждой из трёх составных частей MVC.
- Безболезненная замена модели (другие алгоритмы расчета, способа хранения данных и т.д.).
- Достаточно просто перейти от одного представления, к другому (от HTML к XML или JSON).

Надо сказать, что код примера не идеален. В нём есть просторы для рефакторинга (несмотря на то, что он занимает чуть более ста строк). Скажем, в примере участвует всего лишь одна переменная, поступающая от пользователя (name), но что если их будет много? Как их хранить? Как сделать так, чтобы при изменении метода, не пришлось править код более чем в одном месте?

И напоследок: чаще мыслите не кодом, а схемами, шаблонами, абстракциями. Понятно, что это непросто, непривычно и сложно, но учиться этому нужно обязательно.

Отметим, что класс контроллера вдобавок построен по архитектурному шаблону Facade, суть которого состоит в сокрытии внутреннего устройства системы и предоставлении пользователю только функций самого высшего уровня. Эту суть убедительно доказывают две последние строки кода.

UPD0: На самом деле шаблон MVC куда более общий, мы рассмотрели, скорее, его вариант в виде MVP - Model View Presenter, где контроллер есть "медиатор" между представлением и моделью. Более того, в случае реализации MVC в GUI приложениях используют не так называемый Flow-подход (контроллер сам обращается к модели и представлению), а событийный подход с использованием шаблона Observer.

UPD1: Фреймворк CakePHP использует шаблон MVC, запрос к фреймворку изображён на этой схеме. Схема очень близка к той, что обсуждалась выше.

UPD2: В контроллере, *как правило*, лучше использовать динамическое связывание или абстрактную фабрику.

Есть способ выделить контроллер не в виде класса, а в виде модуля:

В таком случае, схема MVC становится более похожей на классическую, но тогда модель должна представлять специальный интерфейс для доступа к данным.

Отметим также, что некоторые фреймворки устроены так, что все три части MVC есть три отдельных файла, которые связываются по названию автоматически.

12 Разработка приложений с графическим интерфейсом (GUI)

12.1 Объекты

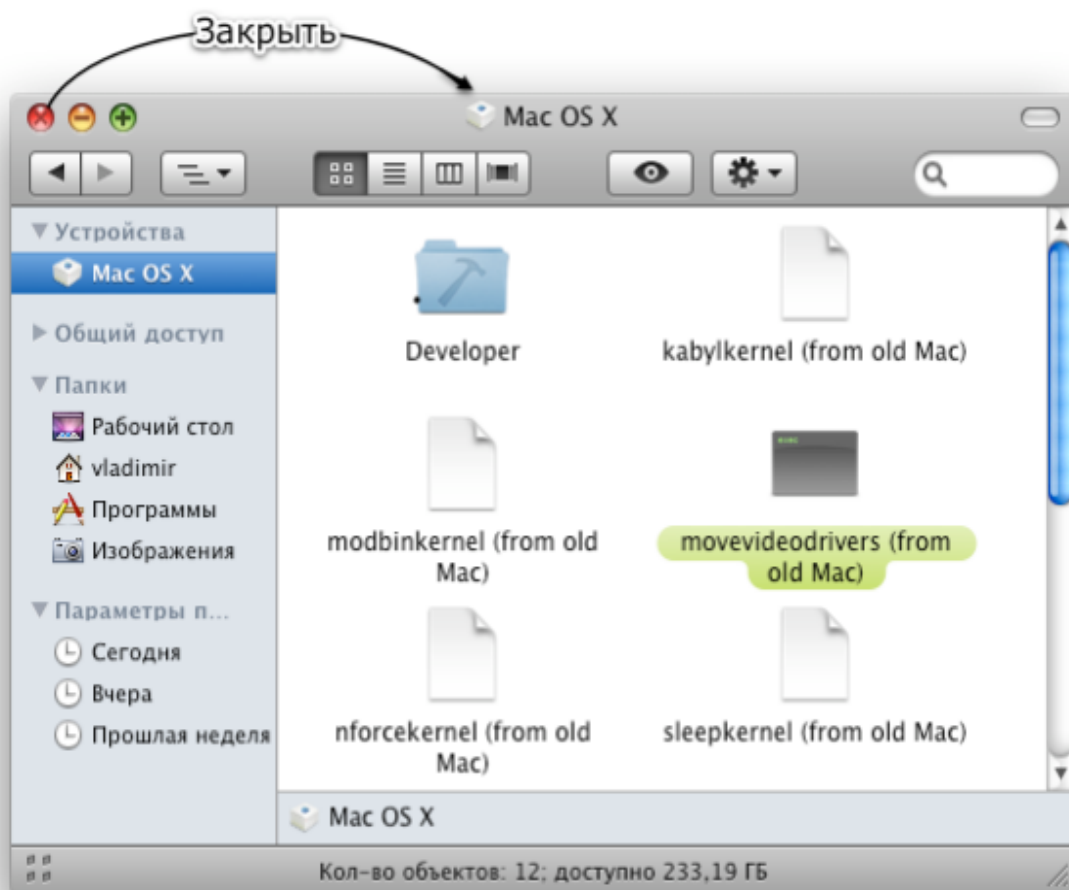
После увеличения наших знаний в Objective-C, мы готовы обсудить, как создать программу с графическим интерфейсом (GUI). Objective-C является фактически расширением языка программирования C. Большинство из того, что мы обсуждали до сих пор, является просто C. Так чем же Objective-C отличается от просто C? Именно в «Objective» части. Objective-C связан с абстрактными понятиями, известными как объекты [7,8].

Давайте взглянем на то, как окна обрабатываются в программах написанных на Objective-C, таких как Finder.

Взглянем на окно Finder в вашем Mac. Вверху слева, есть три кнопки. «Красная» — закрыть. Что же произойдет, если вы закроете окно, нажав красную кнопку? Сообщение будет направлено окну. В ответ на это сообщение, окно выполняет определенный код, с тем чтобы закрыть само себя.

Окно — это объект. Вы можете перемещать его по экрану. Эти три кнопки тоже объекты. Вы можете на них нажать. Эти объекты имеют визуальное представление на экране, но это не обязательно для всех объектов. Например, объект, который обеспечивает соединение между Safari и выбранным сервером не имеет визуального представления на экране.

Объект (например, окно) может содержать другие объекты (например, кнопки).



Сообщение «заккрыть» было отправлено окну.

Вы можете иметь столько окон браузера Safari, сколько вам захочется. Вы думаете, что программисты Apple:

- Запрограммировали каждое из этих окон заранее, используя всю силу своего ума для прогноза, сколько же окон вы захотите открыть;
- Сделали своего рода шаблон и позволили Safari создавать объект окна из него на ходу.

Конечно правильный ответ под номером 2. Они создали некоторый код, называемый классом, который описывает что такое окно, как оно выглядит и ведет себя. Когда вы создаете новое окно, на самом деле этот класс создает его для вас. Этот класс представляет абстрактное окно и любое конкретное окно на самом деле – это экземпляр этого класса (как, например, 76 это просто экземпляр абстрактного класса «число»).

Созданные вами окна располагаются в определенном месте экрана. Если вы свернете окно в Dock, а затем развернете его, оно займет на экране ровно ту позицию что и раньше. Как это работает? Класс определяет поля, предназначенные для хранения положения окна на экране. Экземпляр класса (объект) хранит определенные значения в этих полях. Таким образом, каждый объект хранит свои значения в соответствующих полях и в общем случае для разных окон значения этих полей будут разными.

Класс не только создает объект окна, но и дает ему доступ к набору действий, которые он может выполнять. Одно из этих действий — закрытие окна. Когда вы нажимаете кнопку «закрыть» на окне, кнопка посылает сообщение «закрыть» объекту окна. Действия, которые может выполнять объект называются методами. Как вы можете видеть, они очень похожи на функции и вам не составит большого труда научиться их использованию, если вы будете следовать нашим инструкциям.

Когда класс создает для вас объект окна, он резервирует память (RAM) для сохранения позиции окна и некоторой другой информации. Тем ни менее, он не делает копию кода закрывающего окно. Это было бы потерей компьютерной памяти потому, что этот код одинаков для каждого окна. Код, закрывающий окно должен быть представлен только единожды, но каждый объект окна имеет доступ к этому коду, принадлежащему классу окна.

Код, которой вы увидите в этой главе, содержит строки, которые служат для резервирования и освобождения памяти. Как уже было сказано раньше, мы не будем обсуждать эту сложную тему на этом этапе.

12.2 Приложение с GUI

Сейчас мы напишем программу, содержащую две кнопки и текстовое поле. При нажатии одной кнопки её значение будет введено в текстовое поле. При нажатии на вторую кнопку в текстовое поле будет введено другое значение. Получится программа наподобие двухкнопочного калькулятора, который не может выполнять какие-либо арифметические действия. Конечно, в процессе обучения, вы сможете написать настоящий калькулятор. Эта программа всего лишь один из первых шагов [7].

Если одна из кнопок нашего приложения будет нажата, будет послано сообщение. Это сообщение содержит название метода, который будет выполняться. Сообщение отправляется, хорошо, но куда? В случае окна сообщение послали в тот объект окна, который был экземпляром класса окна. Нам нужен объект, способный к получению сообщения от каждой из этих двух кнопок и способный «сказать» текстовому полю отобразить значение.

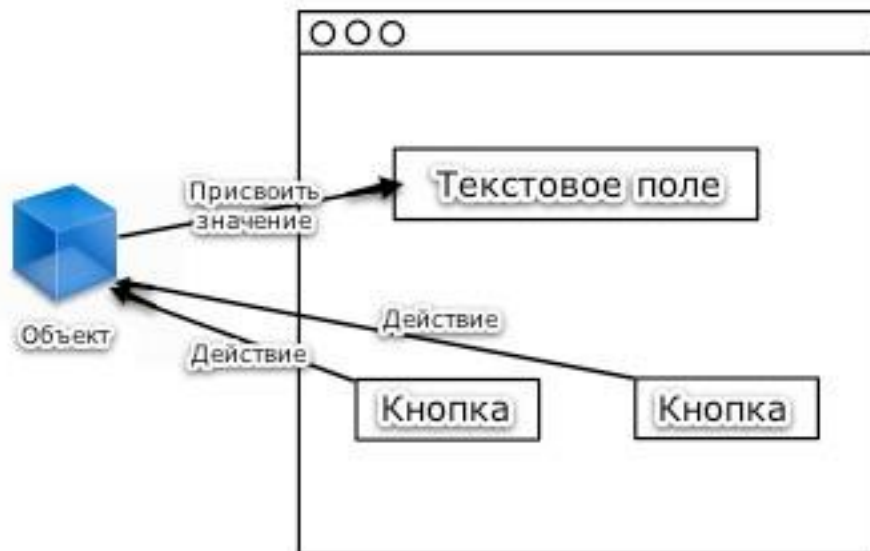


Набросок нашей будущей программы

Для начала создадим наш собственный класс и экземпляр этого класса. Этот объект будет принимать сообщение от кнопок. Также как и окно, наш экземпляр — это объект, но в отличие от объекта окна, мы не увидим наш экземпляр на экране во время выполнения программы. Он всего лишь находится в памяти.

Когда наш экземпляр получает сообщение от одной из двух кнопок, выполняется соответствующий метод. Код этого метода находится в классе (но не в конкретном экземпляре класса). Во время выполнения, этот метод заменит текст в текстовом поле.

Так как же метод одного класса знает как изменить текст в текстовом поле? На самом деле никак. Но это знает само текстовое поле. Таким образом, мы посылаем текстовому полю сообщение с просьбой это сделать. Что же это должно быть за сообщение? Конечно, нам необходимо указать имя получателя (например текстовое поле нашего окна). Также в сообщении необходимо указать, что получатель должен в итоге сделать. Мы определяем это указанием метода, который выполнит текстовое поле по получении сообщения. Конечно, нам нужно знать какие методы текстовое поле вообще способно выполнять и каким образом они вызываются. Также нам нужно сказать текстовому полю какое значение отобразить (в зависимости от нажатой кнопки). То есть, выражение отправки сообщения содержит не только имя объекта и имя метода, но и аргумент (значение), которое должен принимать метод объекта текстового поля.



Приведём набросок обмена сообщениями между объектами нашего приложения.

Здесь представлен общий формат того, как посылаются сообщения в Objective-C, как без аргумента (1.2), так и с ним (1.3):

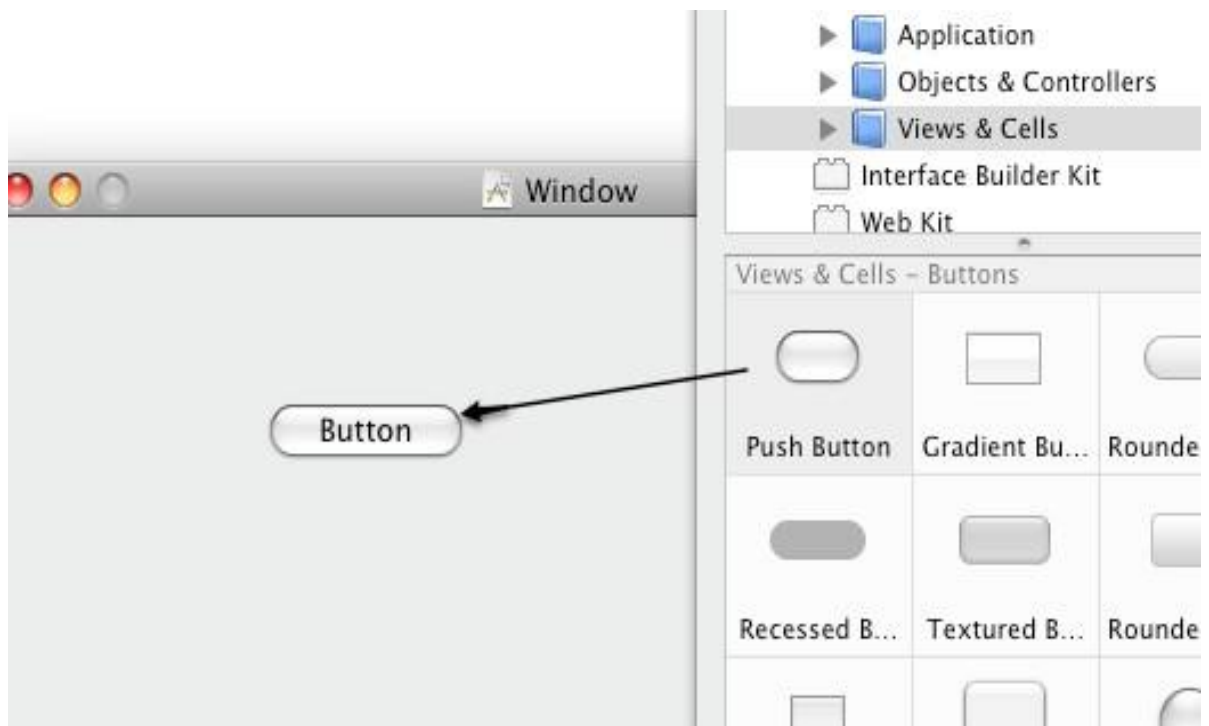
```
//(1)
[receiver message];
[receiver messageWithArgument:theArgument];
```

Как вы можете видеть в каждом из этих блоков, структура, помещённая между квадратными скобками и завершающей точкой с запятой, является последним штрихом. Между скобками, объект-получатель упомянутый первым, следует за именем одного из его методов. Если вызываемый метод требует одно или несколько значений, они должны быть так же указаны (1.3).

12.3 Создание проекта

Давайте взглянем, как это всё работает в действительности. Запустите Xcode для создания нового проекта. Выберите Cocoa Application под заголовком приложения, дайте название проекту (по существующему соглашению, названия приложений с GUI должны начинаться с заглавной буквы). В появившемся фрейме «Groups & Files» окна Xcode откройте каталог Resources. Дважды нажмите на MainMenu.xib (в более ранних версиях Xcode файл, хранящий ресурсы GUI, имел расширение nib — примечание редактора).

Запустится другая программа — Interface Builder. Так как появится много окон, вы можете выбрать пункт Hide Others (Interface Builder ► Hide Others ). Перед вами будут несколько окон. Первое, названное «Window», как раз то, которое будут видеть пользователи вашего приложения. Оно немного великовато, так что можете изменить его размер, как вам нравится. Справа от этого окна находится окно с названием «Library». Это хранилище для всех видов объектов, которые вы можете использовать в своём GUI. Выберите элемент «Views & Cells» из списка в верхней части этого окна и перетащите две кнопки Buttons на ваше окно GUI — «Window». Аналогичным способом перетащите текстовую метку с текстом «Label».



Вдаваясь в подробности, при действии по перетаскиванию кнопки из палитры на окно вашего приложения, происходит создание объекта новой кнопки и помещение его в ваше окно. То же самое происходит для текстового поля и любого другого объекта, который вы можете перетащить из окна палитры.

Имейте в виду, что если держать курсор над иконкой окна палитры, будет отображено имя, такое как `NSButton` или `NSTextView`. Это имена классов, созданных Apple. Позже в этой главе мы покажем как найти методы этих классов, требуемые нам для выполнения необходимых действий в нашей программе.

Выравнивайте объекты, которые вы перетаскили на окно «Window» так, как вам нравится. Изменяйте их размеры, как душе угодно. Для изменения текста объекта кнопки дважды нажмите на нём мышью. Я предлагаю вам исследовать окно палитры как следует, после того как мы закончим этот проект, для того чтобы закрепить навыки добавки остальных объектов на ваше окно.

12.4 Обзор Interface Builder

Для изменения свойств объекта, выберите его и в инспекторе объектов отобразятся его свойства (открыть инспектор объектов — `Tools → Inspector ↑%I`). Рассмотрите их как следует. Например, выберите кнопку «Button» (вы можете увидеть как оно становится выбранным в окне `xib`). Если в строке названия отобразилось «Button attributes», то вы можете изменить свойства данного объекта. Например, указать новый заголовок у кнопки, указав новое значение в поле «Title». Вы увидите, что можно изменять внешний вид вашего приложения в широком диапазоне, без единой строчки написанного кода [7].

Как мы обещали выше, мы создадим класс. Но перед этим, давайте внимательнее изучим, как работают классы.

Для сохранения кучи сил, было бы замечательно создавать на основе того, что уже было создано и написано другими программистами, вместо того чтобы вручную разрабатывать с самого начала. Если вы, например, хотите создать окно с особыми свойствами (или возможностями), вам всего лишь требуется добавить код, реализующий эти возможности, и не нужно писать код, для обработки остального поведения окна, такого как закрытие или минимизация окна. Создавая на основе того что уже написано другими программистами, вы можете наследовать все реализованные ими типы поведения окна совершенно бесплатно. Этот факт отличает Objective-C от чистого C.

Как же это сделано? Итак, существует класс окна (NSWindow), и вы можете написать класс, который наследуется от этого класса.

Возможно, вы добавите какое-то определённое поведение в ваш собственный класс окна. Что же произойдет если ваше окно получит сообщение «Close»? Ведь вы не писали никакого кода для обработки этого сообщения, а так же не копировали код для этих целей в ваш класс. Попросту говоря, если класс вашего окна не содержит кода для определённого метода, сообщение автоматически передается для обработки классу, от которого было унаследовано ваше окно (т.е. его суперклассу). И если необходимо, этот процесс повторяется до тех пор, пока обработчик сообщения не будет найден (или не будет достигнут самый родительский класс всей иерархии наследования в данном случае).

Если метод не может быть найден, то посланное вами сообщение не будет обработано. Это словно поиск гаража для смены шин, когда даже его владелец не в силах оказать помощь вам. В таких случаях Objective-C укажет на ошибку.

А что, если вы хотите использовать свою собственную обработку для метода, уже унаследованного от родительского класса? Это легко, вы имеете возможность переопределить необходимые методы. Например, вы можете написать код, который при нажатии кнопки закрытия окна, перемещает его из области видимости, ещё до действительного его закрытия. Ваш особый класс окна может использовать те же имена методов для закрытия окна, что определены Apple. Итак, когда ваше окно получает сообщение о закрытии, выполняется ваш переопределённый метод, вместо метода определённого Apple. Таким образом, окно может убираться из области видимости, до того как оно будет фактически закрыто.

Да, обработка закрытия окна уже реализована программистами Apple. Внутри нашего метода закрытия окна мы легко можем вызывать метод закрытия окна-родителя, хотя порядок его вызова слегка отличается, для того чтобы удостовериться, что наш собственный метод не будет вызываться рекурсивно.

```
//(2)
// Код для перемещения окна из области видимости.
[super close]; // Вызов метода close родительского класса
```

Этот код слишком наворочен для этого вводного буклета, и мы не ждем, что вы тут же вникните в его смысл, прочитав эти несколько строк.

Существует один класс, являющийся родительским для всех остальных, как царь горы, и называется он NSObject. В общем, все классы, которые вы когда-либо создадите или будете использовать, будут являться классами-потомками класса NSObject, прямо или косвенно. Например класс NSWindow является потомком от класса NSResponder, который в свою очередь потомок от NSObject. В классе NSObject определены общие методы для всех объектов (например генерация текстового описания объекта, опрос объекта способен ли он обработать передаваемое сообщение и т. д.). Давайте посмотрим, как создать класс.

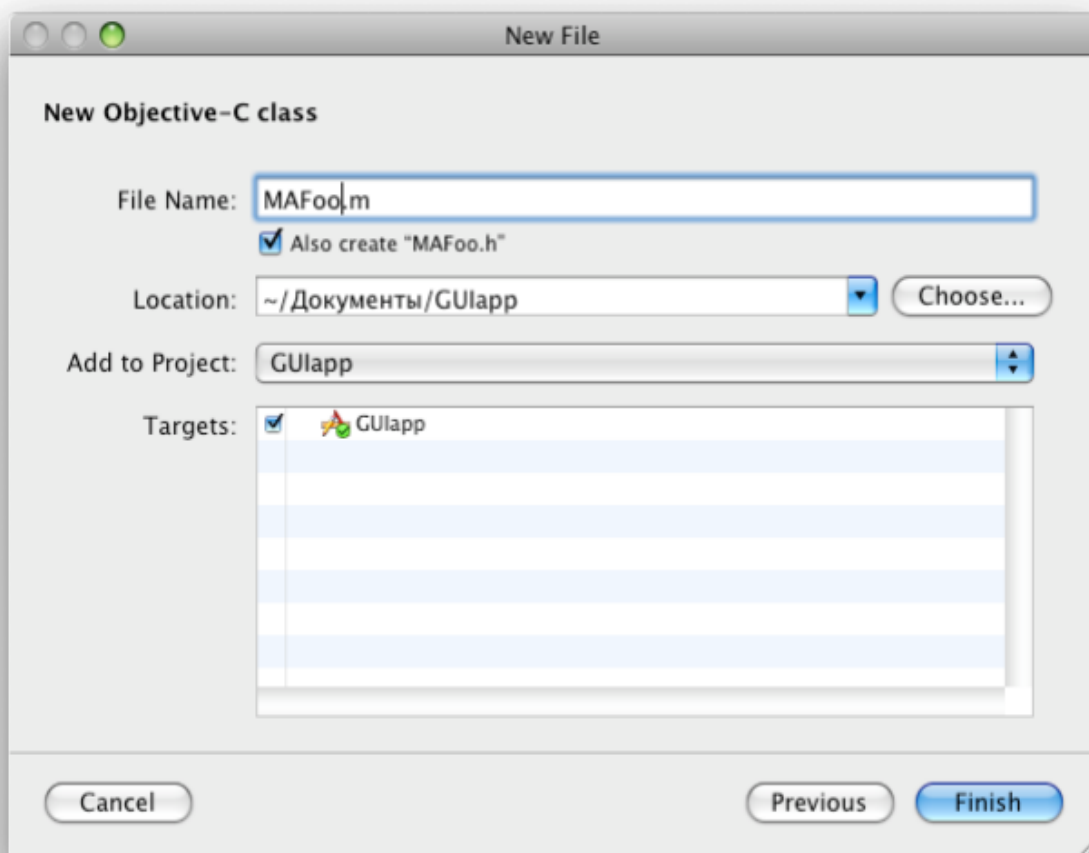
Перейдите на ваш проект в Xcode и создайте новый файл (File → New File... %N). Выберите Objective-C class из списка, затем нажмите кнопку «Next». Задайте имя класса. Здесь он назван «MAFoo» (имя файла класса «MAFoo.m» — примечание редактора). Нажмите кнопку «Finish».

Первые две заглавные буквы «MAFoo» обозначают «My Application». Вы можете назвать класс так, как вам нравится. Как только вы начнете писать свои собственные приложения, мы рекомендуем вам выбрать две или три буквы, которые вы будете использовать для всех классов, чтобы избежать беспорядка с существующими названиями класса. Однако не используйте NS, поскольку это может смутить вас потом. NS используется для классов Apple. Оно обозначает NeXTStep, NeXTStep — операционная система, на которой был основана Mac OS X после того, как Apple, Inc. купила NeXT и вернула Стива Джобса в качестве бонуса.

CocoaDev вики содержит список других префиксов, которые нужно избегать. Вы должны проверить их при выборе собственного:

<http://www.cocoadev.com/index.pl?ChooseYourOwnPrefix>

При создании нового класса вы должны дать ему название, которое передает полезную информацию об этом классе. Например, мы уже видели, что в Cocoa класс, используемый для представления окон называется `NSWindow`. Другим примером является класс, который используется для представления цветов, который называется `NSColor`. В нашем случае, `MAFoo` класс мы создаем только здесь, чтобы проиллюстрировать способ объектов общаться вместе в приложении. Именно поэтому мы дали ему общее имя без какого-либо специального смысла.

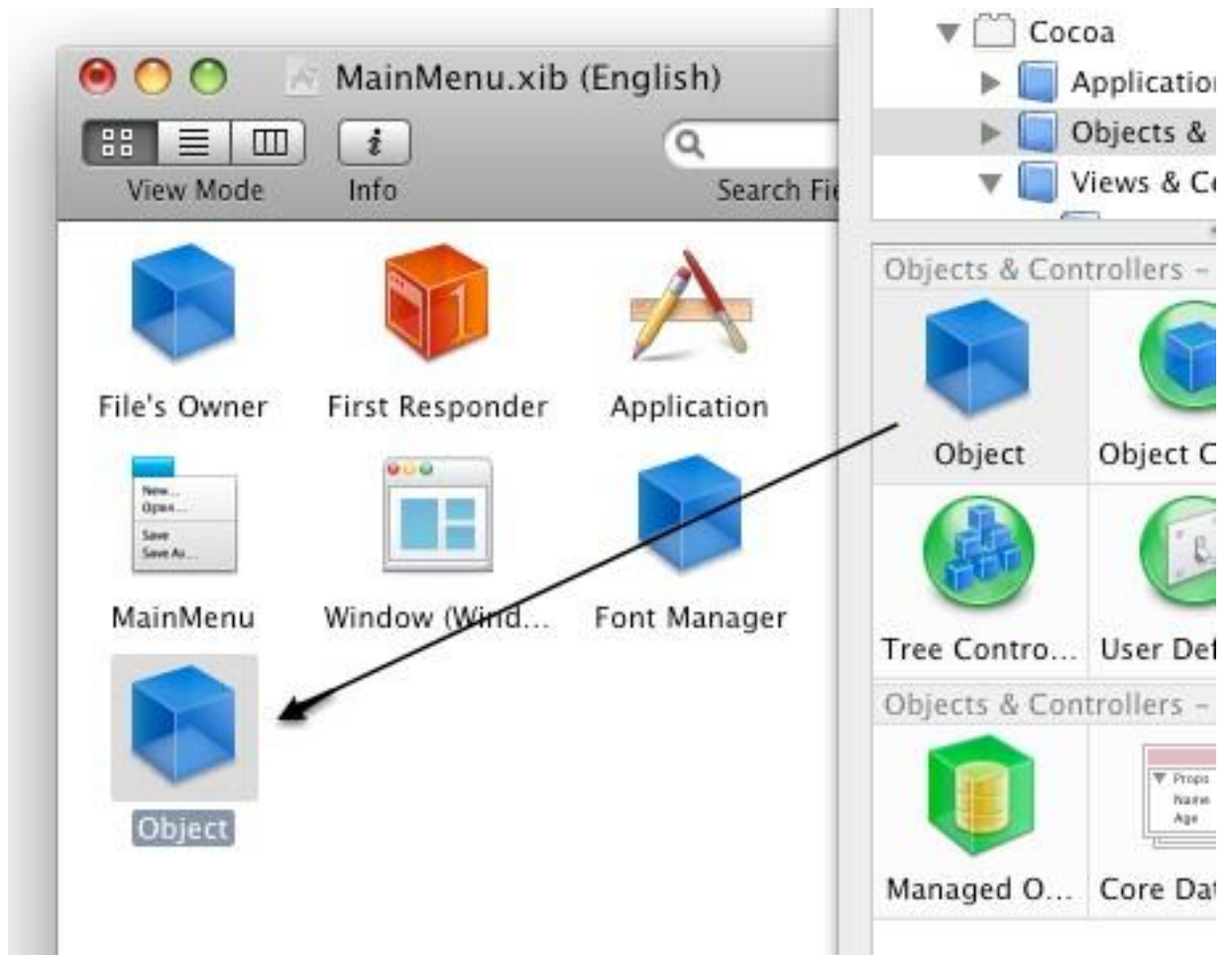


Создание класса «MAFoo»

Вернитесь в Interface Builder, перейдите на палитру «Library» и выберите «Objects & Controllers» из верхнего меню. Теперь тяните Object (синий куб) от палитры до класса MainMenu.xib.

Затем выберите вкладку Identity в инспекторе объектов () и следом отметьте MAFoo из всплывающего меню «Class».

Теперь мы связали MAFoo класс в Xcode с объектом в xib-файле. Это позволит общаться нашему коду и нашему интерфейсу.



Создание нового объекта



Идентификация экземпляра нашего объекта

12.5 Создание связей

Наш следующий шаг заключается в создании связей между кнопками от которых посылаются сообщения до нашего объекта MAFoo. Кроме того, мы собираемся создать соединение от объекта MAFoo к текстовому полю, потому что сообщение должно быть отправлено в текстовое поле. Объект не имеет способа отправлять сообщения к другому объекту, если связи с ним нет. Сделав связи между кнопкой и объектом MAFoo, мы показываем, что кнопка может посылать сообщения на объект MAFoo. С помощью этой

связи кнопки будут иметь возможность отправлять сообщения на MAFoo. Аналогичным образом установление связи от нашего объекта к текстовому полю позволит отсылать ему сообщения.

Давайте еще раз пройдем то, что должно сделать приложение. Каждая из кнопок при нажатии может отсылать сообщение, соответствующее специфическому действию. Это сообщение содержит название метода класса MAFoo, который должен быть выполнен. Сообщение посылается в объект MAFoo — экземпляр класса MAFoo, который мы только что создали. Помните, что сами объектные экземпляры класса не содержат код, который выполняет действие, но классы делают это. Итак, это сообщение, посланное в объект MAFoo, вызывает метод класса MAFoo, чтобы сделать что-нибудь. В нашем случае, посылая сообщение в объект текстового поля. Как и любое сообщение, оно состоит из названия метода, который объект должен будет выполнить. В этом случае, у метода объекта текстового поля есть задача показа значения, и это значение нужно послать как часть сообщения (названное «параметром», помните?), наряду с названием метода, чтобы вызвать его.

Наш класс требует 2 действия (action), которые будут вызваны объектами кнопок. Класс нуждается в одном выходе (outlet), переменной для того, чтобы помнить, какому объекту (т.е. объекту текстового поля) будет послано сообщение.

Удостоверьтесь, что MAFoo выбран в окне MainFile.xib. Затем выберите вкладку Identity в инспекторе объектов (%6). В окне инспектора в секции Actions, нажмите кнопку добавить (нажмите на +, который располагается под списком), чтобы добавить действие (т.е., метод действия) к классу MAFoo. Замените название, заданное Interface Builder более понятным именем (например, вы можете ввести «setTo5:» потому что мы будем программировать этот метод, чтобы отобразить номер 5 в текстовом поле). Добавьте другой метод, и дайте ему название (например «reset:», потому что мы будем программировать его, чтобы отобразить число 0 в текстовом поле). Заметьте, что названия наших методов оканчиваются двоеточием (:). Больше об этом позже.

Теперь в окне инспектора выберите вкладку Outlet, добавьте выход (outlet) и дайте ему название (например, «textField»).

Добавляя действия (actions) и выходы (outlets) к классу MAFoo перед тем, как связать объекты, мы должны дать значащие названия нашим двум кнопкам. Так как первая будет «просить», чтобы наш экземпляр MAFoo показал число 5 в текстовом поле, мы назовем ее «Set to 5» (мы уже знаем, как изменить название кнопки: двойной щелчок на ее имени на экране, и затем вводим новое имя).

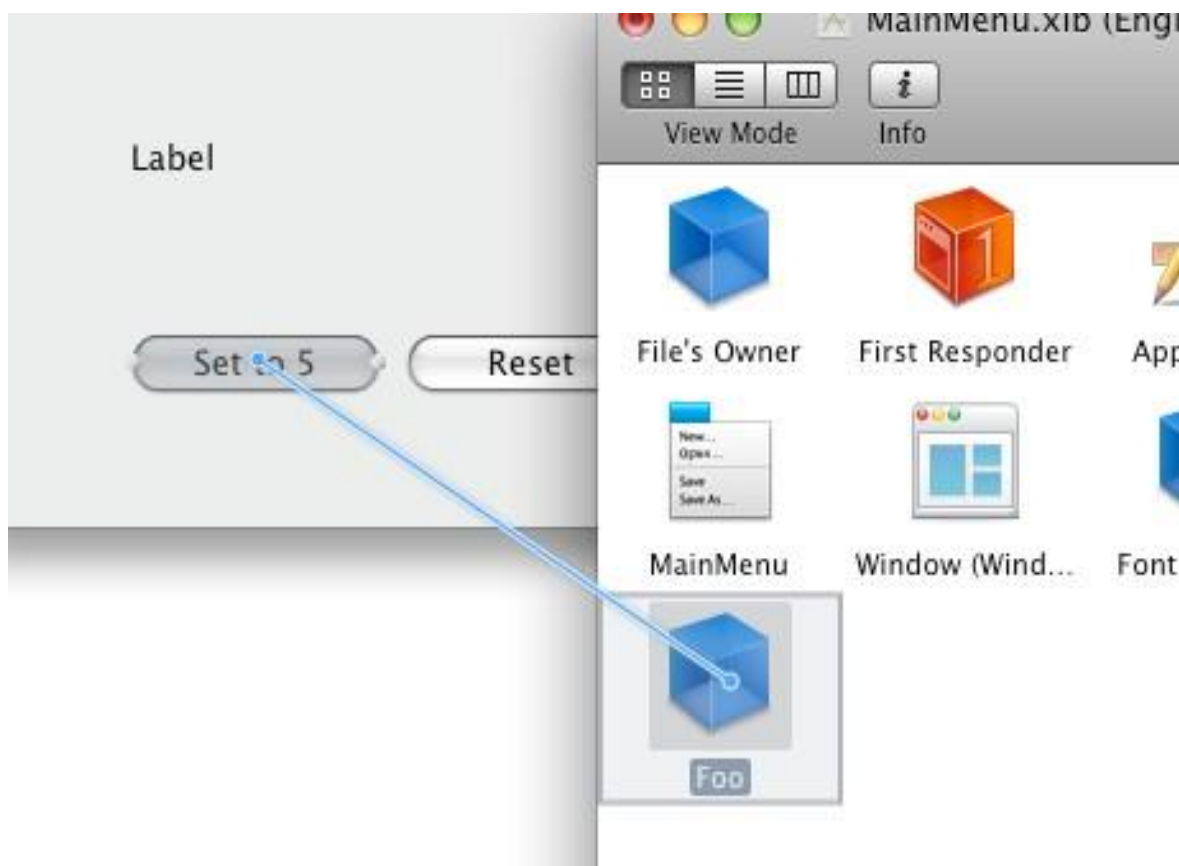
Кроме этого, вторую мы назовем «Reset». Заметим, что называть кнопку специфически не обязательно для работы программы. Это нужно только для того, чтобы наш пользовательский интерфейс был максимально информативным для конечного пользователя.

Теперь мы готовы создать нужные связи между:

- Кнопкой "Reset" и экземпляром MAFoo;
- Кнопкой "Set to 5" и экземпляром MAFoo;
- Экземпляром MAFoo и текстовым полем.

Для создания связи нажмите кнопку ctrl^ на клавиатуре и используйте мышь для перетаскивания от кнопки «Set to 5» до экземпляра MAFoo в окне MainMenu.xib (не делайте этого наоборот!). Линия, представляющая соединение, будет появляться на экране, и меню будет всплывать на иконке представления объекта. Выберите «setTo5:» из списка.

Так же операцию связывания можно произвести просто зажав правую кнопку мыши вместо нажатия ctrl ^ (прим. редактора).



Установка соединения.

Теперь кнопка имеет ссылку на объект MAFoo, и будет направлять ему сообщение «setTo5:» всякий раз при нажатии.

Вы можете связать кнопку «Reset» с объектом MAFoo таким же способом.

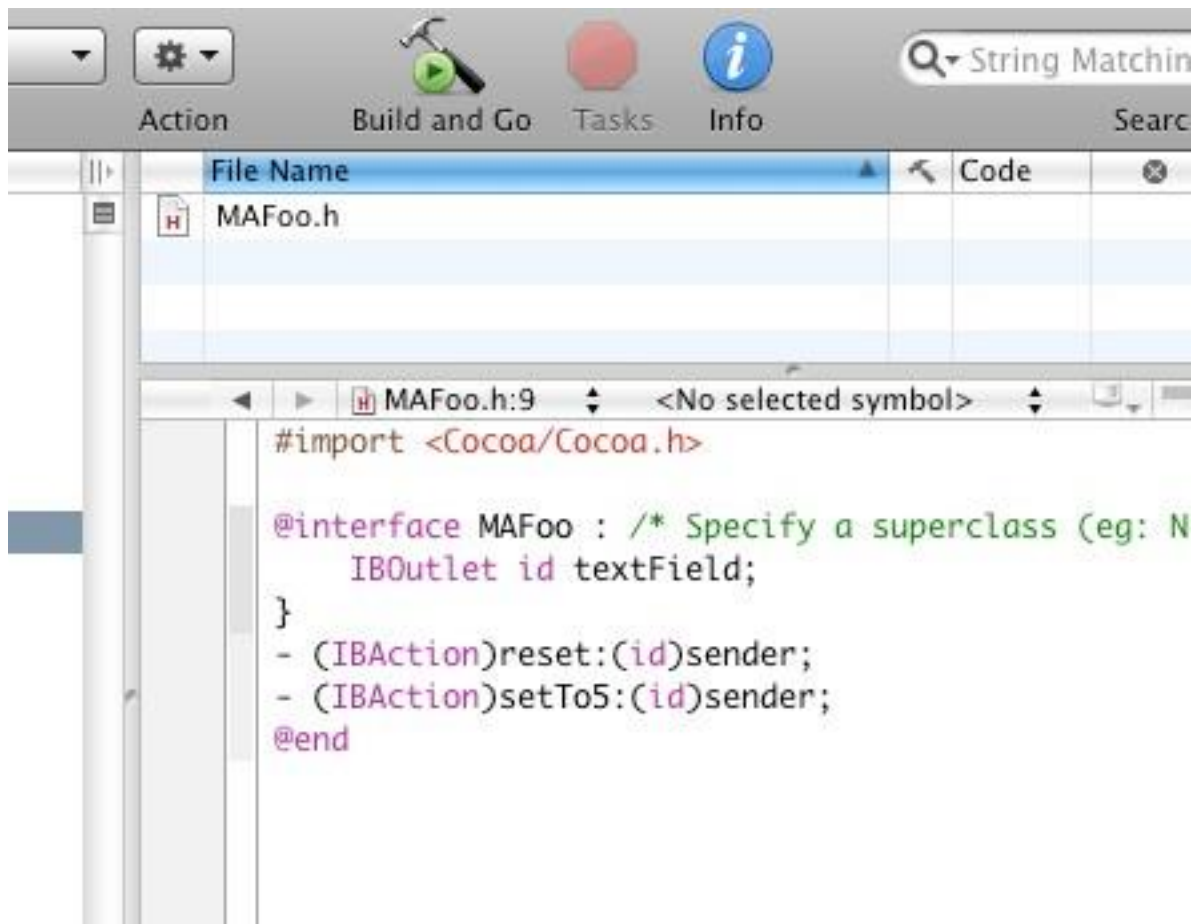
Для создания связи между объектом MAFoo и текстовым полем начните от MAFoo и с нажатой кнопкой `ctrl` перетащите линию до этого поля. Выберите «textField» из меню чтобы присвоить связь.

О чем это все? Хорошо, как вы сейчас увидите, вы только что создали немного кода без написания единой строки.

12.6 Создание кода

Перейдите в меню File в Interface Builder'e и выберите «Write Class Files». Interface Builder спросит у вас, куда конкретно на диск вы хотите поместить сгенерированные файлы. Перейдите в каталог с вашим проектом и замените имеющийся класс MAFoo тем, что вы создали только что [7].

Теперь, если вы переключитесь обратно в Xcode, вы увидите сгенерированные файлы в окне вашего проекта, внутри группы Classes. Нажмите кнопку Editor и затем выберите файл MAFoo.h.



Сгенерированные файлы появились в вашем проекте Xcode.

Один из этих двух файлов, которые мы только что создали, называется MAFoo.h, и это — файл заголовка: он содержит информацию о нашем классе. Например: вы признаете, что есть строка (3.5), содержащая NSObject, в которой говорится, что наш класс наследуется от NSObject.

```
//(3)
/* MAFoo */
#import <cocoa/cocoa.h> // (3.3)

@interface MAFoo : NSObject
{
    IBOutlet id textField; // [57.7]
}
-(IBAction)reset:(id)sender;
-(IBAction)setTo5:(id)sender;
@end
```

Вы видите, что есть один выход (IBOutlet) (3.7) в текстовое поле. Id указывает на объект.

Префикс «IB» — это Interface Builder, его вы использовали для создания этого кода.

IBAction (3.9, 3.10) равнозначен void. Объекту, который посылает сообщение, не возвращается ничего: кнопки в нашей программе не получают ответа от объекта MAFoo в ответ на их сообщение.

Также вы можете видеть, что есть 2 действия (action) программы Interface Builder. Это 2 метода нашего класса. Методы очень похожи на функции, которые мы уже знаем,

но есть различия. Больше о них позже.

Ранее мы видели `#import <Foundation/Foundation.h>` вместо строки (3.3), Это было для приложений без GUI, `#import <Cocoa/Cocoa.h>` же используется для GUI приложений.

Давайте посмотрим второй файл, `MAFoo.m`. Опять мы получаем много кода.

```
//
(4)
#import "MAFoo.h"
@implementation MAFoo

(IBAction)reset:(id)sender // (4.5)
{
}
(IBAction)setTo5:(id)sender
{
}
@end
```

Во-первых, импортирован файл заголовка `MAFoo.h`, таким образом компилятор знает, что ожидать. Могут быть узнаны два метода: `Reset:` и `setTo5:`. Это методы нашего класса. Они подобны функциям в том, что вы должны поместить свой код между фигурными скобками. В нашем приложении при нажатии кнопки посылается сообщение в объект `MAFoo`, прося выполнения одного из методов. Мы не должны написать код для этого. Создание связей между кнопками и объектом `MAFoo` в `Interface Builder` является всем, что нужно. Однако мы должны реализовать эти два метода, т.е. мы должны написать код, который выполняет их функции. В этом случае эти методы только посылают сообщение каждому из экземпляров `MAFoo` для объекта `textField`, и мы их описываем как (4b.7, 4b.11).

```
//(4b)
#import "MAFoo.h"
@implementation MAFoo

- (IBAction)reset:(id)sender
{
[textField setIntValue:0]; // (4b.7)
}
- (IBAction)setTo5:(id)sender
{
[textField setIntValue:5]; // (4b.11)
}
@end
```

Как вы можете видеть, мы посылаем сообщение объекту, на который ссылается выход (outlet) `textField`. Т.к. мы соединили этот выход с текстовым полем, используя `Interface Builder`, наше сообщение будет послано в нужный объект. Сообщение с названием `setIntValue:` вместе с целочисленным значением. Метод `setIntValue:` способен к показу целочисленного значения в объекте текстового поля. В следующей главе мы расскажем, как узнали о нем.

Теперь вы готовы скомпилировать и запустить свое приложение. Обычно нужно нажать кнопку `Build and Go` на панели инструментов `Xcode`. `Xcode` нужно несколько секунд, чтобы собрать и запустить программу. Наконец приложение появится на экране и

вы сможете протестировать его.



Наше приложение работает.

Итак, вы только что создали (очень простое) приложение, для которого сами написали всего 2 строчки кода.

12.7 Поиск методов

В предыдущей главе мы узнали о методах. Мы написали 2 метода сами, но так же использовали 1, предложенный Apple. `setIntValue:` — метод для показа целочисленного значения в текстовом поле. Как мы можем посмотреть доступные методы?

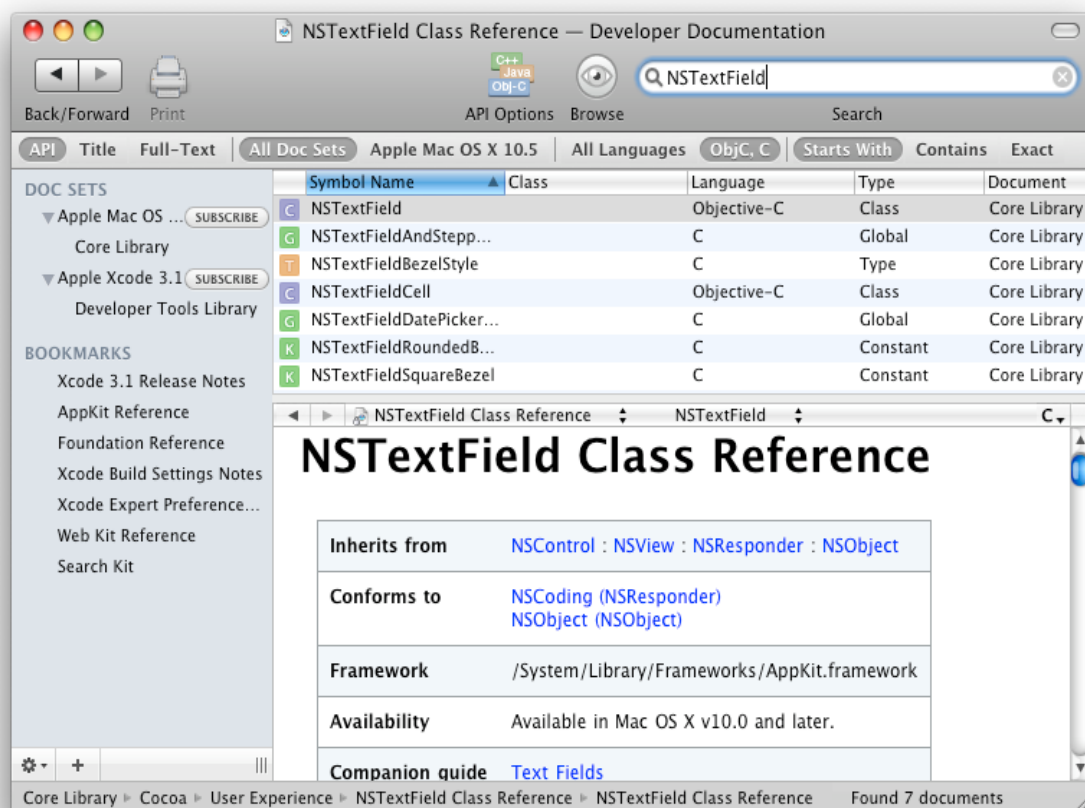
Помните, для каждого использованного вами метода, созданного Apple, вы не должны писать код самостоятельно. И скорее всего, они написаны без ошибок. Поэтому стоит всегда проверять, доступны ли подходящие методы перед написанием своего собственного.

Если вы задержите курсор мыши над объектом в окне палитр в Interface Builder, то появится всплывающая ссылка (tooltip, вероятно). Если вы установите курсор мыши над значком кнопки, то увидите «`NSButton`». Над текстовым полем увидите «`NSTextField`». Каждое из этих названий — имя класса. Давайте проверим класс `NSTextField`, чтобы увидеть, какие методы доступны.

Вернитесь к Xcode и переключитесь на окно документации разработчика (`Help → Documentation %?`). В фрейме слева выберите Apple Mac OS X 10.5, затем введите «`NSTextField`» в поле поиска (убедитесь что выбран режим поиска API). По ходу набора текста в поле поиска, список возможных вариантов существенно уменьшится и вскоре вы увидите появившийся сверху `NSTextField`.

Щёлкните по строке `NSTextField (of type Class)` чтобы получить информацию по классу `NSTextField`, которая отобразится в нижнем фрейме.

Первое, что вы должны учитывать, этот класс наследуется от целого ряда других классов. Последний из них вверху списка это базовый класс `NSObject`.



Навигация по документации Cocoa при помощи Xcode.

Здесь-то мы и начнём наш поиск. Быстрый взгляд по заголовкам скажет нам, что мы не найдем метод, который требуется для отображения значения в текстовом поле объекта. Из-за особенностей принципов наследования, нам нужно рассмотреть непосредственный базовый класс для класса NSTextField, которым является NSControl (если мы и здесь ничего не найдем, то следует просмотреть базовый класс для NSControl, и так далее вверх по иерархии). Так как вся документация выполнена в виде HTML-документа, нам следует щёлкнуть мышью по слову NSControl (как показано выше в пункте Inherits). Это даст нам информацию по классу NSControl.

NSControl наследуется из [NSView](#) : [NSResponder](#) : [NSObject](#).

Как видите, мы перенесли один класс вверх. В списке методов мы видим подзаголовок:

Установка значения контрола (Setting the Control's Value).

То что нам нужно, так это установить значение. Ниже заголовка мы обнаружим:

– [setIntValue](#):

Выглядит многообещающе, так что мы посмотрим описание этого метода щёлкнув по ссылке [setIntValue](#):

[setIntValue](#):

– (void)setIntValue:(int)anInt

Устанавливает значение ячейки назначения (или выбранной ячейки) в целочисленное anInt. Если ячейка в настоящее время редактируется, он прерывает редактирование до установки значения, если ячейки не наследуют [NSActionCell](#) он

отмечает свойства ячейки как требующие обновления(NSActionCell выполняет свои собственные обновления ячеек).

В наших приложениях объект NSTextField является получателем, и мы должны передать ему целочисленное значение. Мы можем увидеть также такую форму записи метода:

```
- (void)setIntValue:(int)anInt
```

В Objective-C знак минуса обозначает начало объявления метода объекта (в противоположность объявлению метода класса, о котором мы поговорим позже). Void показывает то, что никакое значение не возвращается в вызывающий этот метод объект.

Таким образом, если мы посылаем сообщение setIntValue объекту textField, наш MAFoo объект не получает назад никакого значения от объекта текстового поля. И это нормально. После двоеточия (int) говорит нам о том, что переменная anInt должна быть целочисленного типа. В нашем примере мы посылали значение 5 или 0, кои являются целочисленными, и все было замечательно. Иногда немного сложно понять какой же метод использовать в данной ситуации. Тут вы лучше себя будете чувствовать, если глубже ознакомитесь с документацией, так что дерзайте.

А что делать, если вам, например, захотелось прочитать значение из нашего текстового поля объекта textField? Вы помните тот факт, что переменные внутри функций защищены? То же справедливо и для методов. Однако, часто объекты имеют пару близких методов, называемых аксессоры, один для чтения значения, и другой для присваивания значения. Нам уже известно о последнем методе, это setIntValue, первый же выглядит так:

```
/(5)  
- (int) intValue
```

Как видите, этот метод возвращает целое. Таким образом, если мы хотим прочитать целочисленное значение, ассоциированное с нашим текстовым полем, нам нужно послать сообщение вот так:

```
/(6)  
int resultReceived = [textField intValue];
```

Ещё раз заострим внимание на том факте, что в функциях (собственно как и в методах) все имена переменных защищены. Это очень полезно, так как вам не нужно беспокоиться о том, что присвоение значения переменной в одной части вашей программы повлияет на переменную с тем же именем в вашей функции. Однако, имена функций, все же, должны быть уникальными в пределах всей программы. Objective-C обеспечивает защиту на один шаг вперёд: имена методов должны быть уникальными только в пределах одного класса, но разные классы могут содержать одноимённые методы. Это большое подспорье для больших программ, потому что программисты могут писать классы независимо друг от друга, без боязни возникновения конфликтов между именами методов.

Более того, различные методы разных классов могут иметь одинаковые имена, этот принцип называется полиморфизм (от греческого «многоформенность»), и является одной тех особенностей, которая выделяет объектно-ориентированное программирование.

Полиморфизм позволяет вам писать куски кода без знания того, что за классами объектов вы манипулируете. Все это нужно для того, чтобы во время выполнения соответствующие объекты могли обрабатывать сообщения, посланные вами.

Используя преимущества полиморфизма, вы можете писать приложения, являющиеся весьма гибкими и расширяемыми во время разработки. В качестве примера, если мы создали GUI- приложение, затем заменили объект класса текстового поля

объектом другого класса, который может обрабатывать сообщение `setIntValue`, наше приложение все ещё будет работоспособно без необходимости что-то менять в коде или даже без перекомпиляции.

12.8 awakeFromNib

В ранних версиях Interface Builder использовался файл с расширением `nib`. Начиная с версии 3.0 используется файл с расширением `xib`. Данная глава была адаптирована под новую версию Interface Builder [7].

Apple выполнили кучу работы за вас, позволив вам сделать создание приложений ещё более лёгким. В вашем маленьком приложении вам не нужно беспокоиться об отрисовке окна и кнопок на экране и ещё много о чём.

Большинство из этого стало возможным благодаря двум фреймворкам. Это фреймворк Foundation Kit, который мы импортировали в примере (41) главы 3, он выполняет работу по обеспечению большинства сервисов, не связанных с графическим интерфейсом пользователя. Другой фреймворк называется Application Kit, он имеет дело с объектами, которые вы видите на экране, а так же с механизмами взаимодействия с пользователем. Оба фреймворка хорошо документированы.

Теперь же вернемся к нашему GUI-приложению. Допустим, неплохо бы, чтобы наше приложение отображало определенное значение в объекте текстового поля сразу же после запуска и начального отображения окна.

Вся информация для окна хранится в `xib`-файле (`nib` служит для NeXT Interface Builder'a). Это очень хорошая подсказка того, что требуемый нам метод может являться частью Application Kit. Давайте-ка узнаем как получить информацию об этом фреймворке.

Вернитесь к Xcode и переключитесь на окно документации разработчика (Help → Documentation %?). В окне документации убедитесь, что поиск производится по заголовкам (на панели поиска выбран пункт «Title»).

Затем введите «Application Kit» в поле поиска и нажмите Return ←.

XCode предоставит несколько результатов (поиска). Среди них документ «Application Kit Framework Reference».

Внутри документации вы найдете перечень сервисов, которые предоставляет этот фреймворк. В разделе Протоколов (Protocol Reference) есть ссылка «NSNibAwaking». Если вы пройдете по ней, вы найдете документацию по классу NSNibAwaking.

Руководство по протоколу **NSNibAwaking** (неофициальный протокол).

Framework

/System/Library/Frameworks/AppKit.framework

Объявлен в AppKit/NSNibLoading.h

Сопровождающий документ .Загрузка ресурсов (Resource Programming Guide)

Описание протокола

Этот неофициальный протокол состоит из одного метода: `awakeFromNib`. Классы могут использовать этот метод для инициализации информации состояния после того как объекты были загружены из архива Interface Builder (`nib`-файла).

Если мы используем этот метод, он будет вызван после загрузки наших объектов из своего `nib`-файла. Поэтому мы можем использовать его для достижения нашей цели: отображения значения в текстовом поле во время запуска.

Я не хочу сказать, что нахождение правильного метода является всегда тривиальной задачей. Часто требуется некоторое время полистать документацию и с умом использовать ключевые слова для поисков, чтобы найти метод так нужен. По этой причине, для вас очень важно ознакомиться с документацией обоих фреймворков, так чтобы вы знали какие классы и методы вам доступны. Может быть сейчас это вам ещё не

требуется, но оно поможет вам выяснить как заставить свою программу делать то что вы хотите.

Хорошо, мы нашли метод, все что осталось, так это добавить его в наш файл MAFoo.m (7.13).

```
//(7)
#import "MAFoo.h"

@implementation MAFoo
- (IBAction)reset:(id)sender
{
    [textField setIntValue:0];
}
- (IBAction)setTo5:(id)sender
{
    [textField setIntValue:5];
}
- (void)awakeFromNib // [61.13]
{
    [textField setIntValue:0];
}
@end
```

Когда окно открывается, метод `awakeFromNib` будет вызван автоматически. Как результат, текстовое поле покажет ноль когда вы посмотрите на вновь открытое окно.

12.9 Указатели

Эта глава содержит сложные понятия представляющие основу языка C. Эти понятия могут отпугнуть начинающих программистов. Но не бойтесь, если вы не поймёте всё сразу! Понимание принципа работы указателей очень полезно, но к счастью не сильно важно на начальном этапе изучения Objectice-C.

Когда вы декларируете переменную, ваш Мас соединит эту переменную с областью в памяти, для хранения ее значения.

Для примера рассмотрим следующие инструкции:

```
//(8) int x = 4;
```

При запуске, ваш Мас найдет немного незанятого места в памяти и отметит, что в место теперь записана переменная `x` (конечно мы могли и должны были использовать более подходящее имя для нашей переменной). Посмотрите на инструкцию (8) опять. Указание типа переменной (в данном случае `int`) говорит компьютеру сколько памяти требуется для хранения переменной `x`. Если вы укажете тип переменной `long` или `double`, нужно будет выделить больше места.

Инструкция присвоения `x = 4` присваивает значение 4 в зарезервированной памяти. Конечно, компьютер запоминает где в памяти было сохранено значение переменной `x`. Другими словами запоминая адрес переменной `x`. Поэтому, каждый раз, когда вы используете `x` в вашей программе, компьютер «смотрит» в нужном месте (по правильному адресу) и может найти актуальное значение переменной `x`.

Указатель — это просто переменная, хранящая адрес другой переменной.

Адрес переменной можно получить добавлением символа `&` перед переменной. К примеру, чтобы получить адрес переменной `x`, напишите `&x`.

Когда компьютер вычисляет выражение `x`, он возвращает значение переменной `x` (в нашем примере он вернет 4). В противоположность этому, когда компьютер вычисляет выражение `&x`, он возвращает адрес переменной `x`, но не её значение. Адрес это номер который обозначает определённое место в памяти компьютера (что-то типа того как номер комнаты обозначает вполне определённую комнату в отеле).

Указатель можно объявить так:

```
//(9) int *y;
```

Эта команда определяет переменную `y`, которая содержит адрес переменной целочисленного типа (на данный момент она не содержит адрес переменной, т.к. не произведена инициализация — прим. редактора). Повторяю: она не содержит переменную типа `int`, но она содержит адрес в памяти, где лежит эта переменная. Для того чтобы присвоить переменной `y` адрес переменной `x` вам нужно написать следующий код:

```
//(10) y = &x;
```

Теперь переменная `y` указывает на адрес переменной `x`. Используя переменную `y`, вы можете отыскать переменную `x`. Как это сделать.

Вы можете узнать значение, на которое указывает переменная `y` путем добавления звездочки перед указателем. Например, выражение `*y` возвратит значение 4. Оно дает такой же результат, как если бы обратились непосредственно к переменной `x`. Выполнение команды:

```
*y = 5;
```

равносильно выполнению:

```
x = 5;
```

Использование указателей удобно, потому что вы можете захотеть обращаться не к значению переменной, а к её адресу. Например, вы можете захотеть создать функцию, которая увеличивает значение переменной на 1. Что ж, можно это сделать вот так:

```
//(11)
void increment(int x)
{
    x = x + 1;
}
```

Но у вас ничего не получится. Если вызвать эту функцию из программы, вы не получите тот результат, который ожидаете.

```
//(12)
int myValue = 6; increment(myValue); NSLog(@"%d:\n", myValue);
```

Результатом выполнения этого кода будет 6. Почему? Разве вы не увеличили значение переменной `myValue`, вызвав функцию инкремента? Нет. Смотрите, функция (11) берет только значение переменной `myValue`, увеличивает его на единицу и... прекращает выполнение. Функция работает только со значением переменной, которую вы указали в качестве аргумента, а не с переменной, которая хранит это значение. Даже если вы изменили значение `x`, вы изменили только значение переменной, которое получила функция.

Любое такое изменение будет потеряно, когда будет осуществлен возврат из функции. Кроме того x даже не обязательно присваивать переменную. Если вы напишете `increment(5);` какой результат получите?

Если вы хотите написать функцию инкремента, которая действительно работает, то есть принимает в качестве аргумента переменную и изменяет значение этой переменной вам нужно передать в функцию адрес этой переменной. Итак, используйте указатель как аргумент:

```
//(13)
void increment(int *y)
{
    *y = *y + 1;
}
```

затем вы можете сделать, например, такой вызов функции:

```
//(14)
int myValue = 6;
increment(&myValue); // передать адрес
NSLog(@"%d:\n", myValue); // теперь myValue равно 7
```

Список используемых источников

1. Керниган Б.В. Язык программирования С. / Пер. с англ. [Текст] / Б.В. Керниган, Д.М. Ричи – М.: ООО «И.Д.Вильямс», 2009, – 304 с. ISBN: 978-5-8459-0891
2. Кочан Стивен. Программирование на Objective-C 2.0. / Пер. с англ. [Текст] / Стивен Кочан – ЭКОМ Паблишерз, 2010, – 583с. ISBN: 978-5-9790-0131-9
3. Среда разработки Сосоа [Электронный ресурс] – <http://macbug.ru/cocoa/objc>
4. Objective-C с нуля [Электронный ресурс] – <http://touchdev.ru/tag/Objective-C>
5. История Mac OS [Электронный ресурс] – <http://macguide.ru/mac-os-x-history>
6. MVC для начинающих и для интернета в частности [Электронный ресурс] – <http://chtivo.webhost.ru/articles/mvc.php>
7. Bert Altenberg, Alex Clarke, Philippe Mouglin [Электронный ресурс] – <http://www.cocoalab.com>.
8. Марк Дейв. Разработка приложений для iPhone iPad и iPod touch с использованием iOS SDK. / Пер. с англ. [Текст] / Дейв Марк, Джек Наттинг, Джеф Ламарш – М.:ООО «И.Д.Вильямс», 2012. – 624 с. ISBN 978-5-8459-1725-6 (рус.)
9. Малеев Д. Хрестоматия iOS паттернов [Электронный ресурс] – <https://maleevdimka.wordpress.com>.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»
(СГАУ)

Технологии и инструментальные средства разработки программного обеспечения для мобильных устройств

Курс лекций

Семестр А

Солдатова О. П.

Самара

2013

Содержание

1 ОБЗОР ПЛАТФОРМЫ ANDROID	4
1.1 История разработки платформы	4
1.2 Java	4
1.3 Архитектура платформы	5
1.4 Новые функции Android 2.2, предназначенные для разработчиков.....	7
1.5 Android 2.3 (Gingerbread).....	8
1.6 Android 3.0 (Honeycomb)	9
1.7 Android Ice Cream Sandwich	11
1.8 Загрузка приложений из Android Market	12
1.9 Пакеты.....	13
1.10 Android Software Development Kit (SDK)	14
1.11 Краткий обзор объектной технологии.....	15
1.12 Тестирование приложения Doodlz на виртуальном устройстве AVD	17
1.13 Резюме.....	27
2 ИНСТРУМЕНТЫ РАЗРАБОТКИ ANDROID	28
2.1 Введение	28
2.2 Создание нового проекта	29
2.3 Создание графического интерфейса пользователя приложения	
Welcome с помощью визуального макетного редактора модуля ADT	32
2.4 Структура файла main.xml.....	44
2.5 Интегрированная среда разработки Eclipse – советы и хитрости.....	47
2.6 Документация по платформе Android.....	53
2.7 Отладка приложений с помощью DDMS.....	54
2.8 Обзор системы Android	56
2.9 Разработка приложений ANDROID вне ECLIPSE	57
3 ОСНОВЫ ЯЗЫКА JAVA.....	58
3.1 Принципы ООП	58
3.2 Классы и объекты	59
3.3 Пакеты.....	60
3.4 Понятие имени.....	61
3.5 Управление доступом	64
3.6 Создание объектов.....	65
3.7 Конструкторы	66
3.8 Блоки инициализации.....	68
3.9 Методы	69
3.10 Лексические основы языка	74
3.11 Типы данных	78
3.12 Массивы	83
3.13 Операторы	85
3.14 Управление выполнением программы.....	96
3.15 Циклы.....	100

4 СОЗДАНИЕ ПРИЛОЖЕНИЯ ANDROID С ПОМОЩЬЮ JAVA.....	103
4.1 Введение	103
4.2 Создание графического интерфейса приложения	105
4.3 Включение дополнительных функций в приложение	117
4.4 Резюме.....	128
5 РАЗРАБОТКА ПРИЛОЖЕНИЯ ПОИСКОВЫХ ЗАПРОСОВ В ТВИТТЕРЕ	130
5.1 Введение	130
5.2 Тестирование приложения Favorite Twitter Searches	130
5.3 Обзор применяемых технологий.....	133
5.4 Создание графического интерфейса приложения и файлов ресурсов.....	134
5.5 Добавление класса TableLayout и компонентов.....	137
5.6 Создание приложения.....	142
5.7 Файл AndroidManifest.xml.....	154
5.8 Резюме.....	156
6 УПРАВЛЕНИЕ РЕСУРСАМИ ПРИЛОЖЕНИЯ	158
6.1 Ресурсы приложения и системные ресурсы	158
6.2 Конфигурирование файла манифеста ANDROID	166
7 РАЗРАБОТКА ИГРОВОГО ПРИЛОЖЕНИЯ	173
7.1 Проектирование каркаса игрового приложения ANDROID	173
7.2 Реализация анимированного экрана-заставки	178
7.3 Реализация экрана с основным меню	186
7.4 Разработка экрана с инструкциями и экрана с результатами.....	194
8 РАЗРАБОТКА МНОГОПОТОЧНОГО ИГРОВОГО ПРИЛОЖЕНИЯ..	203
8.1 Тестирование приложения	203
8.2 Обзор применяемых технологий.....	204
8.3 Создание графического интерфейса пользователя приложения и файлов ресурсов.....	206
8.4 Создание приложения.....	208
8.5 Резюме.....	230
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ.....	232

1 Обзор платформы Android

1.1 История разработки платформы

Смартфоны, реализованные на платформе Android первого поколения, появились на рынке в октябре 2008 года. Согласно результатам исследований рынка, выполненным компанией Gartner¹, мировые продажи смартфонов на базе Android выросли в девять раз в первом квартале 2010 года по сравнению с первым кварталом предыдущего года. Результаты исследования рынка, выполненные компанией Nielsen², показали, что к марту 2011 года рыночная доля смартфонов Android составила уже 37 % от общего рынка смартфонов, продаваемых в США, тогда как рыночная доля смартфонов Apple iPhone составила 27 %, а рыночная доля смартфонов BlackBerry — 22 %. В августе 2010 года³ ежедневно выполнялась активация более 200 000 смартфонов Android, тогда как всего лишь двумя месяцами раньше активировалось не более 100 000 устройств Android ежедневно. А уже в июне 2011 года ежедневно активировалось более 500 000 устройств Android. В настоящее время на мировом рынке доступно более 300 различных гаджетов Android.

Операционная система Android была разработана компанией Android, Inc. В июле 2005 года акции этой компании приобрела корпорация Google. В ноябре 2007 был сформирован консорциум Open Handset AllianceTM, который изначально объединил 34 компании, а в настоящее время он объединяет уже 81 компанию (www.openhandsetalliance.com/oha_members.html). В задачи этого консорциума входит разработка приложений для платформы Android, внедрение инноваций в технологии, применяемые для разработки мобильных гаджетов, а также повышение удобства работы с устройствами Android при одновременном снижении затрат на производство и снижение цен для потребителей. На основе Android создано множество моделей смартфонов, электронных книг и планшетных компьютеров.

Одно из главных преимуществ платформы Android — ее открытость. Операционная система Android построена на основе открытого исходного кода и распространяется на свободной основе. Это позволяет разработчикам получить доступ к исходному коду Android и понять, каким образом реализованы свойства и функции приложений. Любой пользователь может принять участие в совершенствовании операционной системы Android. Для этого достаточно отправить отчет об обнаруженных ошибках (на веб-странице source.android.com/source/report-bugs.html) либо принять участие в одной из групп дискуссий Open Source Project (source.android.com/community/index.html). В Интернете доступны различные приложения Android с открытым исходным кодом, предлагаемые компанией Google и рядом других производителей

1.2 Java

При разработке приложений Android используется язык программирования Java, являющийся одним из наиболее распространенных. Использование Java стало логичным выбором для платформы Android, потому что это мощный, свободный и открытый язык. На Java разрабатываются полномасштабные корпоративные приложения, расширяется функционал веб-серверов, создаются приложения, предназначенные для пользовательских устройств (мобильных телефонов, пейджеров и персональных цифровых помощников), и это далеко не полный перечень возможных областей его применения.

Java является кроссплатформенным языком. Он позволяет разрабатывать приложения, на зависящие от аппаратных особенностей того или иного устройства. Опытные программисты на языке Java могут быстро освоить создание приложений для платформы Android с помощью Android API и других вспомогательных средств, предлагаемых независимыми производителями.

Открытость платформы способствует быстрому внедрению инноваций. Свыше десяти OEM-производителей из 48 стран, получивших лицензию на производство Android-устройств в 59 странах, выпускают гаджеты на основе платформы Android1. Естественно, что все они конкурируют между собой, что идет на пользу конечному потребителю.

Язык программирования Java является объектно-ориентированным и предоставляет разработчикам возможность получить доступ к мощным библиотекам классов, ускоряющих разработку приложений. Программирование графического интерфейса пользователя является управляемым событиями. В частности, в этой книге приведены примеры приложений, которые реагируют на инициируемые пользователями события, такие как касания экрана или нажатие кнопок. Помимо непосредственного написания кода приложений, можно воспользоваться визуальным конструктором среды Eclipse, позволяющим собирать графический интерфейс из готовых объектов, таких как кнопки и текстовые поля, перетаскивая их в определенные места экрана, добавляя подписи и изменяя их размеры. С помощью интегрированной среды разработки Eclipse и подключаемого модуля ADT (Android Development Tools) можно создавать, тестировать и отлаживать приложения Android, а также заниматься проектированием пользовательского интерфейса.

1.3 Архитектура платформы

Многие современные смартфоны Android сочетают в себе функции мобильных телефонов, интернет-клиентов, MP3-плееров, игровых консолей, цифровых фотоаппаратов и многого другого. Эти устройства упакованы в компактные корпуса и оборудованы полноцветными мультисенсорными экранами. Подобный экран позволяет управлять устройством с помощью касаний и жестов [1].

Мультисенсорный экран упрощает переключение между телефоном, приложениями, музыкальной библиотекой, веб-браузером и т. п. На экране может отображаться клавиатура, предназначенная для ввода сообщений электронной почты и данных приложений (некоторые устройства Android также снабжены физическими клавиатурами). С помощью двух пальцев можно увеличивать масштаб просмотра (путем разведения пальцев) или уменьшать масштаб просмотра (с помощью сведения пальцев) фотографий, видеороликов и веб-страниц. Путем смахивания пальцем экрана по вертикали или по горизонтали можно выполнять прокрутку содержимого этого экрана.

В комплект поставки устройств Android входят различные встроенные приложения, набор которых зависит от устройства. Обычно это приложения Телефон (Phone), Контакты (Contacts), Почта (Mail), Браузер (Browser) и ряд других приложений. Многие производители модифицируют стандартные приложения. В дальнейшем будет продемонстрировано взаимодействие пользователя с приложениями независимо от того, каким образом они были изменены производителем.

Каждая новая версия Android получила наименование от определенного десерта (в алфавитном порядке):

- Android 1.6 (Donut);
- Android 2.0–2.1 (Eclair);
- Android 2.2 (Froyo);
- Android 2.3 (Gingerbread);
- Android 3.0 (Honeycomb).

Версия Android 2.2, которая также называется «Froyo» (замороженный йогурт), появилась в мае 2010 года. В состав этой версии были включены целый ряд новых функций и усовершенствований. В следующих разделах главы будут рассмотрены более новые версии Android 2.3 (Gingerbread, имбирный пряник) и Android 3.0 (Honeycomb, медовые соты).

Усовершенствование Dalvik Virtual Machine привело к увеличению производительности в 2–5 раз по сравнению с Android 2.1. Механизм Chrome V8 ускорил загрузку веб-страниц, содержащих сценарии JavaScript.

Управление памятью ядра позволило улучшить производительность устройств. Пользователи Exchange могут быстро синхронизировать учетные записи Exchange с устройствами Android путем указания имени пользователя и пароля.

Пользователи могут синхронизировать календари Exchange Calendar с приложением Календарь (Calendar).

Доступ к адресам пользователей электронной почты и списков рассылки в пользовательских системах электронной почты Microsoft Exchange, возможность автозавершения имен контактов получателей создаваемых сообщений электронной почты. Пользователи могут создавать пароли, состоящие из букв и цифр, с помощью которых блокируются устройства. В результате предотвращается доступ к заблокированным устройствам и существенно повышается степень безопасности.

Если вы потеряли устройство Android, воспользуйтесь функцией Remote Wipe (Удаленная очистка). Эта функция выполняет сброс устройства к заводским установкам, удаляя при этом все пользовательские данные. Благодаря этому ваши самые личные данные не попадут в руки злоумышленника. При отправке команды удаленной очистки (обычно при помощи специального SMS-сообщения) все ваши данные, которые не были скопированы на резервные носители, будут удалены. Доступность функции Remote Wipe зависит от производителя устройства и диспетчеров политики устройства.

Функция Quick Contact (Быстрый контакт) для устройств Android обеспечивает упрощенный доступ к контактной информации и режимам обмена данными с контактами (например, по электронной почте, с помощью SMS-сообщений или по телефону). Пользователь может коснуться фотографии контакта (например, в списке контактов, галерее фотографий, в окне клиента электронной почты или в календаре), вызвать виджет Quick Contact с различными коммуникационными режимами. Виджет Quick Contact можно включать в состав различных приложений, создаваемых разработчиками.

Элементы управления камерой в Android 2.2 включают функции контроля вспышки и цифрового масштабирования (зума) камеры. Пользователи могут изменять настройки камеры, учитывая условия внешней среды (ночь, солнечный свет, быстрое перемещение), добавлять различные эффекты (сепия, красный либо синий оттенок) и вносить ряд других изменений. Можно запрограммировать предварительный просмотр, настройки съемки, установки выборки и кодирования видео.

Значительно улучшена раскладка клавиатуры. В результате существенно облегчен ввод данных с помощью мультисенсорного экрана. Теперь вы не пропустите нужные клавиши, даже если будете работать с клавиатурой двумя пальцами. Улучшенный словарь Интеллектуальный словарь учитывает статистику использования слов пользователем и в варианты предлагаемых слов включает контакты пользователя.

Усовершенствованный интерфейс браузера включает новую строку адреса, которая обеспечивает выполнение поиска и навигации после ее касания пальцем. С помощью двойного касания веб-страниц осуществляется изменение их масштаба просмотра. Включена поддержка HTML5, обеспечивающего возможность просмотра видео и перетаскивания. Ранее эти возможности были доступны лишь с помощью подключаемых модулей, предлагаемых независимыми поставщиками, например Adobe Flash. (Примечание. Браузер также поддерживает Flash.).

Пользователи могут добавлять раскладки клавиатур для других языков и легко переключаться между ними. Чтобы сменить раскладку, «смахните» клавишу пробела клавиатуры в направлении справа налево. Чтобы добавить клавиатуру (для устройства Android или для эмулятора), выполните команды Settings > Language & keyboard > Android keyboard > Input languages (Настройки > Язык и клавиатура > Клавиатура Android > Языки ввода). Медиафреймворк Android Stagefright обеспечивает воспроизведение видео и

поддерживает прогрессивные HTTP-потоки — отсылка видео браузеру через Интернет с помощью протокола HyperText Transfer Protocol (Протокол передачи гипертекста) и воспроизведение видео даже в процессе его загрузки. Также в Android поддерживается предыдущая версия Медиафрейм- ворк, OpenCORE.

Пользователи могут подключать Android-устройства к другим устройствам Bluetooth, таким как гарнитуры и автомобильные док-станции (предназначены для подключения телефона к автомобильной системе «свободные руки»), передавать сведения о контактах телефонам с поддержкой Bluetooth и выполнять голосовой набор номеров.

В Android 2.x встроена поддержка использования мобильного телефона в качестве точки доступа других устройств к интернету. При этом мобильный телефон выступает в качестве модема и маршрутизатора. Подключение устройств к мобильному телефону может осуществляться как по беспроводному, например WiFi, так и по проводному каналу, например через USB-порт. См. www.engadget.com/2010/05/13/android-2-2-froyotoinclude-usb-tethering-wifi-hotspot-funct/.

1.4 Новые функции Android 2.2, предназначенные для разработчиков

С помощью службы C2DM (Android Cloud to Device Messaging, обмен сообщениями с устройствами через облако) разработчики приложений могут пересылать данные со своих серверов на приложения, установленные на устройствах Android, даже если эти приложения в данный момент не активны. Сервер оповещает приложения о непосредственном подключении к нему для приема обновления или пользовательских данных¹. Путем регистрации в общедоступной учетной записи Android Market можно получить доступ к отчетам об ошибках приложений Android (Android Application Error Reports), в которых находятся отчеты пользователей, создаваемые после аварийного завершения или «зависания» приложений.

В Android 2.2 также появились несколько новых библиотек API, упрощающих добавление различных функциональных свойств в приложения. Некоторые из этих новых фреймворков будут использованы в лекциях. Также используются веб-службы, с помощью которых создаются мэшапы. В этом случае обеспечивается возможность быстрой разработки приложений путем комбинирования веб-служб, используемых в различных организациях, с различными типами вводимой информации (например, RSS, Atom, XML, JSON и др.). Например, на сайте www.housing-maps.com веб-службы применяются для комбинирования списков объектов недвижимости Craigslist (www.craigslist.org) с картами Google Maps (библиотеки API, которые чаще всего применяются при создании мэшапов). В результате пользователь может просмотреть на карте местоположение выбранного дома или квартиры. Веб-службы WeatherBug будут рассматриваться в дальнейшем.

Приложения, которые могут храниться во внешней памяти устройства Android, а не во внутренней памяти (по умолчанию).

Новые функции и возможности, в том числе у API Camera Preview: удвоенная частота смены кадров (20 кадров в секунду), наличие портретной ориентации, элементы управления масштабированием, наличие данных экспозиции и утилиты миниатюр. Новые классы CamcorderProfile применяются в приложениях для определения наличия в пользовательском устройстве реализованного на аппаратном уровне камкордера.

Резервирование данных в «облачном» хранилище с последующим восстановлением после аппаратного сброса устройства к исходным заводским настройкам либо при аварийном восстановлении данных

Создание приложений администратора, предназначенных для управления уровнем безопасности устройства (например, длиной пароля).

Набор графических API, обеспечивающих доступ к OpenGL ES 2.0. Ранее эти API были доступны только с помощью Android NDK — набора инструментов, обеспечивающего использование собственного кода для компонентов приложений,

которые являются критическими к производительности (developer.android.com/sdk/ndk/overview.html).

Набор API, обеспечивающих выбор аудиопотока, автоматическое сканирование файлов в базе данных медиаресурсов (аудио- и видеофайлы), обнаружение завершения загрузки звукового файла, автоматическая пауза и продолжение воспроизведения звука и ряд других возможностей.

Элементы управления режимами (автомобильным, ночным и режимом настольного компьютера) UiModeManager позволяют настраивать пользовательский интерфейс приложения. Библиотека API детектора жестов масштабирования обеспечивает улучшенную обработку событий, а расположенная в нижней части экрана лента TabWidget теперь является настраиваемой.

1.5 Android 2.3 (Gingerbread)

Версия Android 2.3 (Gingerbread — «имбирный пряник»), появившаяся в декабре 2010 года (субверсия Android 2.3.3 с незначительными обновлениями, появилась в феврале 2011), предлагает усовершенствованные пользовательские функции, например, более удобную клавиатуру, улучшенные навигационные функции, более эффективное использование аккумуляторной батареи и ряд других возможностей.

Приложения, которые используют процессорные ресурсы при работе в фоновом режиме, или находятся в неактивном состоянии дольше, чем обычно, могут быть при необходимости принудительно закрыты Android для экономии энергии аккумулятора и повышения производительности системы. Пользователи также могут просматривать список приложений и компонентов системы, потребляющих энергию аккумулятора

С помощью ярлыка Manage Applications (Управление приложениями), доступного в меню Options (Параметры) на экране Home (Главный экран), пользователи могут просматривать все выполняемые приложения. Для каждого приложения можно узнать объем используемого хранилища и памяти, предоставленные приложению разрешения (определяют возможность чтения контактных данных пользователя, создания подключений Bluetooth и других задач), а также выполнить ряд других задач. Пользователи также могут «принудительно завершить выполнение» приложения.

Near Field Communication, NFC (коммуникация ближнего поля) — технология беспроводной высокочастотной связи малого радиуса действия, которая дает возможность обмена данными между устройствами, находящимися на расстоянии около 10 сантиметров. Эта технология — простое расширение стандарта бесконтактных карт, которая объединяет интерфейс смарткарты и считывателя в единое устройство. Устройство NFC может поддерживать связь и с существующими смарткартами и считывателями, и с другими устройствами NFC и таким образом совместимо с существующей инфраструктурой бесконтактных карт, уже использующейся в общественном транспорте и платежных системах. NFC нацелена, прежде всего, на использование в мобильных телефонах.

Вы можете коснуться слова, чтобы выделить его, перетащить маркеры для настройки выделенной области, затем скопировать текст, нажав на выделенную область, после чего вставить текст. Можно также перемещать курсор путем перетаскивания стрелки курсора.

В Android поддерживается протокол Session Initiation Protocol (SIP, Протокол инициализации сеанса) и протокол стандарта Internet Engineering Task Force (IETF). С помощью этих протоколов обеспечивается инициализация и завершение голосовых вызовов, осуществляемых через Интернет. Пользователи, обладающие учетными записями SIP (созданными с помощью инструментов от независимых поставщиков), могут совершать голосовые вызовы других пользователей, имеющих учетные записи SIP, через Интернет. Обратите внимание, что поддержка SIP и вызовов через Интернет присуща далеко не всем устройствам Android. Чтобы получить список SIP-провайдеров, обратитесь

к веб-сайту www.cs.columbia.edu/sip/service-providers.html Пользователи могут получать доступ к файлам, загруженным с электронной почты, браузера или других мест, с помощью приложения Downloads (Загрузки).

В состав платформы также включены многочисленные новые функции для разработчиков, предназначенные для расширенных коммуникаций, разработки игр и мультимедийных приложений. Дополнительные сведения о каждой из этих функций можно найти на веб-сайте developer.android.com/sdk/android-2.3-highlights.html.

Встроены приложения, применяемые для считывания данных из тегов либо устройств NFC и генерирования ответных данных. С помощью приложений Android 2.3.3 можно также записывать теги и работать в одноранговом режиме с другими устройствами. Обратите внимание, что степень поддержки NFC будет различной для разных устройств

Добавление эффектов эквалайзера (настройка усиления в области низких или высоких частот), усиления басов (увеличение громкости звучания басового звука), виртуализации наушников (имитируется пространственное звучание) и реверберации (эхо-эффектов) в одну звуковую дорожку или несколько дорожек.

Встроена поддержка кодеков Advanced Audio Coding (AAC, Расширенный звуковой кодек — «наследник» MP3) и Adaptive Multi-Rate Wideband (AMRWB, Адаптивный широкополосный мультисоставной кодек), обеспечивающих получение звука высокого качества.

В открытый контейнерный формат WebM встроена поддержка открытого алгоритма видео сжатия VP8.

С помощью библиотеки API Camera, обладающей расширенным набором функций, можно получить доступ к фронтальной и тыловой камере устройства, определить используемые функции и активизировать соответствующую камеру.

1.6 Android 3.0 (Honeycomb)

Согласно некоторым оценкам, к 2015 году доля продаваемых планшетов составит более 20 % от всего рынка продаваемых персональных вычислительных устройств¹. Среди существующих планшетов быстрее всего растет доля продаваемых планшетов Android. Согласно сведениям, опубликованным компанией Consumer Electronic Show, в 2011 году было анонсировано 85 новых планшетов Android². В версии Android 3.0 (Honeycomb, «пчелиные соты») появились усовершенствования пользовательского интерфейса, предназначенные для устройств с большим экраном (то есть планшетов). Среди этих улучшений — усовершенствованная клавиатура, обеспечивающая более комфортный ввод данных, 3D-имитация пользовательского интерфейса, панели System (Система) и Action (Действие), облегчающие навигацию, и ряд других улучшений. В распоряжении разработчика также предоставляются новые инструменты, предназначенные для оптимизации приложений, рассчитанных на работу с большими экранами [1].

Новая клавиатура обеспечивает улучшенную точность и эффективность ввода текста. Новый пользовательский интерфейс облегчает выделение, копирование и вставку текста.

Панель System обеспечивает быстрый доступ к навигационным кнопкам, уведомлениям и статусу системы. Эта панель, находящаяся в верхней части экрана, включает специфичные для приложения элементы управления (например, элементы управления навигацией). В списке Recent Apps (Последние приложения), находящемся на панели System, отображаются задания, которые выполняются одновременно. Этот список позволяет переключаться между выполняющимися приложениями

Функция подключения устройства Android к клавиатуре с помощью порта USB или подключения Bluetooth. Эти протоколы, разработанные компанией Microsoft, обеспечивают передачу файлов фотографий, видео и музыки с устройства Android на

компьютер. Можно разработать приложения, с помощью которых пользователи могут создавать и управлять медиафайлами, а также пересылать их на различные устройства. Модем Bluetooth осуществляет подключение к сети Wi-Fi или 3G компьютера или других устройств с помощью модема, в качестве которого используются Android-устройства.

Вместо нескольких отдельных окон новое приложение Browser, которое создает вкладки, облегчает просмотр «не мобильных» веб-сайтов (с помощью улучшенного масштабирования, прокрутки и других дружественных пользователю методик). Также поддерживается режим «инкогнито» при анонимном просмотре сайтов, обеспечивается поддержка мультисенсорного экрана для JavaScript и подключаемых модулей и другие подобные методики. Можно также автоматически входить на веб-сайты Google и синхронизировать закладки с Google Chrome.

Приложение Camera переработано с учетом использования устройств с большим экраном. С помощью этого приложения пользователи могут легко получать доступ к функциям камеры, например к фронтальной камере, к вспышке, автофокусу и другим элементам управления камерой устройства Android. Функции изменения скорости видеозаписи позволяют снимать видео с меньшей частотой кадров, чем стандартная. Воспроизведение такого видео с обычной скоростью создает иллюзию ускоренного развития событий.

Пользовательский интерфейс, состоящий из двух панелей, облегчает чтение, редактирование и организацию контактов. Благодаря быстрой прокрутке облегчается быстрый поиск контактов.

Воспользуйтесь панелью Action (Действие) для распределения электронной почты по папкам и синхронизации вложений. Воспользуйтесь виджетом e-mail, находящимся на главном экране, для упрощенного просмотра сообщений.

Просмотр альбомов в полноэкранном режиме, на котором отображаются миниатюры, облегчающие просмотр фотографий в альбоме.

Версия Android 3.x совместима с приложениями, разработанными с помощью предыдущих версий Android.

Новые и разработанные ранее приложения в среде Android 3 приобретают «объемный» вид благодаря добавлению соответствующего атрибута в файл манифеста приложения.

Добавление новых компоновок и макетов, предназначенных для устройств с большим экраном, в существующие приложения, разработанные для устройств с маленьким экраном.

Распределение действий приложений по модульным фрагментам, которые могут использоваться в различных комбинациях. В Google имеются расширения этой библиотеки API, поэтому она может использоваться в Android 1.6 и более старших версиях.

В число этих виджетов входят поле поиска, календарь, 3D стек, указатель даты/времени и ряд других. Виджеты, находящиеся на экране Home, могут контролироваться с помощью касания. Эти жесты применяются для прокрутки и смахивания содержимого экрана.

Каждое приложение имеет свою собственную постоянную панель Action, предлагающую пользователям возможности навигации и ряд других возможностей.

К числу расширений для игр относятся следующие:

- Улучшенная производительность, обеспечиваемая за счет параллельного «сбора мусора», более быстрого распределения событий и улучшенным видеодрайверам.
- Обработка событий, связанных с вводом собственных данных и сенсорами.
- Новые сенсоры (гироскоп, барометр, сенсор гравитации и ряд других) обеспечивают улучшенную обработку 3D- перемещений.

- Библиотека API Khronos OpenGL ES API предназначена для воспроизведения собственного аудио.
- Библиотека Khronos EGL предназначена для управления собственной графикой.
- Собственный доступ к Activity Lifecycle (Жизненный цикл действия) и к API, предназначенным для управления окнами.
- Библиотеки API Native Asset Manager (Диспетчер собственных ресурсов) и Storage Manager (Диспетчер хранилища).
- Дополнительные возможности уведомлений.

С помощью класса-построителя (builder) выполняется добавление в уведомления приложений больших и маленьких значков, заголовков и флагов приоритета. С помощью буфера обмена пользователи могут копировать и вставлять данные в различные приложения. Фреймворк DragEvent применяется для добавления функций перетаскивания в приложение.

Множественное выделение. С помощью этой функции пользователи могут выделять несколько элементов в списке или таблице

Протокол Media/Picture Transfer Protocol (MTP/ PTP, Передача медиа-файлов/изображений). Благодаря этой функции пользователи могут легко передавать любые типы медиафайлов с устройств Android на главные компьютеры.

Поддержка архитектуры процессором с несколькими ядрами. Технология http Live Streaming (HLS) Обеспечивает выполнение Android 3.x на архитектурах с одним ядром или несколькими ядрами (для улучшения производительности).

Графический механизм Renderscript 3D. Приложения могут поддерживать URL-ссылки для мультимедийных списков воспроизведения в медиафреймворке, используемые для открытия сеанса HTTP Live Streaming. В результате обеспечивается высококачественная поддержка адаптивного видео.

Аппаратно ускоренная двумерная графика. Создание высокопроизводительной 3D-графики для приложений, виджетов и прочих применений, а также перенос вычислений, выполняемых при отображении графики, на графический процессор (GPU, Graphics Processing Unit).

Упрощенная анимация объектов или элементов интерфейса пользователя. Новый анимационный фреймворк. Новый модуль рендеринга OpenGL улучшает производительность при выполнении простых операций с графикой.

Bluetooth A2DP и HSP. Приложения API для профилей Bluetooth Advanced Audio Distribution Profile (A2DP) и Headset Profile (HSP) обеспечивают проверку наличия подключенных устройств Bluetooth, уровень заряда аккумуляторной батареи и некоторые другие параметры.

Фреймворк Digital Rights Management (DRM, Управление цифровыми правами). Новые политики для приложений, выполняющих управление устройствами. Библиотека API, которая позволяет управлять защищенным контентом в приложениях.

Корпоративные приложения по администрированию устройств, поддерживающие различные политики, такие как истечение срока действия паролей и ряд других.

1.7 Android Ice Cream Sandwich

Версия Android Ice Cream Sandwich («сэндвич с мороженым») появилась в октябре 2012 года. Она сочетает в себе возможности Android 2.3 (Gingerbread) и Android 3.0 (Honeycomb) и предназначена для использования на всех устройствах Android. Эта версия Android позволяет включить функции, поддерживаемые в версии Honeycomb («голографический» интерфейс пользователя, новый загрузчик и ряд других функций, ранее доступных только для планшетов), в приложения, предназначенные для смартфонов. В результате обеспечивается простое масштабирование приложений, позволя-

ющее их использовать на различных устройствах. В версии Ice Cream Sandwich также появился ряд новых функций.

0-click NFC Peer-to-Peer Sharing (Общий одноранговый доступ к NFC, не требующий вмешательства пользователя). Пользователи совместимых Android-устройств могут получать общий доступ к контенту (например, к контактам и видео) путем размещения устройств поблизости друг от друга.

Head tracking (Отслеживание лиц). С помощью камеры совместимые устройства могут определять положение глаз, носа и рта пользователя. Камера может также отслеживать направление взгляда пользователя, позволяя создавать приложения, которые изменяют перспективу на основе взгляда пользователя (перерисовывание ландшафтов в трехмерных играх).

Virtual camera operator (Виртуальный оператор камеры). В процессе видеосъемки камера автоматически фокусируется на говорящем человеке. Например, если два человека относятся к одной стороне видеочата, камера идентифицирует говорящего человека из этой пары и сфокусируется на нем.

Android@Home framework (Фреймворк Android@ Home). Обеспечивает создание приложений Android, контролирующих находящиеся у пользователя дома бытовые приборы, например выключатели освещения (которые обычно контролируют специальные лампы), настройку термостатов, контроль системы полива и выполнение других подобных задач

1.8 Загрузка приложений из Android Market

На время выхода этой книги в Google Android Market были доступны сотни тысяч приложений, и это число постоянно росло. Приложения можно загрузить из Android Market непосредственно на ваше устройство Android. Также Android Market рассылает уведомления о наличии обновлений для загруженных приложений.

Comics (Комиксы): Marvel Superheroes, Dilbert Calendar, Jerry Seinfeld, Jokes.

Communication (Связь): Google Voice, Skype mobile™, Wi-Fi Locator, Easy.

Entertainment (Развлечения): Face Melter, Fingerprint Scanner, Fandango® Movies.

Finance (Финансы): Mint.com Personal Finance, PayPal, Debt Payoff Planner.

Games: Arcade & Action (Игры: аркады и экшны): NESoid, Droid Breakout, Raging Thunder 2 Lite, Whac 'em!

Games: Brain & Puzzle (Игры: головоломки): Games: Cards & Casino (Игры: азартные).

Games: Casual (Игры: логические): City Mayor, LOL Libs, Paper Toss, SuperYatzy.

Free Edition Health (Здоровье): Fast Food Calorie Counter, CardioTrainer, StopSmoking.

Lifestyle (Стиль жизни): Zillow Real Estate, Epicurious Recipe App, Family Locator.

Multimedia (Мультимедиа): Pandora Radio, Shazam, Last.fm, iSyncr, Camera Illusion.

News & Weather (Новости и погода): The Weather Channel, CNN, NYTimes, FeedR News Reader.

Productivity (Бизнес): Adobe® Reader®, Documents To Go 2.0 Main App.

Reference (Справочники): Google Sky Map, Dictionary.com, Wikidroid for Wikipedia.

Shopping (Покупки): Gluten Free, Amazon.com, Barcode Scanner, Pkt Auctions, eBay.

Social (Социальные сети): Facebook®, Twitter for Android, MySpace, Bump, AIM.

Sports (Спорт): NFL Mobile, Nascar Mobile, Google Scoreboard.

Themes (Темы): Pixel Zombies Live Wallpaper, Aquarium Live Wallpaper.

Tools (Инструменты): Compass, Droidlight LED Flashlight, AppAlarm Pro.

Travel (Путешествия): Google Earth, Yelp®, Urbanspoon, WHERE, XE Currency.

Demo (Демо-версии): Screen Crack, Bubbles, CouponMap, SnowGlobe.

Software libraries (Библиотеки программ); Translate Tool, Security Guarder, Car Locator Bluetooth Plugin.

Посетите веб-сайт market.android.com и найдите требуемые приложения либо посетите сайты с рекомендациями по выбору и использованию приложений. Здесь вы найдете как бесплатные, так и платные приложения. Цены на приложения, распространяемые через Android Market, устанавливают разработчики, получая 70 % от прибыли. Многие разработчики приложений используют маркетинговую стратегию, суть которой заключается в том, что базовая версия приложения распространяется бесплатно. Если же пользователю понравилась эта версия, он может приобрести полную версию приложения, обладающую расширенным набором функций.

1.9 Пакеты

В Android используется целая коллекция пакетов, являющихся группами связанных предварительно определенных классов. Некоторые из пакетов специфичны для Android, другие относятся к Java и Google. С помощью пакетов обеспечивается удобный доступ к функциям операционной системы Android, а также включение этих функций в приложения. Большая часть пакетов написана на языке программирования Java и доступна из Java-программ. С помощью пакетов Android облегчается создание Android-приложений с уникальным внешним видом и особенностями интерфейса пользователя. Полный перечень пакетов Android приводится на веб-сайте developer.android.com/reference/packages.html.

Android.app: Включает классы высокого уровня в модели приложения Android.

Android.os: Службы операционной системы.

Android.text: Отображение и отслеживание текста на экране устройства.

Android.widget: Классы интерфейса пользователя, предназначенные для виджетов.

Android.net: Классы доступа к Сети.

Android.view: Классы интерфейса пользователя, предназначенные для взаимодействия с пользователем и компоновок.

java.io: Потоки, сериализация и доступ к файловой системе для устройств ввода и вывода.

java.util: Классы утилиты.

Android.content.res: Классы, предназначенные для обеспечения доступа к ресурсам приложения (например, к медиа файлам, цветам, рисункам и прочим ресурсам), а также к информации о конфигурации устройства, влияющей на поведение приложения.

Android.graphics.drawable: Классы, предназначенные только для отображаемых на экране элементов (например, градиентов).

android.media: Классы, предназначенные для обработки медиа интерфейсов аудио и видео.

android.util: Методы утилит и утилиты XML.

android.content: Доступ и публикация данных на устройстве.

android.hardware: Поддержка аппаратного обеспечения устройств.

android.provider: Доступ к провайдерам контента Android.

android.database: Обработка данных, возвращаемых провайдером контента.

android.database.sqlite: Управления базами данных SQLite для частных баз данных.

android.graphics: Графические инструменты, применяемые для рисования на экране.

android.location: Службы, основанные на локации.

com.google.android.maps: Используется в приложении Route Tracker.

android.appwidget: Используется в приложении Weather Viewer.

java.net: Сетевые классы (например, обработка адресов Интернета и запросы HTTP).

javax.xml.parsers: Обработка XML-документов.

org.xml.sax: Simple API для XML (SAX API), используемые для чтения данных из XML-документов.

1.10 Android Software Development Kit (SDK)

В состав Android SDK включены инструменты, предназначенные для создания Android-приложений. Этот набор инструментов доступен на веб-сайте Android Developers (на бесплатной основе). В число этих инструментов входят Java SE, интегрированная среда разработки Eclipse, Android SDK 3.x и подключаемый модуль ADT для Eclipse [1].

Интегрированная среда разработки Eclipse рекомендуется к использованию при разработке Android-приложений. Несмотря на эти рекомендации, для создания Android-приложений можно использовать текстовый редактор и инструменты командной строки. В среде Eclipse поддерживается множество языков программирования, в том числе Java, C++, C, Python, Perl, Ruby на Rails и ряд других. Подавляющее большинство приложений Android создаются на языке Java. В состав Eclipse входят следующие компоненты и функции:

- редактор кода с поддержкой выделения синтаксиса цветом и нумерации строк;
- автоматические отступы и автозавершение (то есть подсказки при вводе кода);
- отладчик;
- система контроля версий;
- поддержка рефакторинга.

Плагин ADT (Android Development Tools, Инструменты разработки Android-приложений) для Eclipse — это расширение интегрированной среды разработки Eclipse. С помощью этого подключаемого модуля можно создавать, выполнять и отлаживать приложения Android, экспортировать их для дальнейшего распространения (например, выгружать на Android Market) и выполнять ряд других операций. Плагин ADT также включает визуальный инструмент, применяемый для создания графического интерфейса пользователя. Чтобы спроектировать интерфейс с помощью этого инструмента, достаточно воспользоваться готовыми компонентами, которые компоуются путем обычного перетаскивания.

Эмулятор Android, включенный в состав Android SDK, позволяет создать имитационную среду для запуска приложений Android под управлением Windows, Mac OS X либо Linux. Эмулятор отображает весьма реалистичное окно интерфейса пользователя Android. Перед запуском приложения на выполнение следует создать AVD (Android Virtual Device, Виртуальное устройство Android). Это устройство определяет функциональные характеристики реального устройства Android, на котором нужно тестировать приложения. А именно, аппаратное обеспечение, системный образ, размер экрана, хранилище данных и ряд других характеристик. Если нужно тестировать приложения на нескольких устройствах Android devices, придется создать отдельные AVD, имитирующие уникальные физические устройства.

Большинство экранных снимков в книге было сделано именно с помощью эмулятора, а не реального устройства Android. С помощью эмулятора можно симитировать большинство жестов Android и элементов управления, используя клавиатуру и мышь компьютера. Конечно, возможности воспроизведения жестов с помощью эмулятора несколько ограничены, поскольку компьютер не в состоянии симитировать все функции аппаратуры Android. Например, чтобы протестировать GPS-приложения с помощью эмулятора, нужно создать файлы, имитирующие данные GPS. Хотя можно смоделировать изменения ориентации (переключение в портретный/альбомный режим), не существует способа смоделировать показания акселерометра (это устройство оценивает ориентацию и наклон устройства). Чтобы протестировать эти и подобные им функции, загрузите приложение на физическое устройство Android.

Имитация жестов Android на эмуляторе (описана на сайте developer.android.com/guide/developing/tools/emulator.html).

Касание: Щелкните левой кнопкой мыши.

Двойное касание: Дважды щелкните левой кнопкой мыши.

Длинное нажатие: Щелкните левой кнопкой мыши и удерживайте ее.

Перетаскивание: Щелкните левой кнопкой мыши, удерживайте ее и перетаскивайте мышью.

Пощелкивание: Щелкните левой кнопкой мыши и, не отпуская ее, переместите указатель в направлении пощелкивания.

Смахивание: Щелкните левой кнопкой мыши и, удерживая ее, переместите указатель мыши в направлении смахивания, а потом быстро отпустите кнопку мыши.

Масштабирование двумя пальцами: Нажмите и удерживайте клавишу Ctrl. Появятся две окружности, имитирующие касание экрана двумя пальцами. Переместите окружности в начальную позицию, щелкните левой кнопкой мыши и, удерживая ее, переместите окружности в конечную позицию.

Имитация органов управления Android с помощью эмулятора (имитация дополнительных органов управления описана на сайте developer.android.com/guide/developing/tools/emulator.html).

Пиктограмма Back (Назад): Esc.

Вызов/набор номера: F3

Камера: Ctrl+5 на цифровой клавиатуре, Ctrl+F3.

Завершить вызов: F4.

Пиктограмма Home (Домой): Home.

Меню (левая программная кнопка): F2 или Page Up.

Питание: F7.

Орган управления устройства Android: Действие в эмуляторе.

Поиск: F5.

* (правая программная кнопка): Shift+F2 или PageDown.

Вращение влево: 7 на цифровой клавиатуре, Ctrl+F11.

Вращение вправо: 9 на цифровой клавиатуре, Ctrl+F12.

Включение/выключение сети мобильной связи: F8.

Кнопка увеличения громкости: + на цифровой клавиатуре, Ctrl+F5.

Кнопка уменьшения громкости: - на цифровой клавиатуре, Ctrl+F6.

1.11 Краткий обзор объектной технологии

При разработке современных и эффективных программ достаточно трудно выполнить такие требования, как быстрота разработки, корректность и экономичность. Современные методики программирования основаны на использовании объектов (или более точно, объектов классов, представляющих собой повторно используемые программные компоненты). В качестве объектов может использоваться дата, время, видео, человек, автомобиль и другие предметы материального мира. Практически каждое существенное может быть адекватным образом представлено программным объектом в терминах атрибутов (например, имя, цвет и размер) и поведений (например, вычисление, перемещение и коммуникация). Разработчики программ пришли к выводу, что использование модульной структуры и объектно-ориентированного проектирования при разработке приложений способствуют более продуктивной работе. Этот подход пришел на смену применявшемуся ранее структурному программированию. К тому же объектно-ориентированные программы зачастую проще понимать, корректировать и модифицировать [1].

Рассмотрим автомобиль в качестве объекта.

Чтобы лучше понять суть объектов и относящегося к ним контента, воспользуемся простой аналогией. Представьте себе, что вы находитесь за рулем автомобиля и

нажимаете педаль газа, чтобы набрать скорость. А теперь ответим на вопрос, что должно произойти до того, как вы получите возможность водить автомобиль? Прежде чем вы получите возможность водить автомобиль, его нужно изготовить. Изготовление любого автомобиля начинается с инженерных чертежей (калек), которые подробно описывают устройство автомобиля. На этих чертежах даже показано устройство педали акселератора. За этой педалью скрываются сложные механизмы, которые фактически ускоряют автомобиль подобно тому, как педаль тормоза скрывает механизмы, тормозящие автомобиль, а руль «скрывает» механизмы, поворачивающие автомобиль. Благодаря этому люди, не имеющие понятия о внутреннем устройстве автомобиля, могут легко им управлять.

Подобно тому, как невозможно готовить пищу на кухне, которая находится лишь на листе бумаги, нельзя водить автомобиль, существующий лишь в чертежах. Прежде чем вы сядете за руль машины, ее нужно воплотить в металл на основе инженерных чертежей. Воплощенный в металле автомобиль имеет реальную педаль газа, с помощью которой он может ускоряться, но, к счастью, он не может это делать самостоятельно, а только с помощью водителя.

Методы и классы. Воспользуемся примером с автомобилем для иллюстрации некоторых ключевых концепций объектно-ориентированного программирования. Для выполнения задачи в программе требуется метод, «скрывающийся» в инструкциях программы, которые фактически выполняют задание. Метод скрывает эти инструкции от пользователя подобно тому, как педаль газа автомобиля скрывает от водителя механизмы, вызывающие ускорение автомобиля. Программная единица, именуемая классом, включает методы, выполняющие задачи класса. Например, класс, представляющий банковский счет, может включать три метода, один из которых выполняет пополнение счета, второй — вывод средств со счета, а третий — запрос текущего баланса. Класс подобен концепции автомобиля, представленного инженерными чертежами, которые также включают чертеж педали газа, рулевого колеса и других механизмов.

Создание экземпляра класса. Подобно тому, как изготовление автомобиля на основе чертежей — обязательное условие его вождения, создание объекта класса — обязательное условие выполнения задач, реализованное на основе методов этого класса. Этот процесс называется созданием экземпляра. Полученный при этом объект называется экземпляром класса.

Повторное использование. Подобно тому, как на основе одних и тех же чертежей можно создать много автомобилей, на основе одного класса можно создать много объектов. Благодаря использованию существующих классов для создания новых классов экономится время и силы разработчика. Повторное использования также облегчает создание более надежных и эффективных систем, поскольку ранее созданные классы и компоненты не нуждаются в тестировании, отладке и оптимизации производительности. Подобно тому, как концепция использования взаимозаменяемых частей легла в основу индустриальной революции, повторно используемые классы — двигатель прогресса в области создания программ, который был вызван внедрением объектной технологии.

Сообщения и вызовы методов. В процессе вождения автомобиля в результате нажатия педали газа отсылается сообщение автомобилю, содержащее запрос на выполнение определенной задачи (ускорение автомобиля). Подобным же образом отсылаются сообщения объекту. Каждое сообщение представляет собой вызов метода, который «сообщает» методу объекта о необходимости выполнения определенной задачи. Например, программа может вызвать метод депозит определенного объекта банковского счета, чтобы пополнить банковский счет.

Атрибуты и переменные экземпляра класса. Любой автомобиль помимо возможности выполнять определенные задачи также обладает атрибутами, такими как цвет, количество дверей, запас топлива в баке, показания спидометра и одометра. Подобно возможностям по выполнению определенных действий, атрибуты автомобиля

представлены на инженерных диаграммах, представляющих собой часть проекта (в качестве атрибутов автомобиля могут выступать одометр и указатель уровня бензина). При вождении автомобиля его атрибуты перемещаются вместе с ним. Каждому автомобилю присущ собственный набор атрибутов. Например, каждый автомобиль «знает» о том, сколько бензина осталось в его баке, но ничего не «знает» о запасах горючего в баках других автомобилей.

Объект, как и автомобиль, имеет собственный набор атрибутов, которые он «переносит» с собой при использовании этого объекта в программах. Эти атрибуты определены в качестве части объекта класса. Например, объект `bankaccount` имеет атрибут баланса, представляющий количество средств на банковском счету. Каждый объект `bankaccount` «знает» о количестве средств на собственном счету, но ничего не «знает» о размерах других банковских счетов. Атрибуты определяются с помощью других переменных экземпляра класса.

Инкапсуляция. Классы инкапсулируют атрибуты и методы в объекты (атрибуты и методы объекта между собой тесно связаны). Объекты могут обмениваться информацией между собой, но обычно они не «знают» о деталях реализации других объектов, которые скрыты внутри самих объектов. Подобное сокрытие информации жизненно важно в практике хорошего программного инжиниринга.

Наследование. С помощью наследования можно быстро и просто создать новый класс объектов. При этом новый класс наследует характеристики существующего класса, которые при этом могут частично изменяться. Также в новый класс добавляются свойства, которые являются уникальными и присущими только этому классу. Если вспомнить аналогию с автомобилем, «трансформер» является объектом более обобщенного класса «автомобиль», у которого может подниматься или опускаться крыша.

Объектно-ориентированный анализ и проектирование. А теперь ответьте на вопрос, каким образом вы собираетесь программировать? Скорее всего, таким же образом, как и большинство других программистов, — включите компьютер и начнете вводить исходный код программы. Подобный подход годится при создании маленьких программ, но что делать в том случае, когда приходится создавать крупный программный комплекс, который, например, управляет тысячами автоответчиков в колл-центре крупного банка? Либо если вам приходится возглавлять команду из 1000 программистов, занятых разработкой системы управления воздушным движением следующего поколения? Начать выполнение столь крупных и сложных проектов с того, что сесть за компьютер и вводить код, означает потерпеть неудачу.

Чтобы выработать наилучшее решение, следует выполнить процедуру детального анализа процесса определения требований к программному проекту (то есть определить задачи, выполняемые системой) и разработать проект, который будет соответствовать этим требованиям (то есть определить, каким образом система будет выполнять поставленные перед ней задачи). В идеале перед началом создания кода следует выполнить эту процедуру и тщательно проанализировать проект (либо поручить выполнение этой задачи коллегам-профессионалам). Если в ходе выполнения этого процесса происходит анализ и проектирование системы с применением объектно-ориентированного подхода, значит мы имеем дело с процессом OOAD (object-oriented analysis and design, объектно-ориентированный анализ и проектирование). Языки программирования, подобные Java, называются объектно-ориентированными. Принципы объектно-ориентированного проектирования могут быть внедрены на практике с помощью подобных языков программирования.

1.12 Тестирование приложения Doodlz на виртуальном устройстве AVD

В этом разделе вы запустите на выполнение и будете «общаться» со своим первым приложением Android. С помощью приложения Doodlz пользователь может «рисовать» на экране, выбирая различные цвета и кисти разных размеров. Это приложение будет

создано в главе 9. Следующая пошаговая инструкция предназначена для импортирования проекта приложения в среду Eclipse и выполнения тестирования на устройстве AVD (Android Virtual Device, Виртуальное устройство Android). Создание и настройка устройства AVD описаны в разделе «Подготовительные действия», находящемся во вводной части книги. Позднее будет рассмотрено, каким образом можно выполнять это приложение на реальном устройстве Android [1].

Экранные снимки, используемые в качестве иллюстраций, были созданы на компьютере, на котором установлены Windows 7, Java SE 6, Eclipse 3.6.1, Android 2.2/2.3/3.0 и подключаемый модуль ADT для Eclipse.

1. *Проверьте конфигурацию вашей системы.* Убедитесь в том, что ваш компьютер настроен в полном соответствии с требованиями, изложенными в разделе «Подготовительные действия», находящегося во вводной части книги.

2. *Откройте среду Eclipse.* Чтобы запустить Eclipse, откройте папку, в которой находится Eclipse, в вашей системе и дважды щелкните на значке Eclipse. Если Eclipse запускается первый раз, появится вкладка Welcome (Добро пожаловать), показанная на рис. 1.1. Щелкните на кнопке Workbench (Рабочая среда), находящейся на этой вкладке, чтобы закрыть вкладку и выбрать представление, используемое для разработки программ, которое носит формальное название «Java perspective in Eclipse».



Рис. 1.1. Вкладка Welcome в окне Eclipse

3. *Откройте диалоговое окно Import (Импорт).* Выполните команды File→Import... (Файл→Импорт), чтобы открыть диалоговое окно Import (рис. 1.2).

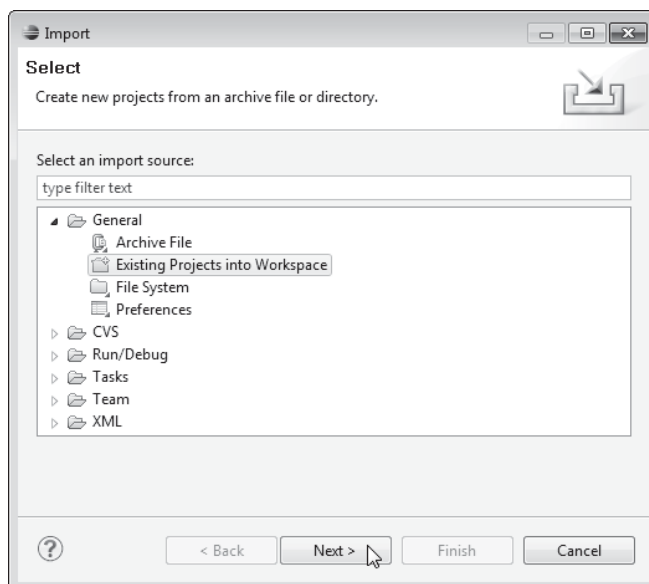


Рис. 1.2. Диалоговое окно Import

4. *Импорт проекта приложения Doodlz.* В диалоговом окне Import раскройте узел General (Общий), выберите параметр Existing Projects into Workspace (Существующие проекты в рабочую среду), потом щелкните на кнопке Next> (Далее>) для выполнения шага Import Projects (Импорт проектов), как показано на рис. 1.3. Установите переключатель Select root directory (Выберите корневой каталог), потом щелкните на кнопке Browse... (Просмотр...). В диалоговом окне Browse For Folder (Просмотр папок), показанном на рис. 1.4, найдите папку Doodlz, которая находится в папке примеров книги, выберите ее и щелкните на кнопке ОК. Нажмите кнопку Finish (Готово), чтобы импортировать проект в Eclipse. Проект появится в окне Package Explorer (Диспетчер пакетов), которое показано на рис. 1.5. Это окно находится в левой части окна Eclipse.

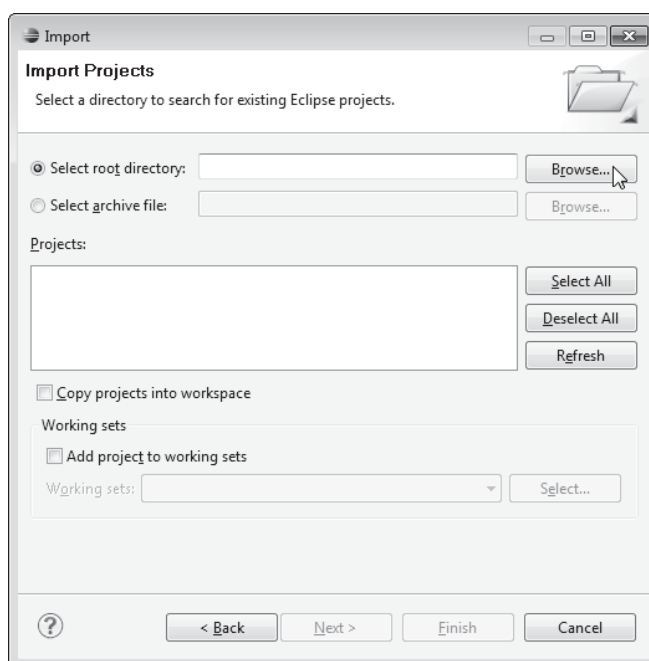


Рис. 1.3. Шаг Import Projects, выполняемый в окне Import

5. *Запуск приложения Doodlz.* В среде Eclipse выберите проект Doodlz в окне Package Explorer (см. рис. 1.5), затем выполните команду Run As Android Application (Выполнить как Приложение Android), выбрав ее в раскрывающемся меню кнопки Run As панели инструментов интегрированной среды разработки (рис. 1.6). В результате начнет выполняться приложение Doodlz на виртуальном устройстве NexusS Android Virtual Device (AVD) (рис. 1.7), создание которого описано в разделе «Подготовительные действия» вводной главы. Если нужно протестировать приложение на различных AVD-устройствах, выполните команды Window Android SDK and AVD Manager (Окно Диспетчер SDK и AVD), потом выберите нужное устройство AVD и щелкните на кнопке Start... (Запуск...). Если в момент запуска приложения уже выполняются несколько устройств AVD, появится диалоговое окно Android Device Chooser (Выбор устройства Android), где можно выбрать устройство AVD, на котором будет выполняться приложение. Подробное рассмотрение окна Android Device Chooser будет выполнено позднее.

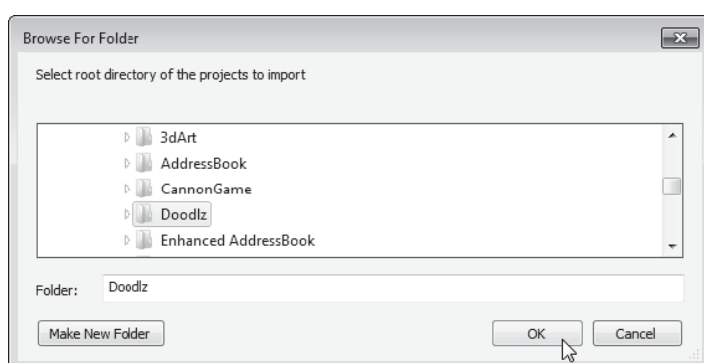


Рис. 1.4. Диалоговое окно Browser For Folder

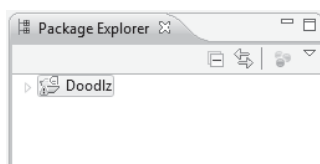


Рис. 1.5. Окно Package Explorer в среде Eclipse

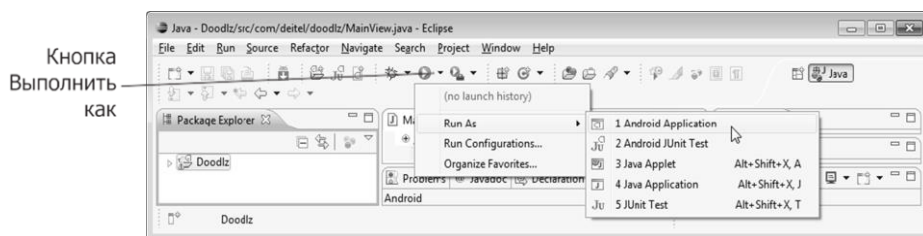


Рис. 1.6. Запуск приложения Doodlz

Структура AVD. В левой части окна AVD отображается выполняющееся приложение. В правой части окна (рис. 1.8) отображаются различные кнопки, которые имитируют физические и программные кнопки реального устройства Android, и клавиатура, имитирующая физическую и программную клавиатуры устройства. Физические кнопки — это реальные кнопки устройства. Программные кнопки отображаются на сенсорном экране устройства. С помощью кнопок

устройства AVD можно взаимодействовать с приложениями и операционной системой Android, выполняемыми на AVD. После установки приложения на устройстве Android можно приступать к созданию нового рисунка путем перетаскивания пальцем в области холста. При использовании устройства AVD «касание» поверхности экрана имитируется мышью.

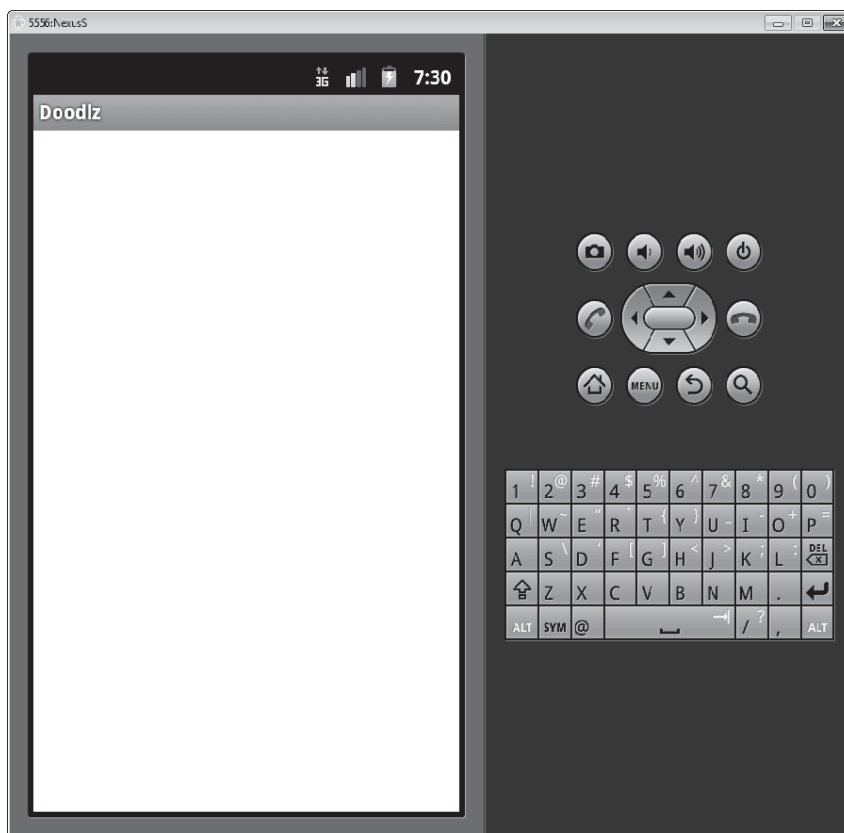


Рис. 1.7. Выполнение приложения Doodlz на виртуальном устройстве Android

6. *Отображение параметров приложения.* Чтобы отобразить параметры приложения, коснитесь кнопки Меню (Menu). Обратите внимание, что на некоторых реальных устройствах эта кнопка обозначена пиктограммой в виде горизонтальных полосок. После этого отображается меню параметров приложения (рис. 1.9). Это меню включает следующие параметры: Color (Цвет), Line Width (Ширина линии), Erase (Удалить), Clear (Очистить) и Save Image (Сохранить изображение). Выберите параметр Color, и на экране появится графический интерфейс пользователя, с помощью которого можно изменить цвет линии. Если же выбрать параметр Line Width, у вас появится возможность изменить толщину линии. Выберите параметр Erase для окрашивания рисуемых линий в белый цвет (в результате цвет полностью исчезает). После выбора параметра Clear рисунок исчезает. Если же выбрать параметр Save Image, изображение будет сохранено в галерее изображений устройства. Все эти параметры будут подробнее рассмотрены далее.

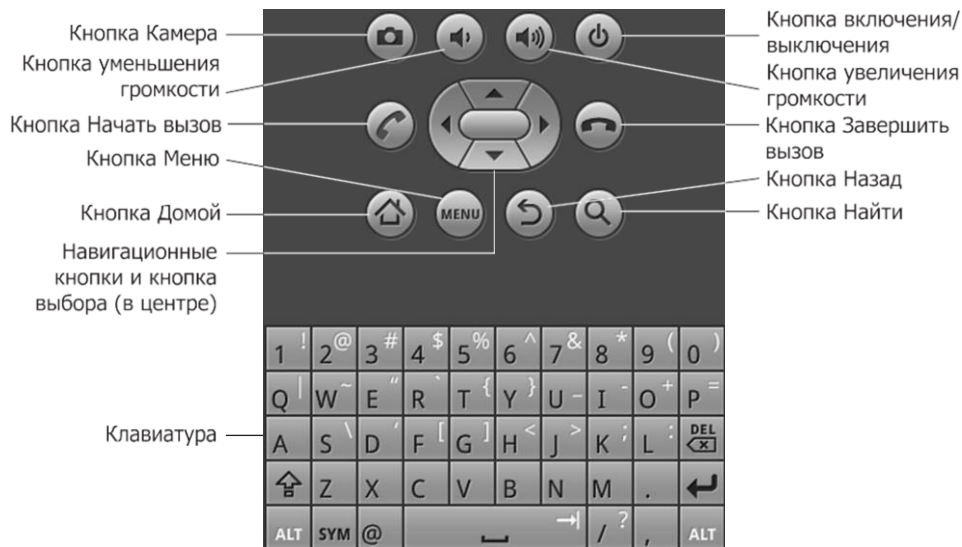


Рис. 1.8. Виртуальное устройство AVD, на котором выполняется приложение Doodlz

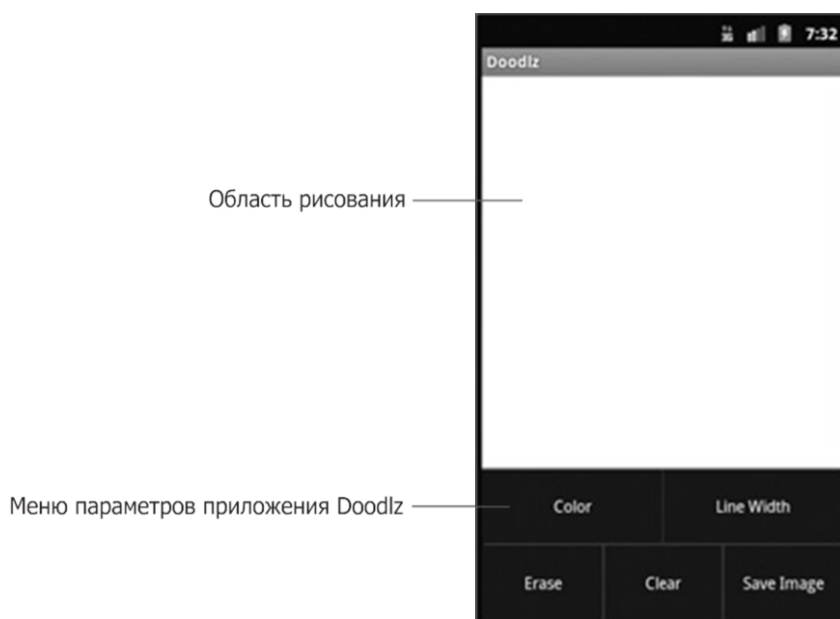


Рис. 1.9. Меню параметров приложения Doodlz

7. *Изменение цвета кисти на красный.* Чтобы изменить цвет кисти, сначала коснитесь параметра Color для отображения панели изменения цвета (рис. 1.10, а). Цвета определяются с помощью цветовой схемы RGBA, в которой с помощью целочисленных цветовых значений (от 0 до 255) определены следующие цветовые компоненты: альфа, красный, зеленый и синий. Панель настройки цвета состоит из полос SeekBar Red (Красный), Green (Зеленый), Blue (Синий) и Alpha (Альфа). С помощью этих элементов направления можно выбирать интенсивность красного, зеленого и синего цветов, а также настраивать прозрачность цвета. Чтобы изменить интенсивность цвета, перетащите полосу SeekBar. В результате в окне приложения отобразится новый цвет. Чтобы выбрать красный цвет, перетащите полосу SeekBar Red в крайнее правое положение (рис. 1.10, а). Нажмите кнопку Done (Готово), чтобы вернуться в область рисования.

Перетащите «палец» (роль которого играет мышь) по экрану и нарисуйте лепестки цветка (рис. 1.10, б).

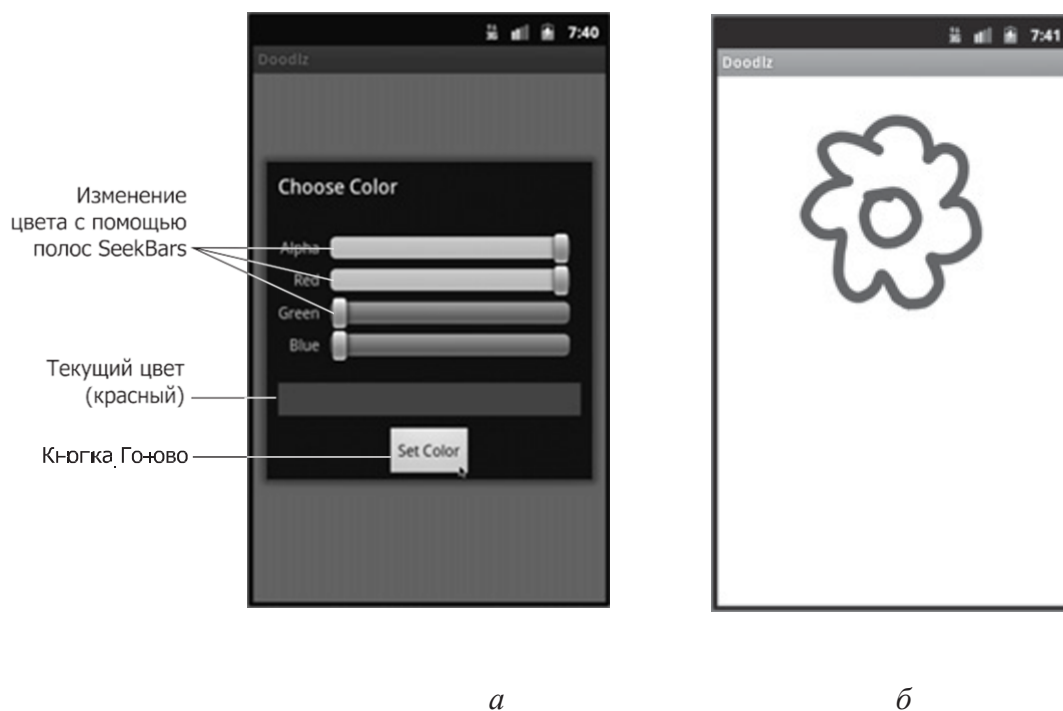


Рис. 1.10. Выбираем красный цвет кисти и рисуем лепестки цветка: а — выбор красного цвета кисти; б — рисование красных лепестков цветка

8. *Изменение цвета кисти на темно-зеленый.* Снова измените цвет кисти, коснувшись кнопки Меню с последующим выбором параметра Color. Выберите темно-зеленый цвет путем перетаскивания полосы Green SeekBar в крайнее правое положение, а полос Red и Blue SeekBars — в крайнее левое положение (рис. 1.11, а).
9. *Изменение толщины линии.* Чтобы изменить толщину линии, коснитесь кнопки Меню, а потом выберите параметр Line Width. Перетащите полосу SeekBar, предназначенную для настройки толщины линии, вправо (рис. 1.11, б). Нажмите кнопку Done, чтобы вернуться в область рисования. Нарисуйте стебель и листья цветка. Повторите шаги 9 и 10, чтобы выбрать более светлый зеленый цвет и более тонкую линию, затем нарисуйте траву (рис. 1.12).



Отображение
линии
в текущем цвете
(зеленый)
и ширины
линии

Регулирование
толщины
линии с помощью
SeekBar



а

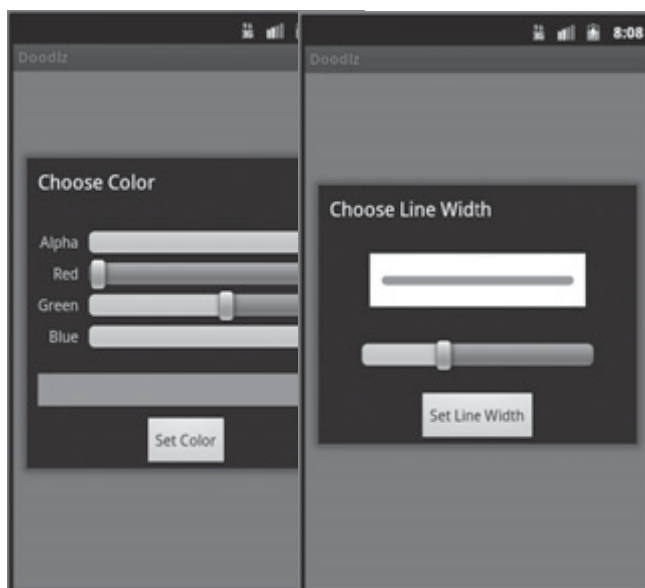
б

Рис. 1.11. Изменение цвета и толщины линии: а — выбор зеленого цвета для рисования; б — выбор более толстой линии



Рис. 1.12. Рисуем стебель и листья после выбора нового цвета и ширины линии

10. *Завершение рисунка.* Воспользуйтесь шагами 9–10, чтобы выбрать синий цвет линии (рис. 1.13, а) и сузить ее (рис. 1.13, б). Вернитесь к области рисования и нарисуйте капли дождя (рис. 1.14).



а

б

Рис. 1.13. Изменение цвета и толщины линии: а — выбор синего цвета для рисования; б — выбор более тонкой линии

11. *Сохраните изображение.* При желании можно сохранить изображение в приложении Gallery (Галерея). Для этого нажмите кнопку Menu (Меню), потом нажмите кнопку Save Image (Сохранить изображение). Данное изображение и другие изображения, хранящиеся в памяти устройства, можно открыть с помощью приложения Gallery.
12. *Вернитесь на главный экран.* Вернитесь на главный экран AVD, щелкнув на кнопке

Home (Домой), находящейся на экране AVD.

Выполнение приложения Doodlz на устройстве Android.

Если у вас имеется устройство Android, протестируйте приложение на этом устройстве.

1. Во-первых, нужно перейти в режим отладки устройства. Запустите приложение Settings (Настройки), установленное на устройстве, затем выберите команды Applications > Development (Приложения > Разработка) и убедитесь в том, что флажок USB debugging (Отладка USB) установлен.



Рис. 1.14. Рисуем капли дождя после выбора нового цвета и толщины линии

2. Подключите устройство к компьютеру с помощью кабеля USB (обычно подобный кабель входит в комплект поставки устройства или же его можно приобрести отдельно).
3. В среде Eclipse перейдите в окно Package Explorer, выберите проект Doodlz, потом выполните команды Run As > Android Application, выбранные в раскрывающемся меню кнопки Run As панели инструментов интегрированной среды разработки (рис. 1.6).

Если отсутствует открытое устройство AVD, но к компьютеру подключено устройство Android, Eclipse автоматически установит приложение на устройство и запустит это приложение. Если открыто одно или большее число устройств AVD и/либо к компьютеру подключены устройства Android, появится диалоговое окно Android Device Chooser (см. рис. 1.15), в котором можно выбрать виртуальное устройство AVD либо реальное устройство, на которое будет установлено и выполнено приложение. В рассматриваемом примере запущено два виртуальных устройства AVD и подключено одно физическое устройство, поэтому приложение может выполняться на одном из трех устройств. Можно установить несколько устройств AVD, с помощью которых эмулируются реальные устройства Android с различными размерами экрана и на которых выполняются различные версии операционной системы Android.

В разделе активных устройств Android (рис. 1.15) доступно лишь одно физическое устройство, подключенное к компьютеру (находится на второй позиции в списке устройств), и три виртуальных устройства AVD. Каждое устройство AVD имеет свое имя AVD, которое может выбрать пользователь (NexusS и MotorolaXoom, как показано на рисунке). Выберите используемое реальное устройство или устройство AVD, щелкните на кнопке ОК для его установки и выполнения на реальном устройстве или виртуальном

устройстве AVD. Если определены другие виртуальные устройства AVD, которые в настоящее время не выполняются, выберите и вызовите на выполнение одно из устройств AVD, находящееся в нижней части диалогового окна Android Device Chooser.

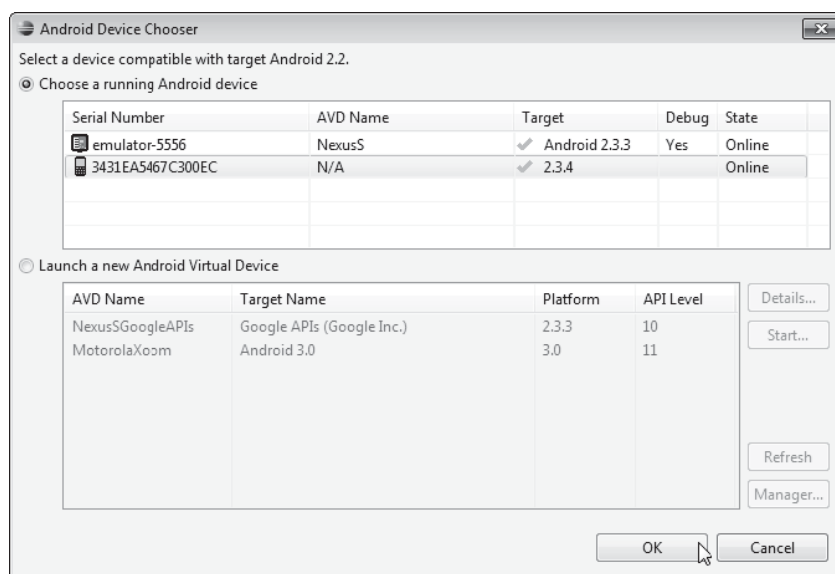


Рис. 1.15. Диалоговое окно Android Device Chooser

Если вы создаете приложения, предназначенные для распространения через Android Market, протестируйте их на возможно большем числе реальных устройств. Не забывайте о том, что некоторые функции могут быть протестированы лишь на реальных устройствах. Если вы не можете найти реальные устройства в достаточном количестве, создайте устройства AVD, имитирующие различные устройства, на которых должно выполняться созданное вами приложение. Просмотрите в Интернете спецификации реальных устройств, а потом сконфигурируйте соответствующим образом каждое виртуальное устройство AVD. Также можно изменить файл config.ini виртуального устройства AVD, как описано в разделе «Setting hardware emulation options» на сайте developer.android.com/guide/developing/tools/avd.html.

Файл config.ini включает параметры, которые не настраиваются с помощью подключаемого модуля ADT в Eclipse. В результате изменения этих параметров можно добиться более точного соответствия выбранного виртуального устройства Android с реальным.

1.13 Резюме

В данной главе вашему вниманию была представлена краткая история Android и функциональные свойства этой платформы. Были рассмотрены функции операционной системы Android 2.2, 2.3 и 3.0. Представлены ссылки на некоторую ключевую документацию в Интернете, группы новостей и форумы, которые позволят вам связаться с сообществом разработчиков. Мы рассмотрели Android Market и ссылки на некоторые популярные обзоры приложений и сайты с рекомендациями. Были изучены жесты Android, а также методика их выполнения на устройствах Android и эмуляторах устройств Android. Были изложены начальные сведения о пакетах Java, Android и Google, которые позволят использовать аппаратные и программные свойства для создания Android-приложений. Многие из этих пакетов будут использованы в дальнейшем. Также были рассмотрены язык программирования Java и Android SDK. Вашему вниманию был предложен краткий обзор концепций объектной технологии, в том числе классы, объекты, атрибуты и поведения. Мы протестировали приложение Doodlz на эмуляторе устройства Android.

2 Инструменты разработки Android

2.1 Введение

В этой главе мы создадим приложение Welcome. Это простое приложение отображает приветственное сообщение и два изображения, и для его создания мы не написали ни одной строчки кода. Воспользуемся возможностями интегрированной среды разработки Eclipse и модулем ADT — двумя наиболее популярными инструментами, применяемыми для разработки и тестирования Android-приложений. Мы рассмотрим возможности среды Eclipse и продемонстрируем, каким образом создается приложение Android (рис. 2.1) с помощью визуального макетного редактора ADT. Этот редактор позволяет создавать графический интерфейс пользователя с помощью технологий перетаскивания. И наконец, мы запустим приложение на виртуальном устройстве Android Virtual Device (AVD) [1].

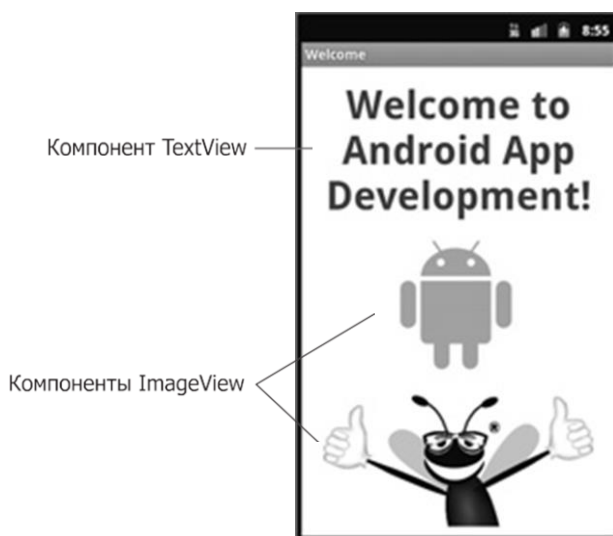


Рис. 2.1. Приложение Welcome

В этой главе вас ожидает знакомство с интегрированной средой разработки Eclipse и подключаемым модулем ADT Plugin. Вы научитесь ориентироваться в Eclipse и создавать новые проекты. Используя визуальный макетный редактор ADT Visual Layout Editor, можно отображать картинки с помощью компонентов ImageViews и текст — с помощью компонента TextView. Будет рассмотрено редактирование свойств компонента GUI (свойство Text компонента TextView и свойства Src компонента ImageView), доступных на вкладке Eclipse Properties (Свойства), а также выполнение приложения на виртуальном устройстве Android Virtual Device (AVD).

Приведенные в главе примеры разработаны с помощью версий Android SDK 2.3.3 и 3.0, а также с помощью среды разработки Eclipse IDE вместе с подключаемым модулем ADT (Android Development Tools) Plugin [1]. Предполагается, что к началу чтения главы вы уже установили Java SE Development Kit (JDK), Android SDK и Eclipse.

В интегрированной среде разработки Eclipse можно управлять, редактировать, компилировать, выполнять и отлаживать приложения. Подключаемый модуль ADT Plugin для Eclipse предоставляет пользователям дополнительные инструменты, применяемые для разработки Android-приложений. Можно также воспользоваться подключаемым модулем ADT Plugin для управления различными версиями платформы Android. Это требуется в случае разработки приложений, предназначенных для различных устройств, на которых установлены различные версии операционной системы Android. После первого запуска среды Eclipse появится вкладка Welcome (Добро пожаловать), которая показана на рис. 2.2. На этой вкладке находится ряд ссылок-пиктограмм. Щелкните на кнопке Workbench (Рабочая среда), чтобы отобразить перспективу разработки Java, в которой начинается

разработка Android-приложений. В Eclipse поддерживается разработка приложений на нескольких языках программирования. Каждый установленный набор инструментов Eclipse представлен отдельной перспективой разработки. В результате изменения перспективы реконфигурируются. Изменение перспектив приведет к реконфигурированию интегрированной среды разработки таким образом, чтобы можно было использовать инструменты для соответствующего языка программирования.

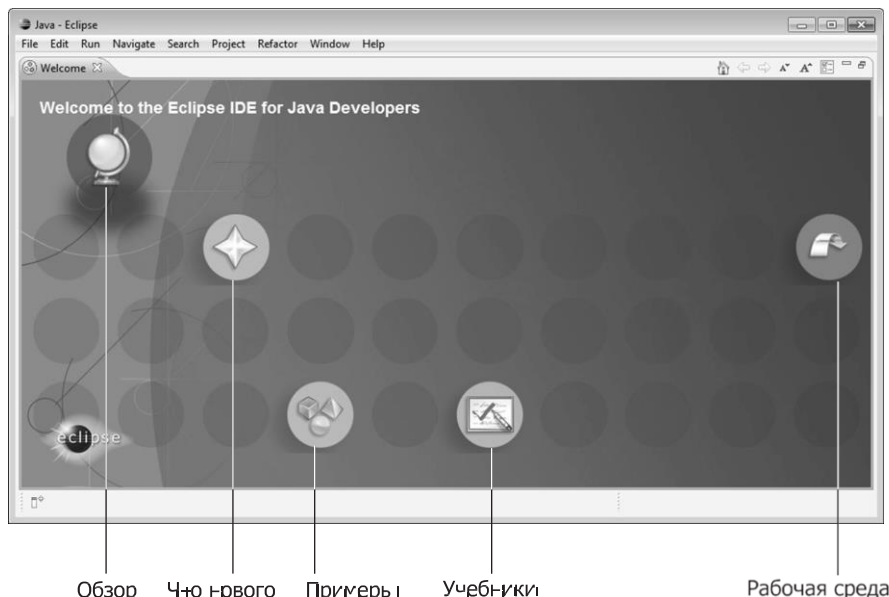


Рис. 2.2 Вкладка Welcome to the Eclipse IDE for Java Developers в окне Eclipse
Ссылки, находящиеся на вкладке Welcome среды Eclipse:

- Overview (Обзор): обзор интегрированной среды разработки и ее функций;
- What's New (Что нового): сведения о новых качествах установленной версии Eclipse, ссылки на сообщества пользователей Eclipse в Интернете и обновления интегрированной среды разработки (DE);
- Samples (Примеры): ссылки на примеры для загруженной конфигурации Eclipse;
- Tutorials (Учебники): учебники, которые помогут вам начать разработку Java-приложений в среде Eclipse и использовать различные возможности Eclipse;
- Workbench (Рабочая среда): переход к перспективе разработки.

2.2 Создание нового проекта

Чтобы начать программировать на Android в Eclipse, выполните команды File☞New☞Project... (Файл☞Новый☞Проект...) для отображения диалогового окна New Project (Новый проект). Раскройте узел Android, выберите параметр Android Project (Проект Android) и щелкните на кнопке Next> (Далее>). На экране появится диалоговое окно New Android Project (Новый проект Android), показанное на рис. 2.3. Это окно также можно отобразить с помощью раскрывающегося списка New (Создать) панели инструментов. После создания первого проекта в меню File☞New и в раскрывающемся меню New появится параметр Android Project (Проект Android) [1].

Проект — это группа связанных файлов, например файлы кода и изображения, которые формируют приложение. В диалоговом окне New Android Project можно создать проект «с нуля» либо воспользоваться существующим исходным кодом (например, примерами кода из книги).

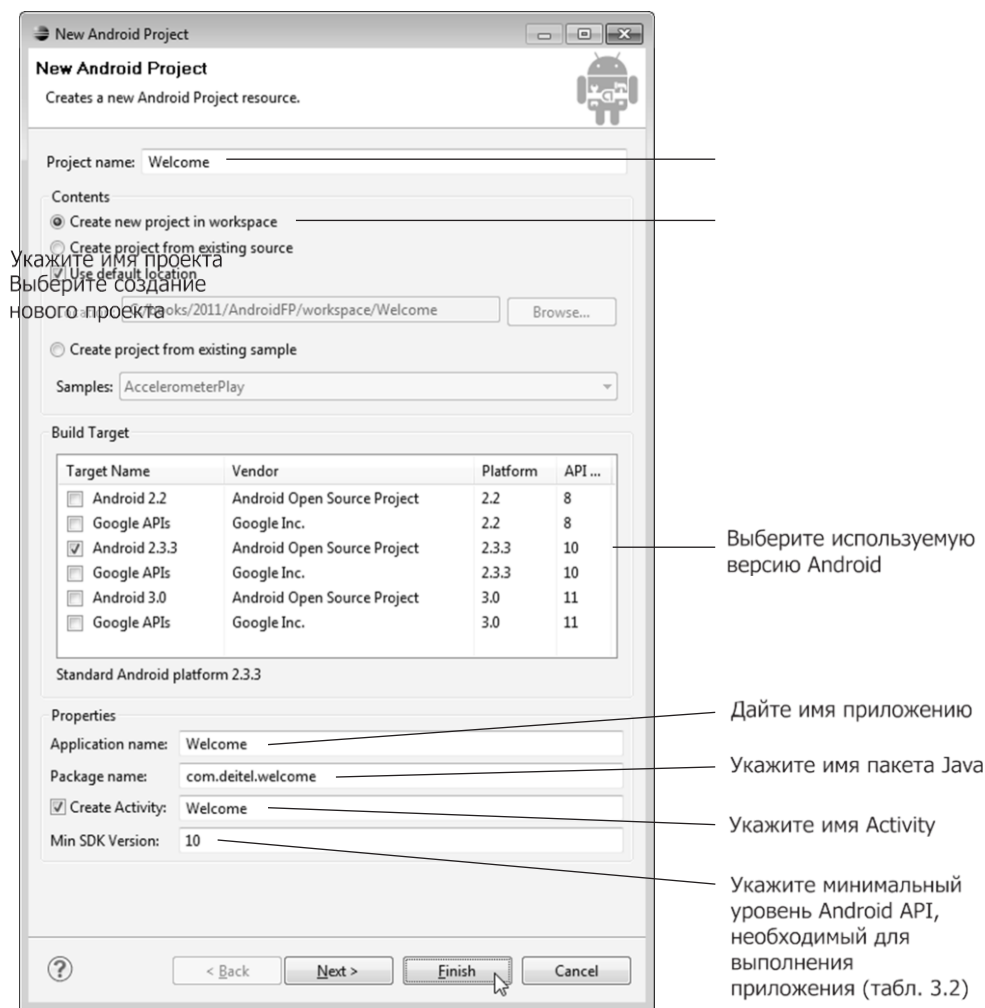


Рис. 3.3. Диалоговое окно New Android Project

В этом диалоговом окне укажите следующие сведения:

1. В поле Project name: (Название проекта:) введите название Welcome. Это имя корневого узла проекта на вкладке Package Explorer (Обозреватель пакетов) в среде Eclipse.
2. В разделе Contents (Содержимое) установите флажок Create new project in workspace (Создать новый проект в рабочей области), если нужно создать новый проект «с нуля», или флажок Create project from existing source (Создать проект на основе существующего), чтобы создать новый проект на основе существующих файлов исходного кода Java.
3. В разделе Build Target (Цель построения) выберите используемую версию Android. В большинстве рассмотренных в книге примеров используется версия 2.3.3, рекомендуется выбрать минимальную версию, поддерживаемую приложением, чтобы оно могло выполняться на возможно большем числе устройств.

В разделе Properties (Свойства) этого диалогового окна введите следующую информацию:

1. В поле Application name: (Имя приложения:) введите Welcome. Мы обычно даем приложениям те же названия, что и проектам, но этого делать не обязательно. Это название отображается на панели, находящейся в верхней части приложения, если она явно не *скрыта* приложением.
2. В Android используются традиционные для Java соглашения о наименовании пакетов, предусматривающие, что имя должно состоять как минимум из двух

частей (например, com.deitel). В поле Package name: (Имя пакета:) введите имя com.deitel.welcome. Мы используем наш домен deitel.com (в обратном порядке), за которым следует название приложения. Все классы и интерфейсы, которые создаются в качестве части приложения, будут помещены в этот пакет Java. Как Android, так и Android Market используют название пакета в качестве уникального идентификатора приложения.

3. В поле Create Activity: (Создать деятельность:) введите Welcome. Это имя класса, который управляет выполнением приложения. Начиная со следующей главы, мы будем изменять класс, чтобы реализовать функциональные свойства приложения.
4. В поле Min SDK Version: (Минимальная версия SDK:) укажите минимальный уровень API, который требуется для выполнения приложения. В результате приложение будет выполняться на устройствах с такой же версией API, либо более старшей. В этой книге мы обычно используем API уровня 10, который соответствует Android 2.3.3, либо API уровня 11, который соответствует Android 3.0. Чтобы выполнить приложение на платформе Android 2.2 или более старшей, выберите API уровня 8. *В подобных случаях следует убедиться в том, что приложение не использует функции, которые присущи более современным версиям Android.* Далее приведен список версий Android SDK и уровней API. *Другие версии SDK не рекомендуются к применению и не должны использоваться.* На веб-странице developer.android.com/resources/dashboard/platform-versions.html приведена текущая доля устройств Android, работающая под соответствующей версией платформы.
5. Чтобы завершить создание проекта, щелкните на кнопке Finish (Готово).

Таблица 2.1. Версии Android SDK и уровни API
(developer.android.com/sdk/index.html):

Версия Android SDK	Уровень API
3.0	11
2.3.3	10
2.2	8
2.1	7
1.6	4
1.5	3

ПРИМЕЧАНИЕ.

Возможно, в процессе загрузки Android SDK вы увидите сообщения об ошибках проекта.

Окно Package Explorer. Как только создается (или открывается) проект, в левой части интегрированной среды разработки появляется окно Package Explorer (Обозреватель пакетов), обеспечивающее доступ ко всем файлам проекта. На рис. 2.4 показано содержимое проекта для приложения Welcome. В узле Welcome представлен проект. Можно одновременно открыть несколько проектов в окне интегрированной среды разработки (IDE). Каждый из проектов имеет собственный узел верхнего уровня.

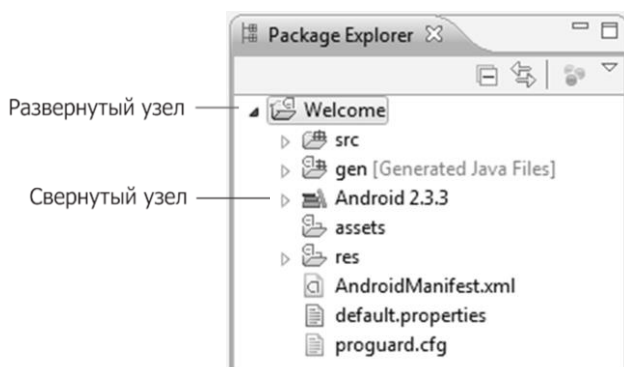


Рис.2.4. Окно Package Explorer

В узле проекта содержимое проекта организовано в виде различных файлов и папок, в том числе:

- src — папка, включающая исходные файлы проекта Java;
- gen — папка, содержащая файлы Java, сгенерированные IDE;
- Android 2.3.3 — папка, включающая версию Android framework, выбранную при создании приложения;
- res — папка, в которой находятся *файлы ресурсов*, связанные с приложением, такие как макеты GUI и изображения, используемые в приложении.

Иные файлы и папки будут рассмотрены в других главах.

2.3 Создание графического интерфейса пользователя приложения Welcome с помощью визуального макетного редактора модуля ADT

А теперь создадим графический интерфейс пользователя для приложения Welcome. С помощью редактора ADT Visual Layout Editor можно создать графический интерфейс пользователя путем перетаскивания в окно приложения компонентов GUI, таких как Buttons, TextViews, ImageViews и др. При создании Android-приложения с помощью Eclipse макет интерфейса пользователя хранится в XML-файле, который по умолчанию называется main.xml. Благодаря определению элементов GUI в XML-файле можно легко отделить логику приложения от его презентации. Файлы макетов являются ресурсами приложения и хранятся в папке res проекта. Макеты GUI находятся в подпапке папки layout. После двойного щелчка на файле main.xml, находящегося в папке приложения /res/layout, появится представление Visual Layout Editor, выбранное по умолчанию (рис. 2.5). Чтобы просмотреть XML-содержимое файла (рис. 2.6), выберите вкладку с именем файла макета (в данном случае, main.xml). Чтобы вернуться обратно к представлению Visual Layout Editor, выберите вкладку Graphical Layout (Графический макет). Структура XML-файла будет рассмотрена ниже [1].

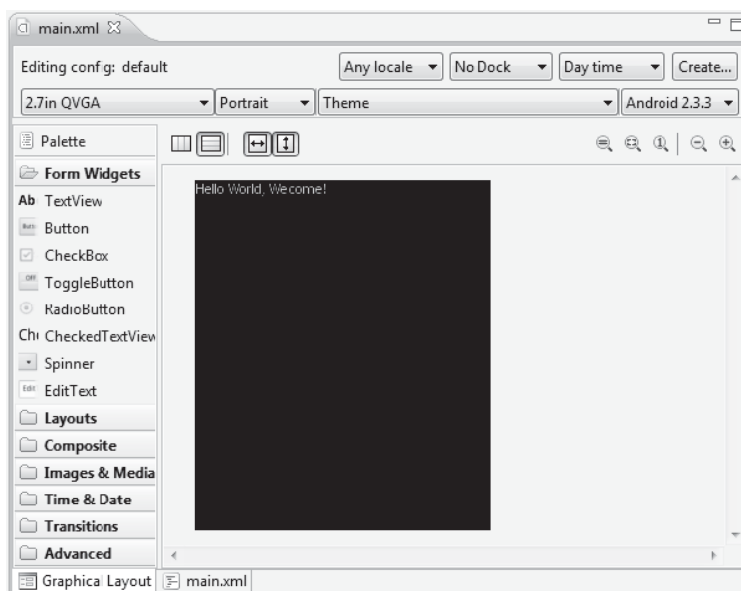


Рис. 2.5. Представление Visual Layout Editor заданного по умолчанию интерфейса приложения.



Вкладка XML view

Рис. 2.6. XML-представление заданного по умолчанию GUI-приложения

Графический интерфейс пользователя, заданный по умолчанию. Графический интерфейс пользователя, заданный по умолчанию для нового Android-приложения, включает компонент `LinearLayout`. Этот компонент имеет черный фон и включает компонент `TextView`, содержащий текст "Hello World, Welcome!" (см. рис. 2.5). Компонент `LinearLayout` упорядочивает элементы GUI вдоль линии по вертикали или горизонтали. Компонент `TextView` позволяет отображать текст. Если запустить приложение на устройстве AVD или на реальном устройстве, появится заданный по умолчанию черный фон и белый текст на нем.

Ниже перечислены некоторые макеты, определяемые пакетом `android.widget1`. В дальнейшем будут рассмотрены дополнительные компоненты GUI, которые могут использоваться в макетах. Чтобы получить полный перечень этих компонентов, обратитесь к веб-сайту developer.android.com/reference/android/widget/package-summary.html.

Макеты Android (package `android.widget`):

- **FrameLayout.** Выделяет пространство для одного компонента. В этот макет можно добавить несколько компонентов, причем каждый из них будет отображаться над правым верхним углом макета. Последний добавленный компонент отображается сверху;

- **LinearLayout.** Расположение компонентов по горизонтали в одной строке или по вертикали в одном столбце;
- **RelativeLayout.** Расположение компонентов относительно друг друга или относительно родительского контейнера;
- **TableLayout.** Расположение компонентов в виде таблицы, состоящей из строк. Для форматирования столбцов таблицы можно воспользоваться макетом TableRow layout (подкласс макета LinearLayout).

ПРИМЕЧАНИЕ. Чтобы включить поддержку экранов, имеющих различные размеры и разрешения, для проектирования графического интерфейса пользователя используйте компоненты RelativeLayout и TableLayout.

Конфигурирование Visual Layout Editor для использования соответствующей библиотеки Android SDK. Если установлено несколько библиотек Android SDK, модуль ADT Plugin по умолчанию выбирает самую последнюю из них независимо от версии SDK, выбранной при создании проекта. Эта библиотека используется для проектирования графического интерфейса пользователя и находится на вкладке Graphical Layout (Графический макет). Обратите внимание на рис. 2.7, где выбрана библиотека Android 2.3.3 в раскрывающемся списке SDK selector (Выбор SDK), находящемся в правой верхней части вкладки Graphic Layout. В результате определяется разработка графического интерфейса пользователя для устройства Android 2.3.3.

Удаление и воссоздание файла main.xml. В процессе создания приложения для этой главы вместо заданного по умолчанию файла main.xml будет использоваться новый файл макета, RelativeLayout (в этом файле определяется относительное расположение компонентов). Чтобы заменить заданный по умолчанию файл main.xml, выполните следующие действия:

1. Убедитесь в том, что файл main.xml закрыт, потом щелкните на нем правой кнопкой мыши (в папке /res/layout проекта) и выберите команду Delete (Удалить) для удаления файла.
2. Чтобы отобразить диалоговое окно New... (Создать...), щелкните правой кнопкой мыши на папке макета и выберите команды New<Other... (Создать<Другой...).
3. В узле Android выберите параметр Android XML File (XML-файл Android) и щелкните на кнопке Next> (Далее>), чтобы отобразить диалоговое окно New Android XML File (Новый XML-файл Android).
4. Определите имя файла, его местоположение и корневой макет для нового файла main.xml (см. рис. 2.7), а потом щелкните на кнопке Finish (Готово).

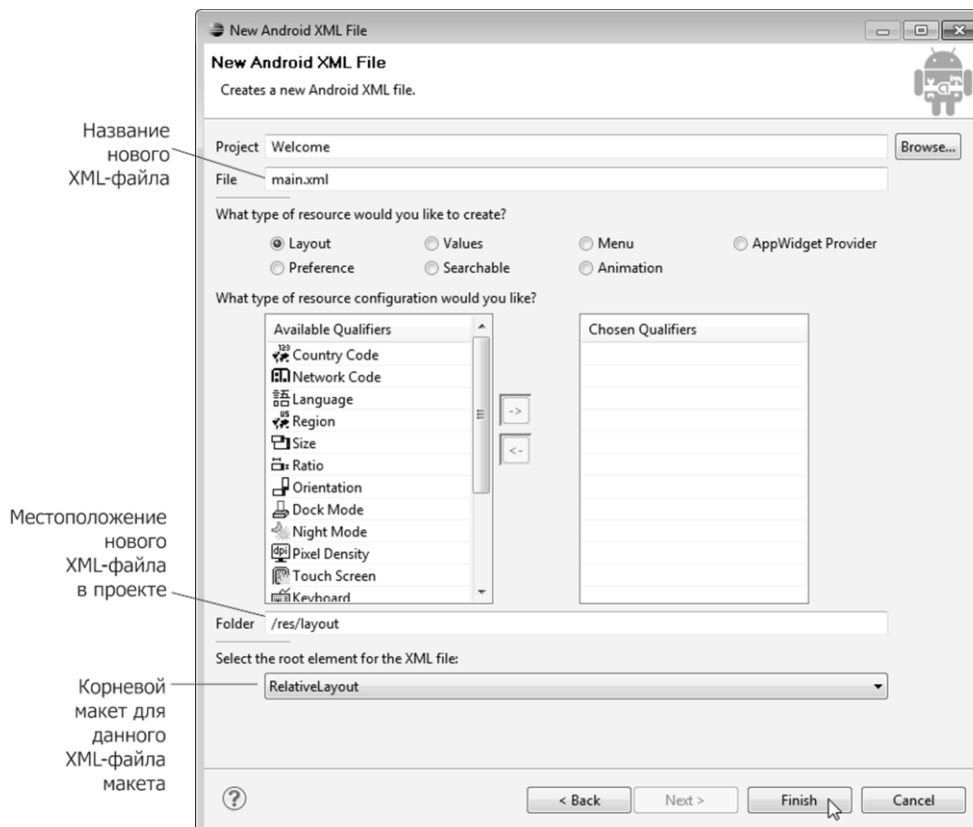


Рис. 2.7. Создание нового файла main.xml в диалоговом окне New Android XML File.

Настройка размера и разрешения экрана для Visual Layout Editor. На рис. 2.8 показан новый файл main.xml в окне редактора Visual Layout Editor. Поскольку операционная система Android может выполняться на множестве различных устройств, в комплект поставки Visual Layout Editor входит ряд различных конфигураций устройств, представляющих разные размеры и разрешения экрана. Эти настройки выбираются в раскрывающемся списке Device Configurations (Конфигурации устройства). Этот раскрывающийся список находится в верхней левой части вкладки Graphic Layout (рис. 3.8). Если стандартные конфигурации не соответствуют устройству, для которого разрабатывается приложение, можно создать собственную конфигурацию устройства «с нуля» либо путем копирования и изменения одной из имеющихся конфигураций.

Для создания примеров в книге мы использовали основное тестовое устройство Samsung Nexus S, которое снабжено 4-дюймовым экраном с разрешением 480×800 пикселей (WVGA). Рекомендуется разрабатывать масштабируемую библиотеку Android GUI, которая позволит корректно формировать изображения на экранах различных устройств. Благодаря свойству масштабируемости не требуется точное соответствие области проектирования Visual Layout Editor экранам физических устройств. Достаточно выбрать подобную конфигурацию устройства. На рис. 2.8 выбран параметр 3.7in WVGA (Nexus One). Экран этого устройства имеет то же разрешение WVGA, что и Nexus S, но немного меньший размер экрана. Экраны многих современных смартфонов имеют разрешение 480×800 или 480×854 пикселей.

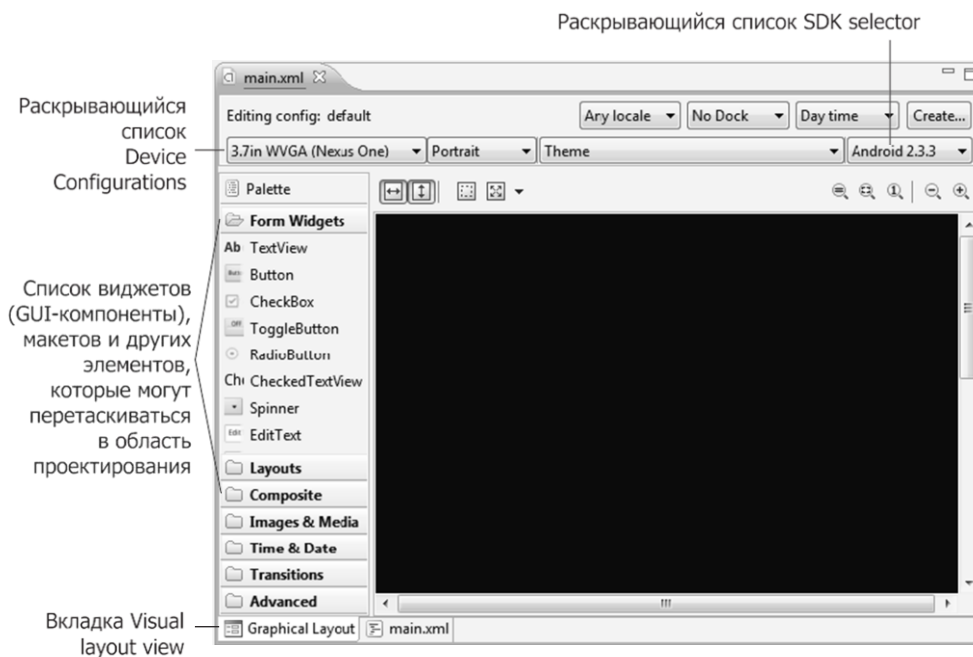


Рис. 2.8. Представление Visual Layout Editor для заданного по умолчанию GUI приложения.

Размеры и разрешение изображений и экрана. Поскольку экраны устройств Android имеют различные размеры, разрешения и пиксельные плотности экрана (выражается в точках на дюйм или DPI), поддерживаются изображения (и другие ресурсы), которые выбираются операционной системой в соответствии с пиксельной плотностью экрана устройства. Поэтому в папке `res` проекта находятся три подпапки — `drawable-hdpi` (высокая плотность), `drawable-mdpi` (средняя плотность) и `drawable-ldpi` (низкая плотность). В этих папках хранятся изображения с различными пиксельными плотностями:

- `Ldpi`: низкая плотность — примерно 120 точек на дюйм;
- `Mdpi`: средняя плотность — примерно 160 точек на дюйм;
- `Hdpi`: высокая плотность — примерно 240 точек на дюйм;
- `Xhdpi`: экстремально высокая плотность — примерно 320 точек на дюйм;
- `Nodpi`: отсутствие масштабирования ресурсов независимо от плотности экрана.

Изображения для устройств, которые имеют ту же самую пиксельную плотность, что и тестовое устройство, находятся в папке `drawable-hdpi`. Изображения, предназначенные для экранов, имеющих среднюю и низкую пиксельную плотность, находятся в папках `drawable-mdpi` и `drawable-ldpi` соответственно. Как и в случае с Android 2.2, можно добавить подпапку `drawable-xhdpi` в папку приложения `res`, чтобы представить экраны, которые имеют экстремально высокие пиксельные плотности. В Android может выполняться масштабирование изображений в целях приведения в соответствие с различными экранными плотностями.

ПРИМЕЧАНИЕ. Чтобы получить подробную информацию о поддержке нескольких экранов и размеров экранов на Android, обратитесь к веб-сайту developer.android.com/guide/practices/screens_support.html.

ПРИМЕЧАНИЕ. Чтобы улучшить отображение на экране устройства с высокой пиксельной плотностью, следует выбирать изображения с высоким разрешением. Изображения с низким разрешением могут некорректно масштабироваться.

Шаг 1. Добавление изображений в проект. А теперь приступим к созданию приложения `Welcome`. Для создания приложения будет использован редактор Visual Layout Editor и окно Outline (Макет), затем будет подробно рассмотрена структура

сгенерированного XML-кода. В следующих главах мы приступим к непосредственному редактированию XML.

ПРИМЕЧАНИЕ. Многие профессиональные Android-разработчики предпочитают создавать GUI непосредственно в XML, а затем с помощью редактора Visual Layout Editor просматривать результаты его выполнения. В процессе ввода XML-кода в представлении XML Eclipse предлагает функции авто завершения, отображая имена компонентов, атрибутов и значения, которые соответствуют вводимым именам и значениям. Благодаря этому можно быстро и корректно вводить XML-код.

Для данного приложения в проект нужно добавить изображения Deitel bug (bug.png) и логотипа Android (android.png). Эти изображения находятся в папке images вместе с примерами книги. Чтобы добавить изображения в проект, выполните следующие действия:

1. В окне Package Explorer откройте папку проекта res.
2. Найдите и откройте папку images, которая находится среди примеров книги, потом перетащите изображения в папку, которая находится в подпапке drawable-hdpi папки res. Теперь изображения могут использоваться в приложении.

Шаг 2. Изменения свойства Id макета RelativeLayout. С помощью окна Properties (Свойства) можно сконфигурировать свойства выбранного макета или компонента без непосредственного редактирования XML-кода. Если окно Properties не отображается, отобразите его двойным щелчком на пункте RelativeLayout в окне Outline (Структура). Можно также выполнить команды Window☿ Show View☿Other... (Окно☿Показать представление☿Другое...), затем выбрать параметр Properties из узла General (Общий) в диалоговом окне Show View (Показать представление). Для выбора макета или компонента щелкните на нем в окне редактора Visual Layout Editor либо выберите его узел в окне Outline (рис. 2.9). Окно Properties не может быть использовано, если макет отображается в представлении XML.



Рис. 2.9. Иерархическое представление GUI в окне Outline

Переименуйте каждый макет и компонент. Это особенно полезно в том случае, если макет или компонент будут подвергаться программной обработке (см. следующие главы). Имя каждого объекта определяется с помощью его свойства Id. Это свойство может применяться для получения доступа и изменения компонента, при этом не нужно знать точное местоположение компонента в XML-коде. Как будет показано позднее, свойство Id также может применяться для указания относительного позиционирования компонентов в макете RelativeLayout.

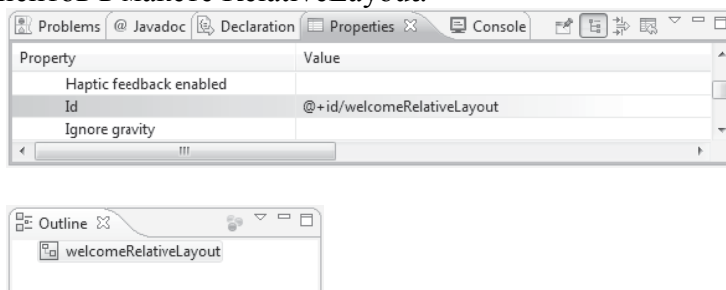


Рис. 2.10. Окно Properties, отображаемое после изменения свойства Id компонента RelativeLayout.

Выберите компонент RelativeLayout, затем, выполнив прокрутку в окне Properties, выберите свойство Id и присвойте ему значение:

@+id/welcomeRelativeLayout

Символ + в конструкции @+id определяет создание нового идентификатора (имя переменной), который указан справа от символа /. На экране появятся окна Properties и Outline (рис. 2.10).

Шаг 3. Присваивание свойству Background значения RelativeLayout. По умолчанию в качестве фонового цвета макета выбирается черный, но его можно изменить (например, на белый). Любой цвет представляет собой результат смешения красного, синего и зеленого компонентов, которые называются RGB-компонентами. Каждый из этих компонентов может принимать целочисленное значение в диапазоне от 0 до 255. Первый компонент задает количество красного в общем цвете, второй — количество зеленого, а третий — количество синего цвета. Во время работы в интегрированной среде разработки (IDE) для определения цвета используется шестнадцатеричный формат. То есть RGB-компоненты представлены в виде значений в диапазоне 00–FF.

Чтобы изменить фоновый цвет, найдите свойство Background в окне Properties и присвойте ему значение #FFFFFF (рис. 2.11). Это значение представляет белый цвет в шестнадцатеричном формате. Формат #RRGGBB — это пары шестнадцатеричных чисел, которые представляют красный, зеленый и синий цвета соответственно. В Android также поддерживаются альфа-значения (прозрачность), которые изменяются в диапазоне 0–255. Значение 0 соответствует полной прозрачности, а значение 255 — полной не прозрачности. Если планируется использовать альфа-значения, укажите цвет в формате #AARRGGBB, где первые две шестнадцатеричных цифры соответствуют альфа-значению. Если обе цифры каждого компонента цвета одинаковы, можно воспользоваться форматом #RGB или #ARGB. Например, значение #FFF трактуется как #FFFFFF.

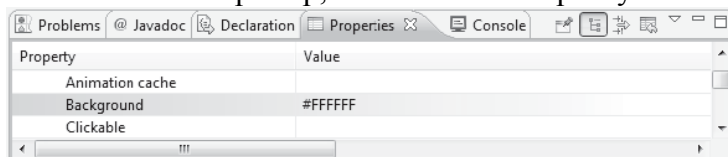


Рис. 2.11. Окно Properties после изменения значения свойства Background компонента RelativeLayout.

Шаг 4. Добавление компонента TextView. На этом шаге добавим в пользовательский интерфейс компонент TextView. В списке Form Widgets (Виджеты форм), находящемся в левой части окна Visual Layout Editor, найдите компонент TextView и перетащите его в область проектирования (рис. 2.12). После добавления нового компонента в пользовательский интерфейс происходит его автоматический выбор и отображение свойств в окне Properties.

Компонент TextView с текстом, заданным по умолчанию

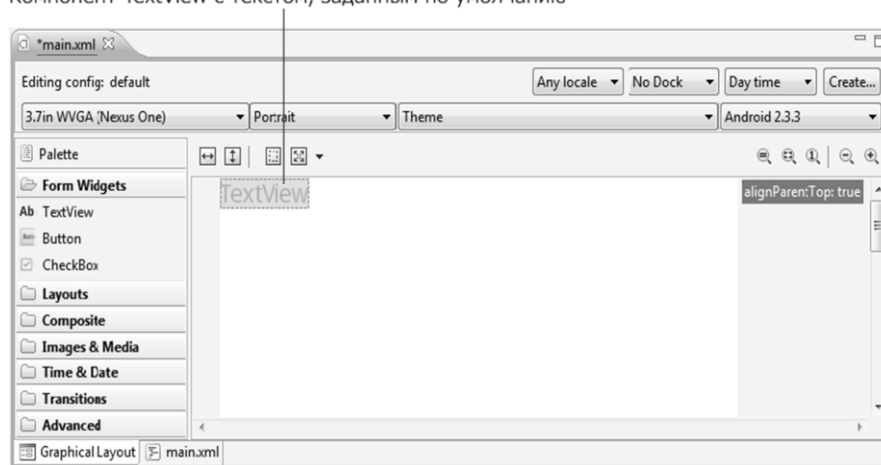


Рис. 2.12. Компонент TextView с текстом, заданным по умолчанию.

Шаг 5. Настройка свойства Text компонента TextView с помощью строковых ресурсов. Согласно документации Android по ресурсам приложений (developer.android.com/guide/topics/resources/index.html) считается хорошим тоном хранить строки, массивы строк, изображения, цвета, размеры шрифтов, размерности и другие ресурсы приложения так, чтобы их можно было использовать отдельно от кода приложения. Например, после экстернализации цветовых значений все компоненты, использующие один и тот же цвет, могут быть перекрашены путем простого изменения значения цвета в центральном файле ресурса.

Если нужно локализовать приложения, создав версии на нескольких разных языках, сохраните строки отдельно от кода приложения. В результате у вас в дальнейшем появится возможность простого изменения этих строк. В папке `res` проекта находится подпапка `values`, в которой помещен файл `strings.xml`. Этот файл применяется для хранения строк. Чтобы сформировать локализованные строки для других языков, создайте отдельные папки `values` для каждого используемого языка.

Например, в папке `values-fr` может находиться файл `strings.xml` для французского языка, а в папке `values-es` — файл `strings.xml` для испанского языка. Можно также формировать названия этих папок с учетом информации о регионе. Например, в папке `values-en-rUS` может находиться файл `strings.xml` для американского диалекта английского языка, а в папке `values-en-rGB` — файл `strings.xml` для диалекта английского языка, используемого в Великобритании.

Чтобы получить дополнительные сведения о локализации, обратитесь к ресурсам:

- developer.android.com/guide/topics/resources/;
- providing-resources.html#AlternativeResources;
- developer.android.com/guide/topics/resources/localization.html.

Чтобы установить значение свойства Text компонента TextView, создадим новый строковый ресурс в файле `strings.xml`.

1. Выберите компонент TextView.
2. В окне Properties найдите свойство Text, щелкните на значении, заданном по умолчанию, затем щелкните на кнопке с многоточием. Эта кнопка находится в правой части поля значения свойства и служит для отображения диалогового окна Resource Chooser (Выбор ресурсов).
3. В диалоговом окне Resource Chooser щелкните на кнопке New String... (Новая строка...), чтобы отобразить диалоговое окно Create New Android String (Создать новую строку Android), показанное на рис. 2.13.
4. Заполните поля String и New R.string (см. рис. 2.13), потом щелкните на кнопке OK, чтобы скрыть диалоговое окно Create New Android String и вернуться к окну Resource Chooser.
5. Автоматически выбирается новый строковый ресурс `welcome`. Щелкните на кнопке OK для выбора этого ресурса.

После выполнения указанных выше шагов в окне Properties появляется свойство Text (рис. 2.14). Запись в форме `@string` свидетельствует о том, что существующий строковый ресурс может быть выбран в файле `strings.xml`, а имя `welcome` говорит о том, что строковый ресурс выбран в данный момент времени.

Основное преимущество определения строковых значений заключается в облегчении локализации приложения, осуществляемой путем создания дополнительных файлов XML-ресурсов для других языков. В каждом файле используется одно и то же имя в поле New R.string и поддерживается интернационализированная строка в поле String. Потом Android может выбрать соответствующий файл ресурса, основываясь на языке, предпочтительном для пользователя устройства. Дополнительные сведения о локализации можно найти на веб-сайте developer.android.com/guide/topics/resources/localization.html.

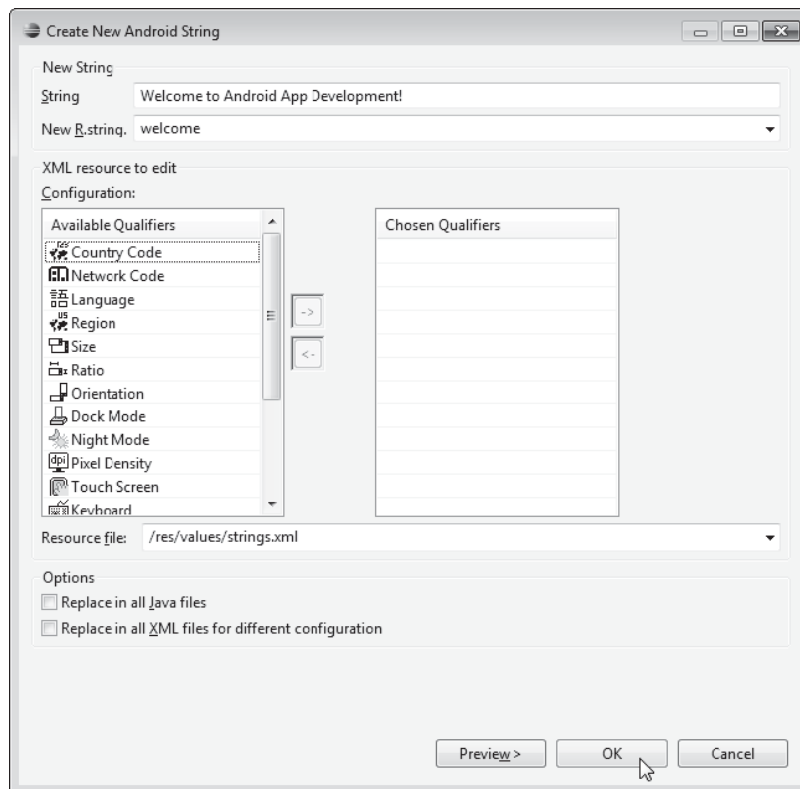


Рис. 2.13. Окно Create New Android String

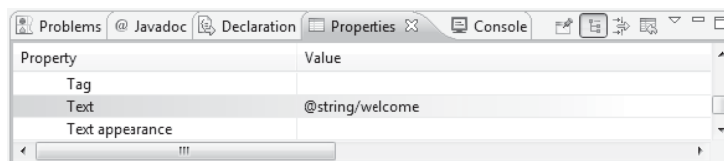


Рис. 2.14. Вид окна Properties после изменения свойства Text компонента TextView

Шаг 6. Настройка свойств Text size и Padding top компонента TextView — пиксели, независимые от плотности и от масштабирования. Размеры компонентов GUI и текста на экране Android могут определяться с помощью различных единиц измерения (табл. 2). Документация, описывающая различные размеры экранов, находится на веб-сайте по адресу developer.android.com/guide/practices/screens_support.html и рекомендует для определения размеров компонентов GUI и других экранных элементов использовать пиксели, независимые от плотности, а размеры шрифтов задавать с помощью пикселей, независимых от масштабирования.

Таблица 2.2. Единицы измерения

Единица измерения	Описание
px	Пиксель
dp или dip	Пиксель, независимый от плотности
sp	Пиксель, независимый от масштабирования
in	Дюймы
mm	Миллиметры

Задание размеров в пикселях, независимых от плотности (dp или dip), позволяет платформе Android автоматически масштабировать графический интерфейс пользователя в зависимости от плотности пикселей экрана физического устройства.

Размер пикселя, независимого от плотности, эквивалентен размеру физического пикселя на экране с разрешением 160 dpi (точек на дюйм). На экране с разрешением 240

dpi размер пикселя, независимого от плотности, будет масштабироваться на коэффициент 240/160 (то есть 1,5). Таким образом, компонент, размер которого составляет 100 пикселей, независимых от плотности, будет масштабирован до размера в 150 физических пикселей на таком экране. На экране с разрешением 120 точек на дюйм каждый независимый от плотности пиксель масштабируется на коэффициент 120/160 (то есть 0,75). Значит, 100 независимых от плотностей пикселей превратятся на таком экране в 75 физических пикселей. Пиксели, независимые от масштаба, масштабируются так же, как и пиксели, независимые от плотности, но их масштаб зависит также и от пред-почитаемого размера шрифта, выбираемого пользователем.

А теперь увеличим размер шрифта для компонента TextView и добавим небольшой отступ над компонентом TextView, чтобы отделить текст от края экрана устройства.

1. Чтобы изменить размер шрифта, выберите компонент TextView, а потом присвойте свойству Text size значение 40sp.
2. Чтобы увеличить отступ между верхним краем макета и компонентом TextView, выберите свойство Layout margin top property в разделе Misc (Разное) окна Properties и присвойте ему значение 10dp.

Шаг 7. Настройка дополнительных свойств компонента TextView.
Сконфигурируйте дополнительные свойства компонента TextView следующим образом:

1. Свойству Id присвойте значение @+id/welcomeTextView.
2. Свойству Text color присвойте значение #00F (синий).
3. Свойству Text style присвойте значение bold. Чтобы выполнить эту операцию, щелкните в области поля Value этого свойства, затем щелкните на кнопке с многоточием для отображения диалогового окна, в котором выбирается стиль шрифта. Установите флажок bold, потом щелкните на кнопке ОК для выбора стиля текста.
4. Чтобы отцентрировать текст компонента TextView при его расположении в нескольких строках свойству Gravity присвойте значение center. Для выполнения этой операции щелкните в области поля Value этого свойства, потом щелкните на кнопке с многоточием для отображения диалогового окна параметров свойства Gravity (рис. 2.15). Установите флажок center и щелкните на кнопке ОК для выбора значения.
5. На экране снова появится окно Visual Layout Editor (рис. 2.16).

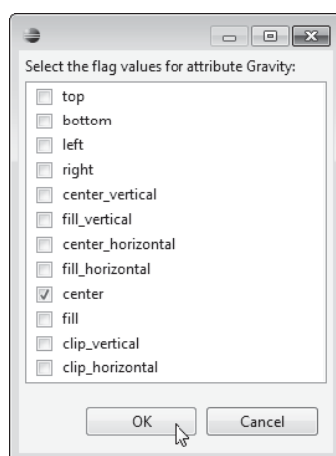


Рис. 2.15. Параметры атрибута gravity для объекта.

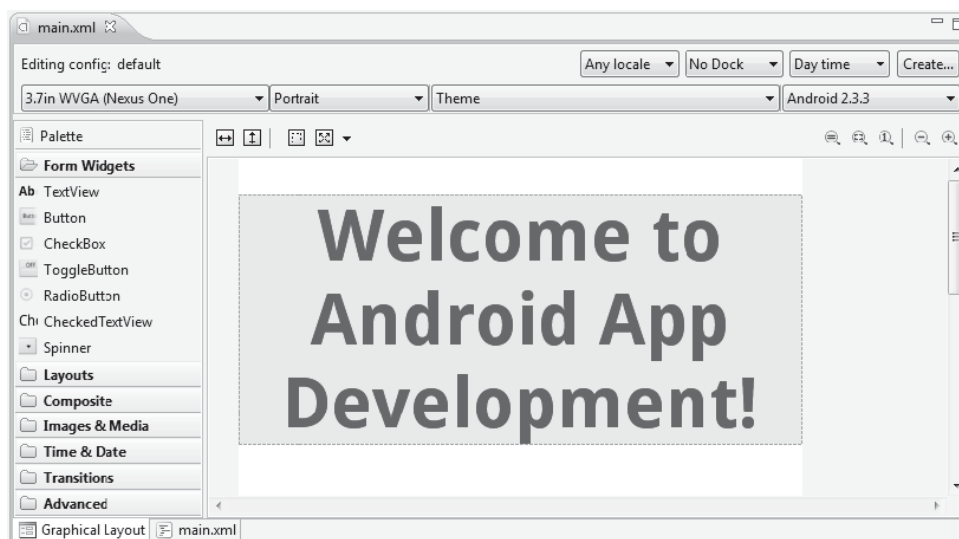


Рис. 2.16. Вид окна Visual Layout Editor после завершения конфигурирования компонента TextView.

Шаг 8. Отображение логотипов Android и Deitel Bug с помощью компонентов ImageViews. А теперь в графический интерфейс пользователя добавим два компонента ImageViews, с помощью которых отображаются два изображения, добавленные в проект на шаге 1. После первого перетаскивания компонента ImageView в окно редактора Visual Layout Editor ничего не происходит. Поэтому для добавления компонентов ImageViews используется окно Outline. Выполните следующие действия:

1. Перетащите компонент ImageView, находящийся в категории Images & Media (Изображения и медиа ресурсы) палитры Visual Layout Editor, в окно Outline (Макет), как показано на рис. 2.17.
2. Новый компонент ImageView появится ниже узла welcomeTextView. Вы не обнаружите какого-либо признака, свидетельствующего о том, что компонент появился ниже компонента TextView в GUI. Это означает, что требуется установить значение свойства Layout below, чем мы и займемся в скором времени.

ПРИМЕЧАНИЕ. Если перетащить компонент ImageView поверх элемента welcomeTextView и удерживать его в течение некоторого времени, вокруг элемента welcomeTextView появится зеленый прямоугольник с секциями. Если перетащить компонент ImageView поверх этих секций, редактор Visual Layout Editor может установить относительное позиционирование.



Рис. 2.17. Перетаскивание компонента ImageView в окно Outline.

1. Свойству Id компонента ImageView присвойте значение `@+id/droidImageView`. В окне Outline отобразится название объекта `droidImageView`.
2. Свойству Layout below объекта `droidImageView` присвойте значение `@id/welcomeTextView`. В результате компонент ImageView будет расположен ниже объекта `welcomeTextView`. Чтобы выполнить эту операцию, щелкните в области поля Value этого свойства, затем щелкните на кнопке с многоточием, чтобы отобразить диалоговое окно Reference Chooser (Выбор ссылок), показанное на рис. 2.18. В узле ID

- находятся названия объектов GUI. Раскройте узел ID и выберите объект `welcomeTextView`.
3. Присвойте свойству `Layout center horizontal` объекта `droidImageView` значение `true`, чтобы отцентрировать компонент `ImageView` внутри макета.
 4. Присвойте свойству `Src` объекта `droidImageView` изображение, которое должно быть отображено. Для выполнения этой операции щелкните в области поля `Value` данного свойства, а потом щелкните на кнопке с многоточием, чтобы отобразить диалоговое окно `Reference Chooser` (рис. 2.19). В узле `Drawable` расположены ресурсы, находящиеся в папках приложения `drawable`, которые находятся в папке `res`. В диалоговом окне раскройте узел `Drawable` и выберите параметр `android`, который представляет изображение `android.png`.
 5. Повторите предыдущие пункты 1–5 для создания компонента `bugImageView`. Присвойте свойству `Id` компонента значение `@+id/bugImageView`, свойству `Src` — значение `bug`, а свойству `Layout below` — значение `droidImageView`.
 6. Теперь окно `Visual Layout Editor` будет выглядеть так, как показано на рис. 2.20.

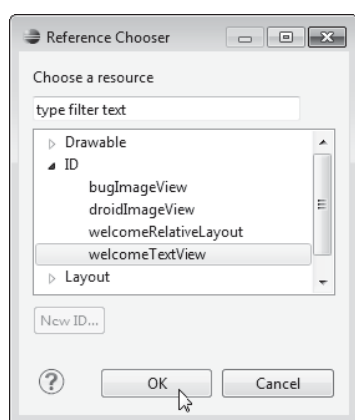


Рис. 2.18. Выбор значения для свойства `Layout below` объекта `droidImageView`.

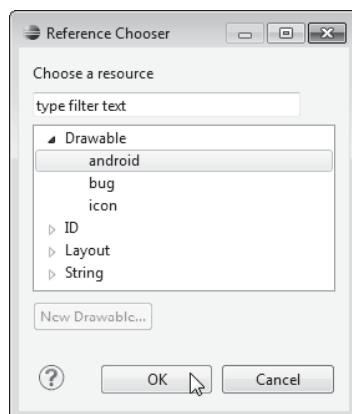


Рис. 2.19. Выбор значения для свойства `Src` объекта `droidImageView`.

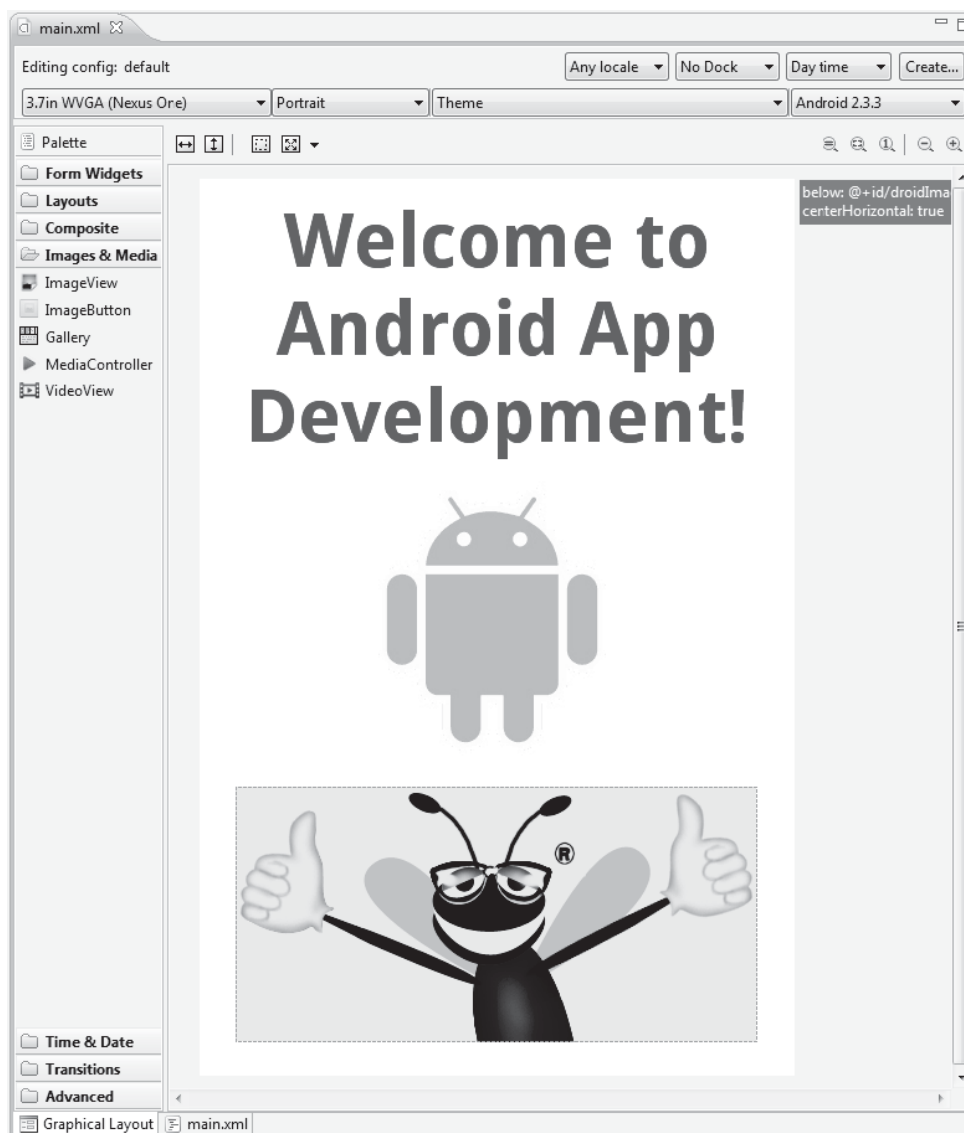


Рис. 2.20. Окно Visual Layout Editor после завершения конфигурирования GUI 3.6.
Структура файла main.xml 135

2.4 Структура файла main.xml

Язык XML позволяет естественным образом описать контент GUI. В формате, воспринимаемом компьютером и человеком, этот язык «говорит» о том, какие макеты и компоненты вы желаете использовать, а также позволяет определить их атрибуты, такие как размер, положение и цвет. Потом подключаемый модуль ADT Plugin анализирует XML и генерирует код, который реализует графический интерфейс пользователя. В листинге 2.1 приведен финальный код файла main.xml. Мы выполнили форматирование XML-кода и добавили комментарии, чтобы повысить удобство чтения XML-кода (чтобы выполнить форматирование, воспользуйтесь командой Eclipse Source ϕ Format (Исходный код ϕ Формат). В процессе чтения XML-кода нетрудно заметить, что имя каждого атрибута XML, состоящее из нескольких слов, не включает пробелы, в то время как названия соответствующих свойств в окне Properties могут включать пробелы. Например, атрибут XML android:paddingTop соответствует свойству Padding top в окне Properties. В среде IDE имена свойств, состоящие из нескольких слов, отображаются отдельно с целью улучшения читабельности.

Листинг 2.1. XML-разметка приложения Welcome.

```

1  <?xml version="1.0" encoding="utf-
8"?> 2  <!-- main.xml -->

3  <! - XML-разметка приложения Welcome. --> 4

5  <! - RelativeLayout, включающий компоненты GUI приложения
-->
6  <RelativeLayout
    xmlns:android=http://schemas.android.com/apk/res/android
7      android:layout_width="match_parent"
8      android:layout_height="match_parent"
9      android:id="@+id/welcomeRelativeLayout"
        android:background="#FFFFFF">
10

11  <! -      TextView, отображающий "Welcome to Android App
Development!" -->
12  <TextView
    android:layout_width="wrap_content"
13      android:layout_height="wrap_content"
14      android:text="@string/welcome"
15      android:textSize="40sp"
        android:id="@+id/welcomeTextView"
16      android:textColor="#00F"
        android:textStyle="bold"
17      android:layout_centerHorizontal="true"
        android:gravity="center"
18      android:layout_marginTop="10dp"></TextView>
19

20  <! - ImageView, который отображает логотип Android -->
21  <ImageView    android:layout_height="wrap_content"
22      android:layout_width="wrap_content"
        android:id="@+id/droidImageView"
23      android:layout_centerHorizontal="true"
24      android:src="@drawable/android"
25      android:layout_below="@id/welcomeTextView"></ImageView>
26

27  <! - ImageView, отображающий логотип Deitel bug -->
28  <ImageView    android:layout_height="wrap_content"
29      android:layout_width="wrap_content"
        android:id="@+id/bugImageView"
30      android:src="@drawable/bug"
31      android:layout_below="@id/droidImageView"
32      android:layout_centerHorizontal="true"></ImageView>
33 </RelativeLayout>

```

welcomeRelativeLayout. Блок welcomeRelativeLayout (строки 6–33) содержит все компоненты GUI приложения. Открывающий тег XML (строки 6–9) устанавливает различные атрибуты RelativeLayout.

В строке 6 используется атрибут xmlns, который указывает все элементы в документе, относящиеся к пространству имен android XML. Этот атрибут обязателен для указания и автоматически генерируется IDE при создании любого файла XML- разметки.

В строках 7–8 указывается значение match_parent для атрибутов android:layout_width и android:layout_height. В результате макет занимает всю высоту и ширину родительского элемента макета (элемента, в который вложен данный макет). В данном случае RelativeLayout является корневым узлом документа XML, в результате чего макет занимает все пространство экрана (помимо строки состояния).

В строке 9 указываются значения атрибутов `android:id` и `android:background` для компонента `welcomeRelativeLayout`.

welcomeTextView. Первый элемент компонента `welcomeRelativeLayout` — это `welcomeTextView` (строки 12–18).

В строках 12 и 13 атрибутам `android:layout_width` и `android:layout_height` присваивается значение `wrap_content`. Это значение свидетельствует о том, что представление будет достаточно большим, чтобы вместить все содержимое, включая значения отступа, с помощью которых определяется отступ вокруг содержимого.

В строке 14 устанавливается атрибут `android:text` таким образом, чтобы ему соответствовал строковый ресурс `welcome`, который был создан на шаге 5 раздела 2.2. В строке 15 атрибуту `android:textSize` присваивается значение `40sp`, а атрибуту `android:id` — значение `"@+id/welcomeTextView"`.

В строке 16 атрибуту `android:textColor` присваивается значение `"#00F"` (выделенный текст), а атрибуту `android:textStyle` — значение `"bold"`.

В строке 17 атрибуту `android:layout_centerHorizontal` присваивается значение `"true"`, которое центрирует компонент по горизонтали в макете. Атрибуту `android:gravity` присваивается значение `"center"`, которое центрирует текст компонента `TextView`. Атрибут `android:gravity` определяет, каким образом позиционируется текст относительно ширины и высоты компонента `TextView`, если текст будет меньше, чем `TextView`.

В строке 18 атрибуту `android:marginTop` присваивается значение `10dp`, в результате чего добавляется дополнительное пространство между верхней частью компонента `TextView` и экрана.

droidImageView. Последние два элемента, вложенные в компонент `welcomeRelativeLayout`, — это `droidImageView` (строки 21–25) и `bugImageView` (строки 28–32). Для двух компонентов `ImageViews` используются одни и те же атрибуты, поэтому будут рассмотрены лишь атрибуты компонента `droidImageView`.

В строках 21 и 22 атрибутам `android:layout_width` и `android:layout_height` присваивается значение `wrap_content`. В строке 22 атрибуту `android:id` присваивается значение `"@+id/droidImageView"`.

В строке 23 атрибуту `android:layout_centerHorizontal` присваивается значение `"true"`, чтобы центрировать компонент в области макета.

В строке 24 атрибуту `android:src` присваивается ресурс, находящийся в папке `drawable` и имеющий название `android`. Этот атрибут представляет изображение `android.png`.

В строке 25 атрибуту `android:layout_below` присваивается значение `"@id/welcomeTextView"`.

Компонент `RelativeLayout` определяет положение каждого компонента относительно других компонентов. В этом случае компонент `ImageView` следует за компонентом `welcomeTextView`.

Чтобы выполнить приложение на виртуальном устройстве `Android Virtual Device (AVD)`, щелкните правой кнопкой мыши на корневом узле приложения в окне `Package Explorer` и выполните команды `Run As Android Application` (Выполнить как Приложение Android). На рис. 2.21 показано запущенное приложение.

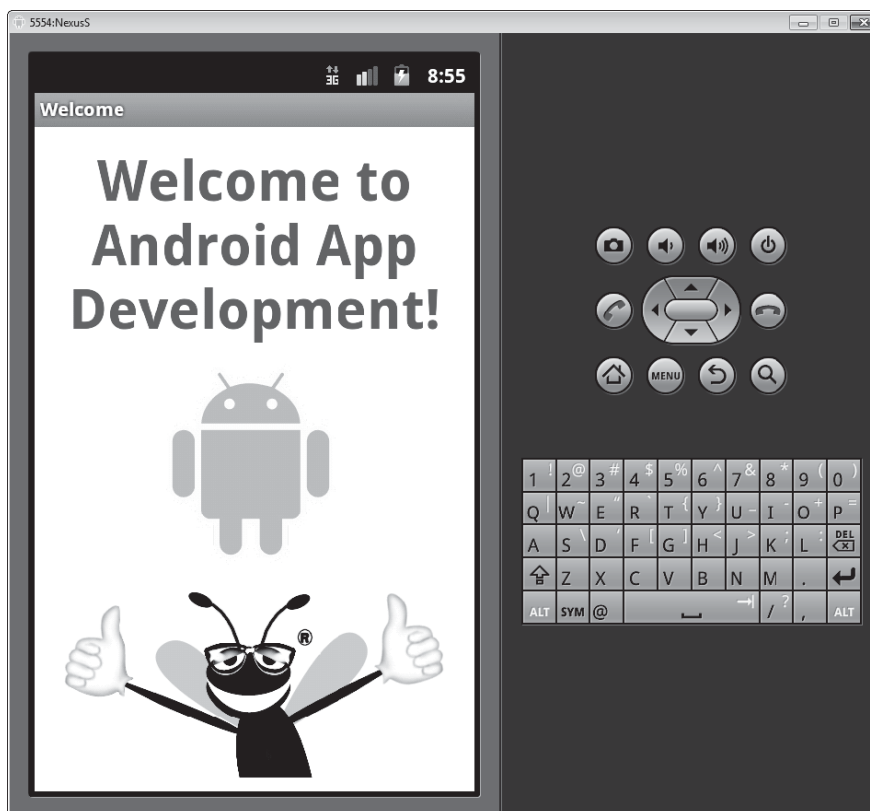


Рис. 2.21. Приложение Welcome, выполняющееся на виртуальном устройстве AVD

2.5 Интегрированная среда разработки Eclipse – советы и хитрости

В этом приложении собрано множество советов и хитростей, которые вы можете взять на вооружение для эффективной работы со средой разработки Eclipse. Эти советы и хитрости связаны с выполнением распространенных задач при разработке Android-приложений, однако они могут также применяться при разработке Java-приложений в среде разработки Eclipse [2].

СОЗДАНИЕ НОВЫХ КЛАССОВ И МЕТОДОВ. Вы можете быстро создать новы»» класс и соответствующий файл с исходным кодом, щелкнув правой кнопкой мыши по названию пакета, в котором должен быть создан этот класс, и выбрав в появившемся контекстном меню команду New => Class (Создать => Класс). В появившемся диалоговом окне New Java Class (Новый класс Java) введите имя создаваемого класса, выберите его суперкласс и интерфейсы, а также укажите, нужно ли создавать стандартные комментарии и методы-заглушки для конструкторов или абстрактных методов суперкласса.

Аналогичным образом вы можете быстро создать методы-заглушки, щелк- n>в правом кнопкой мыши по названию класса или в окне редактора с исходным кодом класса и выбрав в появившемся контекстном меню команду Source => Override/Implement Methods (Исходный код => Переопределить/ Реапнзовать методы). В открывшемся диалоговом окне Override/Implement Methods (Переопределение/Реализация методов) выберите методы, для которых нужно создать «заглушки», и место в исходном коде для добавления методов-заглушек, а также укажите, нужно ли генерировать стандартные блоки комментариев.

ОРГАНИЗАЦИЯ ИНСТРУКЦИЙ IMPORT. Когда вы в первый раз обращаетесь к некоторому классу в вашем коде, вы можете установить указатель мыши на название этого нового класса и в появившемся всплывающем меню щелкнуть по ссылке «Import “имя_ класса” (название пакета)» (Импортировать, имя_класс' (название пакета)), чтобы среда разработки Eclipse быстро добавила подходящую инструкцию import.

Помимо того, команда **Organize imports** (Организовать импорты) (комбинация клавиш **Ctrl+Shift+O** в операционной системе Windows или комбинация клавиш **Cmd+Shift+O** в операционной системе Mac) заставляет среду разработки Eclipse автоматически организовать инструкции **import**. Среда разработки Eclipse удаляет неиспользуемые инструкции **import** и добавляет новые инструкции для используемых пакетов, которые еще не были импортированы.

Если существует неопределенность в выборе пакета для некоторого класса в процессе автоматического импортирования пакетов, например, как в случае с классом **Log** платформы **Android**, среда разработки Eclipse предложит вам выбрать нужный пакет для импортирования.

Наконец, вы можете настроить среду разработки Eclipse таким образом, чтобы она автоматически организовывала инструкции **import** при каждом сохранении файла. Эта настройка может быть установлена как для всего рабочего пространства, так и для отдельного проекта. Настройка данной возможности на уровне отдельного проекта обеспечивает более высокую гибкость в тех случаях, когда вы работаете над несколькими проектами и не хотите вносить никаких изменений в некий код, даже если эти изменения являются улучшениями. Чтобы настроить данную возможность, выполните следующие шаги:

1. Щелкните правой кнопкой мыши по названию проекта и в появившемся контекстном меню выберите команду **Properties** (Свойства).
2. В появившемся диалоговом окне разверните элемент списка **Java Editor** (Редактор Java) и выберите раздел **Save Actions** (Действия при сохранении).
3. Установите флажок **Enable Project Specific Settings** (Включить настройки на уровне проекта), после этого — флажок **Perform the Selected Actions on Save** (Выполнять выбранные действия при сохранении), и, наконец, флажок **Organize Imports** (Организовать импорты).

ДОКУМЕНТИРОВАНИЕ КОДА. Польза от комментариев, регулярно встречающихся в коде, несомненна (при условии, что эти комментарии написаны толково). Комментарии в стиле **Javadoc** применяются во всплывающих диалоговых окнах автозаполнения кода и в других местах, делая их еще более полезными. Чтобы быстро добавить комментарий в стиле **Javadoc** к методу или к классу, просто нажмите комбинацию клавиш **Ctrl+Shift+J** (Windows) или **Cmd+Alt+J** (Mac). В качестве альтернативного варианта вы можете выбирать в контекстном меню команду **Source=>Generate Element Comment** (Исходный код=>Сгенерировать комментарий к элементу), чтобы автоматически добавлять определенные поля в **Javadoc**-комментарии, например, названия параметров и имя автора, тем самым значительно повысив скорость создания подобных комментариев.

ИСПОЛЬЗОВАНИЕ ФУНКЦИИ АВТОЗАПОЛНЕНИЯ. Автозаполнение — это потрясающая возможность, которая ускоряет процесс ввода текста. Если вы еще не видели соответствующее всплывающее диалоговое окно данной возможности или если это окно закрылось, вы можете отобразить его на экране, нажав комбинацию клавиш **Ctrl+Пробел**.

Автозаполнение помогает не только сэкономить время в процессе ввода текста, но также позволяет освежить в вашей памяти сведения о некоторых методах — или найти новый метод. Вы можете просмотреть список всех методов отдельного класса и даже увидеть связанные с ними **Javadoc**-комментарии. Вы можете легко найти статические методы, используя имя класса или имя переменной экземпляра. Для этого введите нужное имя, поставьте точку (и, возможно, нажмите комбинацию клавиш **Ctrl+Пробел**) и прокрутите список всех доступных имен. После этого вы можете начать ввод первой части имени, чтобы отфильтровать результаты.

ЭФФЕКТИВНОЕ РЕДАКТИРОВАНИЕ КОДА. В некоторых случаях вам может показаться, что окно редактора имеет слишком маленькие размеры, особенно, когда вокруг него расположено множество дополнительных маленьких панелей и вкладок.

Попробуйте сделать следующее: дважды щелкните по вкладке исходного файла, который вы хотите изменить. Бум! Теперь окно редактора занимает почти все пространство окна среды разработки Eclipse! Снова дважды щелкните по вкладке, и окно редактора примет первоначальные размеры.

Вам никогда не хотелось видеть одновременно код из двух исходных файлов? Ну, это возможно! Просто щелкните по вкладке исходного файла и, удерживая кнопку мыши, перетащите его к любому краю окна редактора. Вы должны увидеть темный контур, обозначающий новое местоположение окна редактора — либо бок о бок с другим файлом, либо выше или ниже другого файла. Когда вы отпустите кнопку мыши, будет создано второе окно редактора, в которое можно также перетаскивать другие вкладки файлов.

Вам никогда не хотелось видеть одновременно два разных фрагмента в одном исходном файле? Это возможно! Щелкните правой кнопкой мыши по вкладке нужного файла и в появившемся контекстном меню выберите команду **New Editor** (Новый редактор). В результате появится вторая вкладка для одного и того же файла. Теперь, воспользовавшись предыдущим советом, вы можете иметь два разных представления одного и того же файла.

Вам никогда не казалось, что у вас открыто слишком много вкладок для файлов, которые вы уже не редактируете? Мне казалось! Существует ряд решений для данной проблемы. Во-первых, вы можете щелкнуть правой кнопкой мыши по вкладке файла и выбрать в появившемся контекстном меню команду **Close Others** (Закрыть остальные), чтобы закрыть все остальные открытые файлы, кроме выбранного файла. Во-вторых, вы можете быстро закрыть конкретные вкладки, щелкнув средней кнопкой мыши по каждой вкладке. (Этот способ работает даже на компьютерах Mac, оснащенных мышью, позволяющей совершать щелчки средней кнопкой, например, мышью с колесиком прокрутки). Наконец, вы можете использовать настройку среды разработки Eclipse, позволяющую ограничивать количество открытых редакторов:

1. Откройте диалоговое окно **Preferences** (Настройки) среды разработки Eclipse.
2. В открытом диалоговом окне **Preferences** (Настройки) разверните элемент списка **General** (Общие), выберите раздел **Editors** (Редакторы) и установите флажок **Close Editors Automatically** (Автоматически закрывать редакторы).
3. Измените значение параметра **Number of Opened Editors Before Closing** (Количество открытых редакторов перед закрытием).

На мой взгляд, значение 8 — это оптимальное значение для параметра **Number of Opened Editors Before Closing** (Количество открытых редакторов перед закрытием), позволяющее избежать нагромождения вкладок редакторов и при этом иметь достаточное число открытых редакторов для выполнения работы и для просмотра кода.

Также стоит отметить, что, если вы установите переключатель **When All Editors Are Dirty or Pinned** (Когда все редакторы содержат несохраненный код или закреплены) в положение **Open New Editor** (Открывать новый редактор), будет открыто большее число окон редакторов в том случае, когда вы активно редактируете файлы, количество которых превышает значение, указанное для параметра **Number of Opened Editors Before Closing** (Количество открытых редакторов перед закрытием). Таким образом, эта настройка не оказывает влияния на производительность, когда вы одновременно редактируете большое количество файлов, но помогает поддерживать порядок в среде разработки в процессе выполнения большинства повседневных задач.

ПЕРЕИМЕНОВАНИЕ ПРАКТИЧЕСКИ ВСЕГО. Инструмент **Rename** (Переименовать) среды разработки Eclipse — очень мощный. Он может быть использован для переименования переменных, методов, классов и много другого. В большинстве случаев вы можете просто щелкнуть правой кнопкой мыши по элементу, который вы хотите переименовать, и выбрать в появившемся контекстном меню команду **Refactor=>Rename** (Рефакторинг=>Переименовать). В качестве альтернативы после того, как вы нашли нужный элемент, можно нажать комбинацию клавиш **ALT+Shift+R**, чтобы

запустить процесс переименования. Если вы переименовываете класс верхнего уровня в файле, имя файла также должно быть изменено. Если файл находится под управлением системы контроля исходного кода, среда разработки Eclipse обычно выполняет все необходимые действия, связанные с его переименованием.

Если среда разработки Eclipse может определить, что переименовываемый элемент связан с другим элементом, имеющим точно такое же имя, все вхождения данного имени также будут переименованы. В некоторых случаях это означает, что новое имя будет указано даже в комментариях. Довольно удобно!

ФОРМАТИРОВАНИЕ КОДА. Среда разработки Eclipse имеет встроенный механизм форматирования Java-кода.

Форматирование кода при помощи данного механизма полезно для поддержания согласованности стиля кода, для применения нового стиля к старому коду или для приведения стилей в соответствие с требованиями различных клиентов или задач (например, для книги или статьи).

Чтобы быстро отформатировать небольшой фрагмент кода, выделите данный код и нажмите комбинацию клавиш Ctrl+Shift+F (Windows) или Cmd+Shift+F (Mac). Код будет отформатирован в соответствии с текущими настройками. Если никакой фрагмент кода не выделен, будет отформатирован весь код в файле. В некоторых случаях вам необходимо выделить больший фрагмент кода, например целый метод, чтобы сохранить уровни отступов и расположение фигурных скобок.

Настройки форматирования среды разработки Eclipse можно найти в диалоговом окне Properties (Свойства), в разделе Formatter (Форматтер) группы Java Code Style (Стиль кода Java). Вы можете установить эти настройки на уровне проекта или на уровне рабочего пространства. Вы можете применить и изменить десятки правил, чтобы код соответствовал вашему собственному стилю.

ОРГАНИЗАЦИЯ КОДА. В некоторых случаях форматирования кода недостаточно, чтобы сделать его понятным и удобочитаемым. В процессе разработки сложного действия появиться множество вложенных классов и методов, разбросанных по всему файлу. В этом случае вам поможет одна из хитростей разработки Eclipse: открыв нужный файл, убедитесь, что на экране также отображается панель Outline (Обзор). Просто перетаскивайте методы и классы на панели Outline (Обзор), чтобы разместить их в подходящем логическом порядке. У вас есть метод, который вызывается лишь из определенного класса, но при этом доступен всем другим классам. Просто перетащите этот метод в данный класс.

Описанный подход работает практически со всеми элементами, доступными на панели Outline (Обзор), включая методы и переменные.

РЕФАКТОРИНГ С УДОВОЛЬСТВИЕМ. Вы никогда не замечали за собой, что написали много повторяющихся фрагментов кода, который выглядит, например, наподобие следующего кода?

```
TextView nameCol = new TextView(this);
nameCol.setTextColor(getResources().getColor(R.color.title_color));
nameCol.setTextSize(getResources().getDimension(R.dimen.help_text_size));
nameCol.setText(scoreUserName);
table.addView(nameCol);
```

Этот код устанавливает цвет текста, размер текста и сам текст. Если вы написали два или более фрагментов подобного кода, вашему коду может пойти на пользу рефакторинг. Среда разработки Eclipse предоставляет два крайне полезных инструмента — команды Extract Local Variable (Извлечь локальную переменную) и Extract Method (Извлечь метод) — для ускорения данного процесса и его максимального упрощения.

Чтобы воспользоваться командой Extract Local Variable (Извлечь локальную переменную), выполните следующие шаги:

1. Выделите выражение `getResources().getColor(R.color.title_color)`.
2. Щелкните правой кнопкой мыши и в появившемся контекстном меню выберите команду `Refactor=>Extract Local Variable` (Рефакторинг=>Извлечь локальную переменную) (или нажмите комбинацию клавиш `Ctrl+Alt+L`).
3. В открывшемся диалоговом окне введите имя переменной и, не сбрасывая флажок `Replace All Occurrences` (Заменить все вхождения), нажмите на кнопку `ОК`, чтобы понаблюдать за волшебством.
4. Повторите шаги 1-3 для выражения, устанавливающего размер текста. Полученным результатом должен выглядеть следующий код:

```
int textColor =
getResources().getColor(R.color.title_color);
float textSize =
getResources().getDimension(R.dimen.help_text_size);
TextView nameCol = new TextView(this);
nameCol.setTextSize(textSize);
nameCol.setText(scoreUserName);
nameCol.setTextColor(textColor);
table.addView(nameCol);
```

Изменения также отразились и в оставшихся пяти строках кода. Как это удобно, не правда ли? Теперь настало время испытать в действии второй инструмент. Выполните следующие шаги, чтобы воспользоваться командой `Extract Method` (Извлечь метод):

1. Выделите все пять строк первого фрагмента кода.
2. Щелкните правой кнопкой мыши и в появившемся контекстном меню выберите команду `Refactor=>Extract Method` (Рефакторинг => Извлечь метод) (или нажмите комбинацию клавиш `Ctrl+Alt+M`).
3. В открывшемся диалоговом окне присвойте новому методу название и при желании измените имена переменных. (При желании вы также можете перемещать переменные вверх или вниз по списку.) Затем нажмите на кнопку `ОК` и понаблюдайте за волшебством. По умолчанию, новый метод будет расположен сразу за вашим текущим методом.

Если другие фрагменты в коде являются полностью идентичными выделенному фрагменту, то есть выражения в других фрагментах кода идут в таком же порядке, все типы совпадают и т. д., они также будут заменены вызовом данного нового метода! Вы можете увидеть количество дополнительных вхождений в диалоговом окне команды `Extract Method` (Извлечь метод). Если это количество отличается от ваших ожиданий, убедитесь, что все фрагменты кода в точности соответствуют одному и тому же шаблону.

Теперь ваш код будет выглядеть примерно так:

```
addTextToRowWithValues(newRow, scoreUserName, textColor,
textSize);
```

Работать с этим кодом проще по сравнению с первоначальным кодом, и этот новый код был создан практически без нажатий клавиш! Если до реализации рефакторинга в вашем коде было десять подобных фрагментов, благодаря использованию полезной команды среды разработки `Eclipse` вы сэкономите уйму времени.

РЕШЕНИЕ ЗАГАДОЧНЫХ ПРОБЛЕМ КОМПИЛЯЦИИ. Порой вы можете столкнуться с ситуацией, когда среда разработки `Eclipse` обнаружит ошибки компиляции там, где еще несколько мгновений назад их попросту не было. В подобной ситуации можно опробовать несколько несложных хитростей среды разработки `Eclipse`.

Во-первых, попробуйте обновить проект: просто щелкните правой кнопкой мыши по названию проекта и выберите команду `Refresh` (Обновить), или нажмите клавишу `F5`.

Если это не помогло, попробуйте удалить файл R.java, который находится в каталоге gen под названием конкретного компилируемого пакета. (Не беспокойтесь: этот файл создается при каждой компиляции проекта.)

Если напротив пункта меню Compile Automatically (Компилировать автоматически) установлен флажок, этот файл будет пересоздан автоматически. В противном случае, вам понадобится скомпилировать проект вручную.

Наконец, вы можете попробовать очистить проект. Для этого выберите команду меню Project => Clean (Проект => Очистить) и в открывшемся диалоговом окне Clean (Очистка) выберите проекты, которые вы хотите очистить. Среда разработки Eclipse удалит все временные файлы и затем перекомпилирует проект(ы).

СОЗДАНИЕ ПОЛЬЗОВАТЕЛЬСКИХ ФИЛЬТРОВ ДЛЯ ЖУРНАЛИРУЕМОЙ ИНФОРМАЦИИ. Каждая инструкция журналирования в Android-приложении имеет тег. Эти теги могут быть использованы вместе с фильтрами, определяемыми на панели LogCat. Чтобы добавить новый фильтр, щелкните по значку «плюс» зеленого цвета на панели LogCat.

Укажите название для фильтра — возможно, используя имя тега — и введите тег, который вы хотите использовать. Теперь на панели LogCat появится другая вкладка, на которой будут отображаться сообщения, содержащие указанный тег. Помимо этого, вы можете создавать фильтры, которые отображают элементы в зависимости от степени серьезности сообщения.

Соглашение по именованию тегов, применяемое при разработке Android-приложений, подразумевает использование в качестве имени тега названия класса. Вы часто видели это в коде, прилагаемом к данной книге. Обратите внимание, что в каждом классе мы создаем константу с одним и тем же именем, чтобы упростить вызов инструкций журналирования. Вот пример:

```
public static final String  
DEBUG_TAG  
= "MyClassName";
```

Тем не менее, данное соглашение не обязательно. Теги могут быть организованы по отдельным задачам, которые выполняются в различных действиях, или вы можете использовать любой другой разумный способ организации тегов, соответствующий вашим требованиям.

ПЕРЕМЕЩЕНИЕ ВКЛАДOK. Среда разработки Eclipse предоставляет ряд довольно неплохих компоновок панелей, относящихся к стандартным перспективам. Тем не менее у каждого из нас есть свой стиль работы, а для разработки Android-приложений доступные перспективы имеют неудовлетворительные стандартные компоновки.

Например, вкладка Properties (Свойства) обычно находится в нижней части экрана.

Для написания кода такое решение работает неплохо, поскольку эта вкладка занимает всего лишь несколько строк по высоте. Однако при разработке макетов для Android-приложений такой вариант размещения не совсем удобен.

К счастью, в среде разработки Eclipse эту проблему легко исправить: просто перетащите вкладку, щелкнув мышью по самой вкладке (ее заголовку), в новое местоположение, например, в вертикальную группу вкладок, расположенную в правой части окна среды разработки Eclipse. Это позволит получить необходимое вертикальное пространство для отображения десятков свойств, которые часто появляются на этой панели.

Вы можете поэкспериментировать, чтобы найти оптимальную для вас компоновку вкладок. Каждая перспектива также имеет свою собственную компоновку, и перспективы могут быть ориентированы на выполнение определенных задач. Если вы окончательно

испортили перспективу или просто хотите начать все сначала, достаточно выбрать команду Window => Reset Perspective (Окно => Восстановить перспективу).

ИНТЕГРАЦИЯ С СИСТЕМОЙ КОНТРОЛЯ ИСХОДНОГО КОДА. Среда разработки Eclipse имеет возможность интегрироваться со многими системами контроля исходного кода путем использования дополнений. Благодаря этой интеграции среда разработки Eclipse может самостоятельно получать последнюю версию файла, позволяя вносить изменения в файл, когда вы приступите к редактированию файла, выгружать изменения в систему контроля исходного кода, обновлять содержимое файла, отображать статус файла и выполнять множество других задач, которые зависят от реализуемой поддержки функциональности конкретным дополнением. Наиболее распространенными являются дополнения для систем контроля исходного кода CVS, Subversion, Perforce, git и многих других.

Вообще говоря, не все файлы подходят для помещения в систему исходного кода. Для проектов Android-приложений любые файлы находятся в каталогах bin и gen, не должны помещаться в систему контроля исходного кода.

Чтобы исключить добавление этих файлов в систему контроля исходного кода из среды разработки Eclipse, откройте диалоговое окно Preferences (Настройки), разверните элемент списка Team (Командная работа; и выберите раздел Ignored Resources (Игнорируемые ресурсы). Поочередно добавьте в список Ignore Patterns (Игнорируемые шаблоны) маски файлов *.apk, *.ar и *.dex, используя кнопку Add Pattern (Добавить шаблон).

2.6 Документация по платформе Android

На сегодняшний день в распоряжении разработчиков Android находится около десятка средств разработки, облегчающих процесс создания качественных приложений.

Приоритетная задача на ранней стадии освоения Android — получить представление о доступных инструментах и о том, как они могут использоваться. Когда вы сталкиваетесь с какой-либо проблемой, у вас уже будет некоторое представление о том, какие инструменты могут вам помочь в ее решении. Средства разработки Android находятся в подпапке /tools папки установки Android SDK. В этом случае вы узнаете о большинстве самых важных инструментов, доступных для свободного использования. Эта информация поможет вам разрабатывать приложения Android быстрее и легче [2].

Хотя сама по себе документация Android — не инструмент, это ресурс первостепенной важности для каждого разработчика Android. Ее HTML-вариант можно найти в подпапке /docs Android SDK, обращайтесь к нему при возникновении каких-либо затруднений.

Последнюю редакцию справочной документации можно найти на веб-сайте разработчиков Android: [http:// developer.android.com](http://developer.android.com).

Документация Android состоит из шести разделов:

- **SDK** важная информация о SDK, установленном на вашем компьютере. Здесь имеются сведения о версиях и описание всех известных проблем каждой из них. Эта информация может вам пригодиться, если онлайн-справка была обновлена для новой версии, но вы хотите разрабатывать приложение на более старой версии SDK.
- **Dev Guide (Руководство разработчика)** — эта вкладка содержит руководство разработчика Android, включая ответы на часто задаваемые вопросы, а также пошаговые примеры и глоссарий терминов Android для начинающих.
- **Reference (Справка)** — на этой вкладке представлен поиск по списку пакетов и классов всех программных интерфейсов Android.
- **Blog (Блог)** — официальный блог разработчиков Android. Здесь вы можете найти последние новости и объявления. Это прекрасное место для поиска

примеров с практическими рекомендациями, обучения оптимизации приложений Android и получения информации о новых версиях SDK и конкурсах для разработчиков.

- **Videos (Видео)** — обучающий видеоматериал, который можно посмотреть онлайн. Здесь вы найдете видеоролики о платформе Android с советами по разработке, а также записи конференций Google на английском языке.
- **Resources (Ресурсы)**. Раздел Community (Сообщество) этой вкладки — 1 ваша отправная точка для обсуждения вопросов с другими разработчиками Android.

Кроме того, вы можете вступить в одну или несколько тематических групп Google по вашему желанию.

Рекомендуем вам провести первое ознакомление с документацией Android SDK сейчас.

Но перед тем, как перейти к онлайн-документации, ознакомьтесь с ее локальным вариантом.

2.7 Отладка приложений с помощью DDMS

Мониторинговый сервис отладки Dalvik (DDMS, Dalvik Debug Monitor Service) — утилита отладки, интегрируемая в Eclipse через перспективу DDMS. Перспектива DDMS предоставляет несколько полезных возможностей взаимодействия с мобильными телефонами и эмуляторами.

Возможности DDMS можно грубо разделить на пять категорий:

- управление задачами;
- управление файлами;
- взаимодействие с эмулятором;
- журналирование;
- создание скриншотов.

DDMS и перспектива DDMS — основные инструменты отладки. Рассмотрим их возможности более детально. Инструмент DDMS может быть запущен отдельно от Eclipse. Его исполняемый файл находится в папке Android SDK /tools.

УПРАВЛЕНИЕ ЗАДАЧАМИ. На панели Devices (Устройства) перспективы DDMS приводится список (Мониторов и мобильных телефонов, подключенных в настоящий момент. Вы можете выбрать отдельные экземпляры, просмотреть текущие процессы и потоки. Чтобы просмотреть поток, щелкните мышью по интересующему процессу устройства, например, com.androidbook.droid1, затем на кнопке обновления потока. Вы можете также запросить сбор лишних данных на процессе и затем посмотреть обновления стека, щелкнув по кнопке с изображением зеленого цилиндра. Наконец, можно остановить процесс, щелкнув по кнопке с изображением знака «stop».

ОТЛАДКА В ПЕРСПЕКТИВЕ DDMS. В пределах перспективы DDMS вы можете выбрать определенный процесс на эмуляторе или мобильном телефоне и затем нажатием на кнопку отладки с изображением зеленого жука присоединить отладчик к этому процессу. Для использования этой возможности на вашем рабочем пространстве Eclipse должен быть исходный код. Это относится только к Eclipse, в автономной версии DDMS этого делать нельзя.

ОБЗОР ФАЙЛОВОЙ СИСТЕМЫ ANDROID. Для просмотра файлов и папок в эмуляторе или на устройстве вы можете воспользоваться панелью File Explorer (Проводник файлов) в перспективе DDMS. Вы можете копировать файлы между файловой системой Android и вашим компьютером с помощью кнопок перемещения.

Удалять файлы и папки вы можете кнопкой с изображением знака «минус» или клавишей Delete. При этом не будет выведено сообщение с предложением о подтверждении удаления и отмены этого действия.

ВЗАИМОДЕЙСТВИЕ С ЭМУЛЯТОРАМИ. DDMS может инициировать множество событий, таких как голосовые вызовы, отправка и прием SMS и запрос геолокационных координат на определенных экземплярах эмулятора.

Эта возможность доступна на панели Emulator Control (Управление эмулятором) в DDMS. Все события могут быть инициированы только в одну сторону — из DDMS к устройству. Эти возможности доступны только для эмуляторов. В случае с мобильными телефонами вы должны использовать реальные вызовы и реальные сообщения.

ИМИТАЦИЯ ВХОДЯЩИХ ГОЛОСОВЫХ ВЫЗОВОВ НА ЭМУЛЯТОРЕ. Входящие голосовые вызовы инициируются на панели Emulator Control (Управление эмулятором) в DDMS. Так как это не настоящий звонок, никакие данные (звукового или другого типа) между вызывающей стороной и адресатом не передаются.

ИМИТАЦИЯ ВХОДЯЩИХ SMS НА ЭМУЛЯТОРЕ. На панели Emulator Control (Управление эмулятором) в DDMS вы можете также имитировать отставку SMS к эмулятору. Отправка SMS осуществляется по такому же принципу, что и голосовой вызов.

СОЗДАНИЕ СКРИНШОТОВ С ЭМУЛЯТОРА ЛИБИ МОБИЛЬНОГО ТЕЛЕФОНА. Одна из возможностей, которая может пригодиться при отладке приложения как на мобильном телефоне, так и на эмуляторе, — это создание скриншота текущего экрана.

С помощью кнопки Screen Capture (Создать скриншот) в DDMS вы можете сделать скриншот. Возможность создания скриншота особенно полезна при работе с реальными мобильными телефонами. Чтобы создать скриншот, выполните следующие шаги:

1. Откройте DDMS, выберите устройство (или эмулятор), с которого вы хотите создать скриншот.
2. На выбранном устройстве или эмуляторе выведите изображение экрана, скриншот которого вы хотите создать. Настройте внешний вид, если это необходимо.
3. Нажмите на кнопку с изображением разноцветного квадрата. Откроется диалоговое окно создания скриншота.
4. В этом диалоговом окне нажмите на кнопку Save (Сохранить), чтобы сохранить, скриншот на жесткий диск.

ПРОСМОТР ЖУРНАЛА. Встроенная в DDMS утилита журналирования LogCat позволяет просматривать консоль журналирования Android. Возможно, вы уже обращали внимание на панель LogCat и диагностическую информацию.

В Eclipse вы можете фильтровать данные журнала по необходимым критериям.

С помощью тегов вы можете создавать собственные фильтры.

РАБОТА С ЭМУЛЯТОРОМ ANDROID. Эмулятор Android — один из самых мощных инструментов, который есть в распоряжении разработчика. Поэтому каждый разработчик должен уметь им пользоваться и знать его возможности. Эмулятор Android интегрируется в Eclipse IDE с помощью плагина ADT.

- Эмулятор Android — очень удобный инструмент, но у него есть некоторые ограничения:
- Эмулятор — не полноценное устройство. Он моделирует лишь общее «поведение» мобильного телефона, поэтому специфические аппаратные функции не могут быть доступны. Такие данные, как геолокационная информация со спутника, информация об уровне заряда батареи питания и качестве связи, симулируются компьютером. Дополнительные возможности устройства, такие как управление камерой, ограничены.
- Телефонные звонки имитируются, поэтому они не могут быть приняты или отправлены в настоящей сети. SMS также имитируются и не пересылаются по настоящей сети.
- Поддержка USB и Bluetooth отсутствует. Помните, эмулятор Android не заменяет при тестировании реальное устройство.

Как разработчик, вы можете вводить данные в эмуляторе несколькими способами:

- Компьютерная мышь позволяет щелкать по элементам управления, прокручивать и перетаскивать их (например, ползунковый регулятор урони! громкости).
- Мышь позволяет также имитировать нажатие отдельных клавиш на программной или реальной клавиатуре.
- Клавиатура используется для ввода текста в элементах управления а также для управления определенными состояниями эмулятора.

Помимо этих в вашем распоряжении есть еще целый набор доступных команд. Полный их список вы можете найти в официальной документации к эмулятору. Онлайн-версия доступна по адресу: developer.android.com/guide/developing/tools_emulator.html.

2.8 Обзор системы Android

Если у вас небольшой опыт работы с устройствами Android, то сейчас подходящее время для изучения системы Android с точки зрения пользователя. В таблице 2.3 представлены некоторые встроенные возможности Android.

Таблица 2.3. Типы системных экранов Android и их свойства.

Тип экрана	Описание
Экран по умолчанию	Это основной экран. На нем отображаются основные элементы интерфейса и пункты меню
Экран набора номера	Это встроенное приложение для совершения телефонных звонков. Помните, что эмулятор обладает ограниченным набором возможностей
Экран обмена текстовыми сообщениями	Встроенное приложения для отправки и получения SMS. Помните, что возможности по обмену сообщениями ограничены
Веб-браузер	Встроенный веб-браузер. Обратите внимание, что эмулятор использует реальное интернет-соединение на вашем компьютере
Список контактов	Вся информация о контактах
Каталог приложений	Здесь отображаются все установленные приложения. Чтобы перейти к нему, вернитесь к экрану по умолчанию и переместите серый ползунок вверх
Экран настройки параметров	Это встроенное приложение для настройки широкого диапазона параметров эмулятора, таких как управление приложениями, настройки звука и экрана, а также региональные и языковые настройки
Инструменты разработки	Встроенное приложение для настройки параметров инструментов разработки

Использование обложек (skins) в эмуляторе. Возможности эмулятора, такие как размер экрана, его ориентация, а также то, имеет ли эмулятор физическую или виртуальную клавиатуру, определяются обложкой эмулятора. Android SDK поддерживает множество различных обложек, которые имитируют различные разрешения телефонных экранов (по умолчанию — HVGA). Совместимость обложки зависит от версии целевой платформы. Подбор подходящей обложки — часть процесса конфигурации AVD.

Использование образа SD-карты в эмуляторе. Для сохранения данных в эмуляторе необходим сконфигурированный образ SD-карты. Хорошо сконфигурированный образ SD-карты пригодится вам, например, для хранения таких медиа-файлов, как графические данные с камеры и звуковые файлы. Создавать образ SD-

карты удобнее всего одновременно с созданием AVD. При этом образ SD-карты должен быть не меньше 9 мегабайт.

Несмотря на то, что все наиболее важные инструменты уже были рассмотрены, существует еще несколько специализированных утилит, которые входят в состав Android SDK.

- Android Hierarchy Viewer (Просмотр иерархии Android) позволяет разработчикам проверять компоненты пользовательского интерфейса, такие как View Properties (Свойства представления), во время выполнения приложения.
- Draw 9-Patch tool (Инструмент растяжения NinePatch) — служит для разработки растягиваемых PNG-файлов.
- AIDL Compiler (Компилятор AIDL) — служит для создания удаленных интерфейсов для упрощения взаимодействия между процессами (IPC) на платформе Android.
- Утилита командной строки mksdcard позволяет разработчикам создавать автономные образы SD-карт для использования в AVD и эмуляторе.

2.9 Разработка приложений ANDROID вне ECLIPSE

Для разработки приложений Android предпочтительнее пользоваться средой разработки Eclipse, но без нее можно обойтись. Плагин ADT для Eclipse предоставляет удобную возможность для выполнения многих задач разработки: создания, отладки, сжатия и лицензирования приложений Android.

Разработчики, не использующие Eclipse, или те, кому нужны более мощные средства отладки, недоступные в плагине ADT, могут использовать эти базовые инструменты непосредственно в командной строке. Все инструменты, перечисленные ниже, находятся в папке /tools Android SDK.

- android — служит для создания файлов проекта Android и управления AVD.
- aapt [Android Asset Packaging Tool, инструмент сжатия данных Android) — служит для сжатия файлов проекта Android в APK-файлы для последующей установки на эмуляторе или мобильном телефоне.
- ddms — имеет собственный пользовательский интерфейс, который напоминает перспективу DDMS в Eclipse.
- adb (Android Debug Bridge) — имеет интерфейс командной строки для взаимодействия с эмулятором или устройством.

В состав Android SDK входит множество мощных инструментов для решения общих задач:

- При разработке приложений Android. Документация Android — главный источник информации для каждого разработчика средство отладки DDMS, интегрированное в среду разработки Eclipse. Используется для наблюдения за работой приложения на эмуляторе или физическом устройстве.
- Эмулятор Android может использоваться для выполнения и отладки приложений Android почти без необходимости использования реального устройства.
- Существует также множество других инструментов для взаимодействия с мобильными телефонами и эмуляторами в интерфейсе командной строки, а также специальные утилиты для разработки пользовательских интерфейсов Android и сжатия приложений.

3 Основы языка Java

3.1 Принципы ООП

Основные принципы ООП:

- **Инкапсуляция** – сокрытие реализации класса и отделение его внутреннего представления от внешнего.
- **Наследование** – отношение между классами, при котором один класс использует структуру или поведение другого (одионое наследование) или других (множественное наследование) классов.
- **Полиморфизм** – положение теории типов, согласно которому имена (например, переменных) могут обозначать объекты разных (но имеющих общего родителя) классов.

Достоинства ООП:

- **Упрощение разработки.** Включает в себя **разделение функциональности** (чем больше и сложнее программная система, тем важнее становится разбить ее на небольшие, четко очерченные части. Чтобы справиться со сложностью, необходимо абстрагироваться от мелких деталей. Для этой цели классы представляют собой весьма удобный инструмент), **локализацию кода** (данные и операции над ними вместе образуют определенную сущность, и они не разносятся по всей программе, как это нередко бывает в случае процедурного программирования, а описываются вместе. Локализация кода и данных улучшает наглядность и удобство сопровождения программного обеспечения), **инкапсуляцию** (позволяет привести свойство модульности, что облегчает распараллеливание выполнения задачи между несколькими исполнителями и обновление версий отдельных компонент).
- **Возможность создания расширяемых систем.** Включает в себя **обработку разнородных структур данных** (программы могут работать, не различая вида объектов, что существенно упрощает код. Новые виды могут быть добавлены в любой момент. Это полиморфизм), **изменение поведения на этапе выполнения** (на этапе исполнения один объект может быть заменен другим, что позволяет легко без изменения кода адаптировать алгоритм, в зависимости от того, какой используется объект. Это полиморфизм), **работу с наследниками** (алгоритмы можно обобщить настолько, что они уже смогут работать более чем с одним видом объектов. Это полиморфизм).
- **Легкость модернизации с сохранением совместимости.**

Недостатки ООП:

- **Неэффективность на этапе выполнения.** Существует фактор, который влияет на время выполнения: это инкапсуляция данных. Рекомендуется не предоставлять прямой доступ к полям класса, а выполнять каждую операцию над данными через методы. Такая схема приводит к необходимости выполнения процедурного вызова при каждом доступе к данным.
- **Неэффективность в смысле распределения памяти.** Динамическое связывание и проверка типа на этапе выполнения требуют по ходу работы информации о типе объекта. Такая информация хранится в дескрипторе типа, и он выделяется один на класс. Каждый объект имеет невидимый указатель на дескриптор типа для своего класса. Таким образом, в объектно-ориентированных программах требуемая дополнительная память выражается в одном указателе для объекта и в одном дескрипторе типа для класса.
- **Излишняя избыточность.** В библиотечном классе часто содержится больше методов, чем это реально необходимо. А поскольку лишние методы не

могут быть удалены, то они становятся мертвым грузом. Это не воздействует на время выполнения, но влияет на возрастание размера кода.

- **Психологическая сложность проектирования.**
- **Техническая сложность проектирования и документирования.** Документирование классов - задача более трудная, чем это было в случае процедур и модулей. Поскольку любой метод может быть переопределен, в документации должно говориться не только о том, что делает данный метод, но также и о том, в каком контексте он вызывается. Для абстрактных методов, которые пусты, в документации должно даже говориться о том, для каких целей предполагается использовать переопределяемый метод.

3.2 Классы и объекты

Всё сказанное далее будет относиться к JDK 1.4, пока не будет специально оговорено, что рассматривается JDK 1.5 [3].

Программы на языке Java создаются из классов (classes). Используя объявление класса, можно построить любое число объектов (objects), или, как еще говорят, экземпляров класса. В ООП определены следующие ключевые свойства объектов:

- **Поведение объекта** – что с ним можно делать и какие методы к нему можно применять.
- **Состояние объекта** – как этот объект реагирует на применение методов.
- **Сущность** или **уникальность объекта** – чем данный объект отличается от других, характеризующихся таким же поведением и состоянием.

Все объекты, являющиеся экземплярами одного и того же класса, ведут себя одинаково и имеют одинаковый набор свойств. Поведение объекта определяется методами, которые можно вызвать. Каждый объект сохраняет информацию о своем состоянии. Со временем состояние объекта может измениться, но только в результате вызовов методов. Если состояние объекта изменилось вследствие иных причин, значит, принцип инкапсуляции не соблюден. Состояние объекта не полностью описывает его, так как каждый объект имеет свою собственную сущность. Индивидуальные объекты, представляющие собой экземпляры класса, всегда отличаются своей сущностью и обычно отличаются своим состоянием.

Все сущности в Java являются объектами, классами или интерфейсами. Ниже приведен пример простого класса, названного Body.

Пример 1.

```
class Body {
    public long idNum;
    public String name;
    public Body orbits;

    public static long nextId = 0;
}
```

В объявлении класса содержится ключевое слово class, за которым следует наименование класса и перечень его членов, заключенных в фигурные скобки. Объявление класса создает новый тип (type), так что ссылки на объекты этого типа могут выглядеть так:

Body mercury;

mercury – это переменная, способная хранить ссылку на объект типа Body. Выражение не создает объект, а только описывает ссылку, которая способна указывать на объект класса. На протяжении своего существования переменная *mercury* может ссылаться на множество различных объектов типа Body. Все эти объекты должны быть явным образом созданы.

Класс может содержать члены (members) трех основных категорий:

- **Поля (fields)** – переменные, относящиеся к классу и его объектам и в совокупности определяющие состояние класса или конкретного объекта.
- **Методы (methods)** – именованные фрагменты исполняемого кода класса, обуславливающие особенности поведения объектов класса.
- **Вложенные классы и вложенные интерфейсы** – объявления классов или интерфейсов, размещенные в контексте объявлений других классов или интерфейсов.

В объявлении класса может употребляться ряд служебных слов-модификаторов, придающих классу дополнительные свойства.

- **public.** Модификатор public помечает класс признаком общедоступности. Он означает, что в любом коде позволено объявлять ссылки на объекты класса и обращаться к его членам, отмеченным как public. Если модификатор public не задан, класс будет доступен только в контексте пакета, которому принадлежит.
- **abstract.** Класс, обозначенный модификатором abstract, трактуется как неполный, другими словами, создавать экземпляры такого класса запрещено. Подобное свойство класса обычно обусловлено наличием в его объявлении абстрактных методов (снабженных тем же модификатором abstract), которые, как предполагается по определению, должны быть реализованы в производных классах.
- **final.** Класс, определенный как final, не допускает наследования.
- **strictfp (strict floating point).** Присутствие в объявлении класса модификатора strictfp означает, что операции с плавающей запятой, предусмотренные методами-членами класса, должны выполняться точно и единообразно всеми виртуальными машинами Java.

Очевидно, что в объявлении класса не могут использоваться одновременно модификаторы final и abstract. Объявление способно содержать несколько модификаторов, порядок их указания несущественен. Большинство инструментальных средств разработки Java выдвигается требование, чтобы объявление класса с модификатором public сохранялось в файле с тем же именем, которое присвоено классу, отсюда следует, что файл не может содержать более одного объявления класса, помеченного как public (количество не публичных классов может быть произвольным).

3.3 Пакеты

Пакет – это комплект программного обеспечения, который может распространяться независимо и применяться при разработке приложений в сочетании с другими пакетами. Членами пакетов являются взаимосвязанные классы, интерфейсы, вложенные пакеты, а также дополнительные файлы ресурсов (например, графические), адресуемые в коде классов. Полезность использования пакетов обусловлена несколькими причинами.

- **Пакеты позволяют группировать взаимосвязанные классы и интерфейсы в единое целое.** Например, библиотечные классы, предназначенные для решения задач статистического анализа, целесообразно объединить в пакете с именем наподобие stats.
- **Способствуют созданию пространств имён, позволяющих избежать конфликтов идентификаторов, относящихся к разным типам.** В интерфейсах и классах, принадлежащих одному пакету, для обозначения public-членов могут быть использованы некоторые популярные идентификаторы – list, component – которые обладают строго определённым смыслом в контексте текущего пакета, но способны вступать в конфликт с одноимёнными членами других пакетов.

- **Обеспечивают дополнительные средства защиты элементов кода.** Фрагменты кода внутри пакета могут взаимодействовать, используя права доступа, которыми не обладает любой внешний код.

Каждый исходный файл, в котором размещены объявления классов и интерфейсов, относящихся к пакету `attr`, должен содержать специальное объявление:

```
package attr;
```

Это объявление должно располагаться в начале текста файла, до объявления классов или интерфейсов. В пределах файла допускается задать только одно объявление `package`. Наименование любого типа, принадлежащего пакету, неявно снабжается префиксом имени этого пакета.

Если тип принудительно не объявлен как часть некоторого пакета, он располагается в анонимном (`unnamed`) или безымянном пакете. Способ оформления кода в виде анонимного пакета вполне приемлем, когда речь идёт об исполняемом приложении (или апплете), которое заведомо не предназначено для вызова из какой бы то ни было сторонней программы. Тексты классов, ориентированных на совместное использование, должны быть размещены в именованных пакетах.

Об имени типа, в которое включён префикс названия пакета, отделённый символом точки, говорят как о **полном имени типа** (например, полным именем класса `String` является `java.lang.String`). При написании кода, которому необходимо обращаться к членам определённого пакета, возможно использование двух подходов. Один из них состоит в употреблении полного имени нужного типа. Он удобен, если вам нужно только несколько элементов пакета, но если учесть, что имена пакетов бывают достаточно длинными, неоднократное задание полного имени типа может оказаться утомительным занятием. Другой подход связан с импортом пакета или отдельной его части. Если вам необходимы средства пакета `attr`, нужно поместить в верхней части текста с исходным кодом (после объявления `package`, если таковое имеется, но перед любыми другими строками) следующую инструкцию импорта:

```
import attr.*;
```

Теперь для ссылки на типы, принадлежащие пакету `attr`, можно использовать простые имена, например, `Attributed`. Команду импорта, в которой используется символ `*`, называют объявлением **импорта по требованию** (***import on demand***). Можно воспользоваться также инструкцией **импорта единственного типа** (***single type import***):

```
import attr.Attributed;
```

Процедура импорта кода, принадлежащего текущему пакету, выполняется неявно, поэтому любые типы пакета допускают непосредственное обращение из кода других типов, объявленных в том же пакете.

3.4 Понятие имени

Имена задаются посредством идентификаторов и указывают на компоненты программы (типы, переменные, поля, методы и т.д.). Задача управления именами компонентов программы решается посредством двух механизмов. Во-первых, единое пространство имён (`namespace`) делится на категории, отвечающие различным разновидностям компонентов. Во-вторых, имена, видимые в одной части программы, «скрываются» от других посредством соответствующих контекстов. Так, например, механизм деления пространства имён позволяет использовать один и тот же идентификатор для поля и метода класса, а средства контекстного сокрытия дают возможность пользоваться одним идентификатором для обозначения переменных-счётчиков всех циклов `for`. Существует шесть различных **пространств имён**:

- Пакеты
- Типы
- Поля
- Методы

- Локальные переменные и параметры
- Метки

Механизм разделения пространства имён обеспечивает гибкие возможности написания программ, но при неразумном использовании он способен доставить и серьёзные неприятности (см. пример 2) [3].

Пример 2 (душераздирающий, но синтаксически правильный код)

```
package Reuse;

class Reuse {
    Reuse Reuse (Reuse Reuse) {
        Reuse:
        for(;;) {
            if (Reuse.Reuse(Reuse) == Reuse)
                break Reuse;
        }
        return Reuse;
    }
}
```

Каждое объявление имени определяет контекст, в котором имя может использоваться. Например, контекстом параметра метода, служит блок тела этого метода; контекст локальной переменной определяется границами блока, в котором она определена; контекст переменной цикла, объявленной в секции инициализации заголовка цикла `for`, распространяется на блок тела цикла.

Имя нельзя использовать за пределами его контекста – например, один метод класса не способен обращаться к параметрам другого метода. Контексты, однако, могут быть вложенными, и код внутреннего контекста обладает правами доступа ко всем именам внешнего контекста. Например, разрешено обращаться из тела цикла `for` к локальным переменным, объявленным в пределах того же метода, где находится и цикл `for`.

Подразумевается, что имена, объявленные во внешних контекстах, могут быть скрыты именами, принадлежащими внутренним контекстам. Например, имена членов класса способны перекрываться именами локальных переменных, унаследованные члены класса – членами, объявленными в текущем классе.

Не разрешается перекрывать имена во вложенных контекстах внутри одного блока кода. Это означает, что локальная переменная в теле метода не может обладать тем же именем, что и параметр метода; переменную цикла `for` не разрешается обозначать именем, которое принадлежит локальной переменной или параметру; во вложенном блоке нельзя повторно использовать имена, объявленные во внешнем блоке.

```
{
    int a = 0;
    {
        int a = 2; // неверно, переменная уже объявлена
        // ...
    }
}
```

Впрочем, ничто не запрещает создавать в пределах блока различные (не вложенные) циклы `for` или применять не вложенные блоки с объявлениями одноимённых переменных.

Примеры именования

- Пакеты (`java.lang`, `com.magic.attr`)
- Типы (`Student`, `ArrayIndexOutOfBoundsException`, `Runnable`)
- Поля (`value`, `enabled`, `distanceFromShop`)
- Методы (`getValue`, `setValue`, `toString`, `length`)
- Поля - константы (`PI`, `SIZE_MIN`, `SIZE_MAX`)

Модуль компиляции хранится в *.java файле и является единичной порцией входных данных для компилятора. Состоит из:

- Объявления пакета (package MyPackage;)
- import-выражений (import java.lang.Double;
- import java.lang.*;)
- Объявлений верхнего уровня – классов, интерфейсов.

Переменные, объявленные в классе, называют **полями (fields)**. Примером поля может служить переменная name класса Body, рассмотренного выше. Объявление поля состоит из наименования типа, за которым следует идентификатор переменной и необязательная конструкция инициализации, позволяющая присвоить переменной некое исходное значение. Каждый объект класса Body обладает собственными копиями трех полей: типа long для хранения уникального номера, позволяющего различить объект среди ему подобных, типа String, содержащего ссылку на строку имени объекта, и типа Body, ссылающегося на другой объект того же типа, который представляет небесное светило, «центральное» по отношению к текущему. Объект (экземпляр) класса обладает «личными» копиями полей, т.е. собственным – в общем случае, уникальным – состоянием. Поля объекта принято называть **переменными экземпляра**. Например, изменение содержимого поля *orbits*, принадлежащего одному из объектов *Body*, не воздействует на одноименные поля других объектов того же типа.

В конструкции объявления поля разрешено использовать дополнительные модификаторы, задающие определенные свойства поля.

- **Модификаторы доступа** - будут рассмотрены далее.
- **static**
- **final**
- **transient** – относится к проблеме сериализации (представления объекта в виде последовательности байтов данных) и будет рассмотрен далее.
- **volatile** – связан с вопросами синхронизации потоков вычислений и управления памятью, будет рассмотрен далее.

В объявлении поля одновременно не могут использоваться модификаторы final и volatile.

Поле может быть инициализировано в конструкции объявления с помощью оператора присваивания и значения соответствующего типа. В примере класса Body поле nextID инициализируется нулем. В качестве выражения инициализации допускается применять не только константы, но и имена других полей, конструкции вызова методов или более сложные выражения, состоящие из всех названных элементов вместе. Единственное требование состоит в том, что тип выражения инициализации должен совпадать с типом поля; кроме того, если в выражении используется вызов метода, тот не должен генерировать объявляемые исключения, поскольку в данном случае их «некому» будет отловить и обработать. Ниже приведены примеры допустимых выражений инициализации.

```
double zero = 0.0;
double zeroCopy = zero;
double rootTwo = Math.sqrt(2);
```

Если поле явно не инициализировано, ему присваивается значение соответствующего типа, принятое по умолчанию:

Тип	Значение по умолчанию
boolean	false
char	'\u0000'
byte, short, int, long	0
float	+0.0f
double	+0.0
ссылка на объект	null

В соответствии с принятым соглашением `null` выражает тот факт, что объект еще не создан или создан неправильно.

Иногда требуется, чтобы существовала единственная копия поля, общая для всех объектов класса. С этой целью в объявление поля вводится модификатор `static` (подобные поля называют **статическими полями** или **переменными класса**). Статическое поле «обитает» в пределах класса в единственном экземпляре, независимо от того, сколько объектов класса было создано и создавались ли таковые вообще.

В примере 1 класса `Body` объявлено одно статическое поле, `nextID`, предназначенное для хранения очередного доступного для использования порядкового номера объекта. Далее мы увидим, что в каждом вновь создаваемом объекте класса `Body` переменной `idNum` будет присваиваться текущее содержимое `nextID` с последующим увеличением значения `nextID` на единицу. Для создания объектов `Body` и ведения учета их порядковых номеров вполне достаточно одного экземпляра переменной `nextID`.

В контексте «родного» класса на статическое поле можно ссылаться непосредственно, но если предпринимается попытка доступа к полю извне, идентификатору поля следует предпослать имя класса. Например, значение `nextID` может быть выведено на экран следующим образом:

```
System.out.println(Body.nextID);
```

Собственно говоря, для иллюстрации возможностей доступа к статическим полям вполне подходит и поле `out` класса `System`.

Для обращения к статическому члену класса разрешено также пользоваться ссылкой на объект этого класса:

```
System.out.println(mercury.nextID);
```

Такой возможностью, однако, не следует злоупотреблять, поскольку в подобном случае создается ложное впечатление, что переменная `nextID` есть частный атрибут объекта `mercury`, а не член класса `Body` в целом. При обращении к статическому члену посредством ссылки на объект компилятор определяет имя соответствующего класса, а значением ссылки как таковой может быть даже `null`.

Значение поля, снабженного **модификатором `final`**, после инициализации уже не может быть изменено, так как любая попытка присваивания переменной нового содержимого приводит к ошибке времени компиляции. Модификатор `final` применяется для обозначения постоянства некоторого свойства класса или объекта на протяжении всего его жизненного цикла.

Поля `final` исполняют функции именованных констант. В языке `Java` часто возникает необходимость в поле, являющимся единственным для всех объектов класса и значение этого поля не может быть изменено. Такое поле называется **константой класса** и объявляется с помощью ключевых слов `static final`.

Если поле, помеченное как `final`, лишено инициализатора, его принято называть **blank final**. Наличие возможности подобного объявления полезно в тех случаях, когда для инициализации поля простого выражения недостаточно. Поле `final` должно быть инициализировано только единожды в ходе инициализации класса (в случае, если поле `static`) либо в процессе конструирования объекта класса (если поле не статическое). Компилятор проверяет, выполнена ли такая операция, и выдает сообщение об ошибке, если выявляет, что поле `final` не получило соответствующего исходного значения.

3.5 Управление доступом

Если бы любой член любого класса или объекта был способен напрямую обращаться к структурным элементам иного произвольно взятого класса или объекта, то восприятие, отладка и поддержка такого программного кода была бы крайне затруднительна. Одно из преимуществ ООП состоит в поддержке концепции инкапсуляции или сокрытия данных. Чтобы воплотить ее в жизнь, необходимы средства языка, позволяющие регламентировать, кто обладает доступом к членам класса или

интерфейса либо к классу или интерфейсу как таковому. Возможность инкапсуляции реализуется на практике с помощью модификаторов доступа, применяемых по отношению к классам и интерфейсам в целом и их членам в частности.

Все члены класса всегда доступны в контексте самого класса. Механизм управления доступом к членам класса извне основан на использовании модификаторов доступа четырех различных категорий.

- **private**. Элементы класса, помеченные как `private`, доступны только в контексте этого класса.
- **package (default)**. Элементы, объявленные без указания модификатора доступа, открыты для самого класса и классов, размещенных в том же пакете.
- **protected**. Элементы, обозначенные с помощью служебного слова `protected`, доступны в пределах «родного» класса, классов того же пакета и классов-наследников.
- **public**. Элементы, объявленные как `public`, открыты во всех случаях, когда доступен сам класс.

Модификаторы доступа `private` и `protected` применимы исключительно по отношению к членам классов, но не к самим классам или интерфейсам (если только последние не являются вложенными). Чтобы обеспечить возможность доступа к члену класса из некоторого блока кода, первым делом следует открыть для доступа класс, которому принадлежит член.

Важно понимать, что контроль доступа осуществляется на уровне классов (или интерфейсов), но не на уровне объектов. Это значит, что члены класса потенциально открыты для доступа, независимо от того, к какому именно объекту класса выполняется обращение в тот или иной момент.

3.6 Создание объектов

Объекты, представляющие небесные тела и соответствующие классу `Body`, могут создаваться и инициализироваться следующим образом:

Пример 3.

```
Body sun = new Body();
sun.idNum = Body.nextID++;
sun.name = "Солнце";
sun.orbits = null;

Body earth = new Body();
earth.idNum = Body.nextID++;
earth.name = "Земля";
earth.orbits = sun;
```

Сначала объявляется переменная `sun`, предназначенная для хранения ссылки на объект типа `Body`. Подобное объявление не предполагает немедленного создания объекта – оно всего лишь определяет переменную конкретного типа. Объект, на который ссылается переменная `sun`, создается посредством оператора ***new***. **Создание ссылки и создание объекта – различные операции!!!** Конструкция, предполагающая использование `new` – это наиболее употребительный способ создания объектов. Намереваясь создать объект с помощью оператора `new`, мы задаем имя соответствующего класса и перечисляем требуемые аргументы, если таковые предусмотрены. Исполняющая система выделяет фрагмент памяти, необходимый для размещения содержимого полей объекта, и инициализирует их значениями, принятыми по умолчанию. После этого отрабатывают блоки инициализации и конструкторы. По завершению процесса система возвращает ссылку на созданный объект.

Если система не находит достаточного фрагмента свободной памяти, она обычно активизирует процесс сборки мусора, чтобы попытаться освободить занятые участки

памяти. Если и далее нехватка памяти все еще ощущается, оператор `new` генерирует исключение типа `OutOfMemoryError`.

Объекты, создаваемые с помощью оператора `new`, нигде в программе явно не удаляются. Виртуальная машина Java всю ответственность за очистку памяти берет на себя, используя механизм сборки мусора, подразумевающий, что объекты, ссылки на которые отсутствуют, автоматически уничтожаются без вмешательства прикладной программы. Если объект больше не нужен, вы должны просто перестать на него ссылаться.

3.7 Конструкторы

Вновь созданный объект приобретает некоторое исходное состояние. Чтобы состояние оказалось корректным, можно предусмотреть явную инициализацию полей класса в момент их объявления либо положиться на значения, предлагаемые по умолчанию. Впрочем, часто случается, что для приведения объекта в требуемое исходное состояние простой инициализации недостаточно: например, для получения данных необходим дополнительный код либо операцию инициализации нельзя свести к присваиванию единственного выражения [3].

Для достижения целей, выходящих за рамки потребностей простой инициализации, в составе класса предусмотрены специальные члены – конструкторы (constructors). **Конструктор** – это блок выражений, которые используются для инициализации созданного объекта. Инициализация выполняется до того момента, когда оператор `new` вернет в вызывающий блок ссылку на объект. Конструкторы обладают тем же именем, что и класс, в составе которого они объявляются. При создании объекта класса следует указывать конструктор, который, подобно обычным методам класса, способен принимать любое (в том числе и нулевое) число аргументов, но в отличие от методов не может возвращать значения какого бы то ни было типа. Конструкторы вызываются после присваивания полям вновь созданного объекта значений по умолчанию и выполнения явных инструкций инициализации полей.

В дополненной версии класса `Body`, приведенной ниже, объект приводится в исходное состояние, как с помощью выражений инициализации, так и посредством конструктора.

Пример 4.

```
class Body {
    public long idNum;
    public String name = "<Без имени>";
    public Body orbits = null;

    private static long nextID = 0;

    Body() {
        idNum = nextID++;
    }
}
```

Объявление конструктора состоит из имени класса, за которым следует список (возможно, пустой) параметров, ограниченный круглыми скобками, и тело конструктора – блок выражений, заключенный в фигурные скобки.

Конструктор класса `Body` объявлен без параметров. Его назначение – обеспечивать уникальность значения поля `idNum` вновь создаваемого объекта. В исходном варианте класса небольшая ошибка, связанная, например, с неаккуратно выполненной операцией присваивания значения полю `idNum` либо с отсутствием инструкций приращения содержимого поля `nextID`, могла бы привести к тому, что несколько объектов `Body` получили бы один и тот же порядковый номер. Это было бы ошибкой, так как значения `idNum` различных объектов класса должны быть уникальными. Передав ответственность за выбор верных значений `idNum` самому классу, мы раз и навсегда избавляемся от

подобных ошибок. Теперь конструктор класса `Body` – это единственный субъект, который изменяет содержимое поля `nextID` и нуждается в доступе к нему. Поэтому мы должны обозначить переменную `nextID` модификатором `private`, чтобы предотвратить возможность обращения к ней за пределами класса. Сделав это, мы исключим один из потенциальных источников ошибок, грозящих будущим пользователям нашего класса.

Выражения инициализации переменных `name` и `orbits` в теле объявления класса присваивают последним некоторые допустимые и целесообразные значения. Теперь при создании объект автоматически приобретает исходный набор свойств, описывающих его состояние. Далее возможно изменить некоторые из них по своему усмотрению.

Пример 5.

```
Body sun = new Body(); // idNum = 0
sun.name = "Солнце";

Body earth = new Body(); // idNum = 1
earth.name = "Земля";
earth.orbits = sun;
```

В процессе создания объекта с помощью оператора `new` конструктор класса `Body` вызывается после присваивания полям `name` и `orbits` предусмотренных нами выражений инициализации.

Рассмотренный выше случай – когда заранее, в момент написания программы известно имя астрономического объекта и вокруг какого небесного тела проходит орбита его движения – относительно редок. Вполне резонным будет создать еще один конструктор, который принимает значение имени объекта и ссылки на центральный объект в качестве аргументов:

Пример 6.

```
Body (String bodyName, Body orbitsAround) {
    this();
    name = bodyName;
    orbits = orbitsAround;
}
```

В данном примере один конструктор класса обращается к другому посредством выражения ***this()*** – первой исполняемой инструкции в теле конструктора-инициатора. Подобное предложение называют ***явным вызовом конструктора***. Если конструктор, к которому вы намереваетесь обратиться явно, предполагает задание аргументов, при вызове они должны быть переданы. Какой из конструкторов будет вызван – это обуславливается количеством аргументов и набором их типов. В данном случае выражение `this()` означает вызов конструктора без параметров, позволяющего установить значение `idNum` объекта и увеличить текущее содержимое статического поля `nextID` на единицу. Обращение к `this()` дает возможность избежать повторения кода инициализации `idNum` и изменения `nextID`. Теперь код, предусматривающий создание объектов, становится существенно более простым:

Пример 7.

```
Body sun = new Body("Солнце", null);
Body earth = new Body("Земля", sun);
```

Версия конструктора, вызываемого в процессе выполнения оператора `new`, определяется структурой списка передаваемых аргументов.

Если применяется выражение явного вызова конструктора, оно должно быть первой исполняемой инструкцией в теле конструктора-инициатора. Выражения, используемые в качестве аргументов вызова, не должны содержать ссылки на поля и методы текущего объекта – во всех случаях предполагается, что на этой стадии конструирование объекта еще не завершено.

Применение специализированных конструкторов может быть обусловлено следующими причинами:

- Без помощи конструкторов с параметрами некоторые классы не в состоянии обеспечить свои объекты приемлемыми исходными значениями.
- При использовании дополнительных конструкторов задача определения начальных свойств объектов упрощается (наглядный пример – конструктор класса `Body` с двумя параметрами).
- Создание объекта зачастую связано с большими издержками из-за некорректного выбора инициализаторов. Например, если объект класса содержит таблицу и вычислительные затраты на ее конструирование велики, совершенно неразумно предусматривать инициализатор по умолчанию, заведомо зная, что позже придется отбросить результат его работы и повторить инициализацию, чтобы придать таблице свойства, нужные в конкретной ситуации. Целесообразнее сразу применить подходящий конструктор, который способен создать таблицу с требуемыми свойствами.
- Если конструктор не помечен признаком `public`, круг субъектов, которые могут им воспользоваться для создания экземпляров класса, ограничивается. Можно, например, запретить программистам, использующим пакет, создавать объекты класса, предусмотрев для всех конструкторов класса признак доступа на уровне пакета.

Весьма широкое применение находят и конструкторы без параметров. Если, объявляя класс, вы не создали ни одного конструктора, какой бы то ни было разновидности, компилятор автоматически включит в состав класса пустой конструктор без параметров. Подобный конструктор (его называют **конструктором по умолчанию**) создается только в том случае, если других конструкторов не существует, поскольку можно привести примеры классов, в которых применение конструктора без параметров нецелесообразно или вовсе невозможно. Если необходимы и конструктор без параметров, и один или несколько дополнительных конструкторов, первый должен быть создан явно. Конструктор по умолчанию получает тот же признак доступа, что и класс, для которого он создается: если объявление класса снабжено модификатором `public`, то и конструктор будет помечен как `public`.

В тексте конструкторов допускается упоминание объявляемых исключений. Предложение `throws` помещается после списка параметров, непосредственно перед открывающей фигурной скобкой, которая отмечает начало тела конструктора. Если в объявлении конструктора присутствует предложение `throws`, любой метод, который косвенным образом обращается к конструктору при выполнении оператора `new`, обязан либо обеспечить «отлов» исключений упомянутых типов с помощью соответствующих предложений `catch`, либо перечислить эти типы в разделе `throws` собственного объявления.

3.8 Блоки инициализации

Еще один способ осуществления сложных операций по инициализации полей объекта связан с использованием блоков инициализации, которые представляют собой наборы выражений инициализации полей, заключенные в фигурные скобки и размещенные внутри класса вне объявления методов или конструкторов. Блок инициализации выполняется так же, как если бы он был расположен в верхней части тела любого конструктора. Если блоков инициализации несколько, они выполняются в порядке следования в тексте класса. Блок инициализации способен генерировать исключения, если их объявления перечислены в предложениях `throws` всех конструкторов класса. В примере 8 конструктор без параметров заменен равноценным блоком инициализации:

Пример 8.

```
class Body {
    public long idNum;
    public String name = "<Без имени>";
    public Body orbits = null;
```

```

        private static long nextID = 0;

        {
            idNum = nextID++;
        }
        Body (String bodyName, Body orbitsAround) {
            name = bodyName;
            orbits = orbitsAround;
        }
    }
}

```

Теперь конструктор с двумя параметрами не должен выполнять явный вызов конструктора без параметров, как раньше, поскольку такового больше нет. Подобный вариант применения блока инициализации нельзя назвать самым удачным – мы обратились к нему только для того, чтобы продемонстрировать синтаксис. На практике блоки инициализации используют для выполнения более сложных операций, когда отсутствует необходимость в объявлении конструкторов с параметрами, и нет особых причин для обращения к конструктору без параметров. Основное назначение блоков инициализации, связано с обеспечением корректного исходного состояния создаваемого объекта, но в реальности можно заставить их выполнять любые необходимые операции – компилятор не проверяет, что именно делается внутри блока инициализации.

Статическим полям класса позволено иметь инициализаторы, но язык предусматривает возможность выполнения более сложных операций в контексте блока статической инициализации. Блок статической инициализации во многом подобен обычному блоку инициализации – отличия состоят в применении модификатора `static`, возможности обращения только к статическим членам класса и запрете на генерацию объявляемых исключений. Далее приведен пример статической инициализации небольшого массива простых чисел (простым называют натуральное число, которое без остатка делится только на единицу и само на себя).

Пример 9.

```

class Primes {
    static int[] knownPrimes = new int[4];

    static {
        knownPrimes[0] = 2;
        for (int i = 1; i < knownPrimes.length; i++)
            knownPrimes[i] = nextPrime(i);
    }
    // объявление nextPrime...
}

```

Поля класса инициализируются последовательно – сначала выполняются первые по порядку выражение или блок инициализации, затем следующие и т.д. Операции статической инициализации осуществляются при загрузке класса. В примере 9 компилятор может поручиться, что массив `knownPrimes` будет создан еще до того, как начнутся выполняться инструкции блока статической инициализации. Подобным образом обеспечиваются и гарантии того, что к моменту возможного обращения к статическому полю класса извне оно будет проинициализировано корректно.

3.9 Методы

Методы класса, как правило, содержат код, который способен адекватно воспринимать состояние объекта и изменять его. В некоторых классах предлагаются поля, помеченные модификатором `public` или `protected`, т.е. открытые для непосредственного обращения из стороннего программного кода, но в большинстве случаев подобный подход нельзя признать приемлемым. Большинство объектов предназначено для получения решений, которые нельзя свести к простым элементам данных.

Ниже рассмотрен текст метода main, который непосредственно вызывается и выполняется виртуальной машиной Java, предусматривающий создание объекта класса Body и вывод на экран содержимого его полей [3].

Пример 10.

```
class BodyPrint {
    public static void main (String[] args) {
        Body sun = new Body("Солнце", null);
        Body earth = new Body("Земля", sun);
        System.out.println("Тело " + earth.name +
+                               " вращается вокруг тела "
+                               earth.orbits.name +
+                               " и обладает номером "+
+                               earth.idnum);
    }
}
```

Конструкция объявления метода состоит из двух частей: заголовка метода и его тела. Заголовок в общем случае включает набор модификаторов, наименование типа возвращаемого значения, сигнатуру и предложение throws, перечисляющие классы исключений, которые могут генерироваться методом. Сигнатура метода состоит из наименования – идентификатора метода и списка (возможно, пустого) типов параметров, заключенного в круглые скобки. В объявлении любого метода должны быть указаны, по меньшей мере, тип возвращаемого значения и сигнатура – модификаторы и список throws необязательны. Тело метода представляет собой набор выражений, ограниченный фигурными скобками.

В объявлении метода применяются модификаторы следующих категорий:

- **Модификаторы доступа** (уже были рассмотрены ранее).
- **abstract.** Модификатором abstract помечаются методы, которые точно не определены в контексте текущего класса. В объявлении методов abstract отсутствует тело – оно заменяется символом точки с запятой. Предполагается, что абстрактные методы должны быть реализованы в некотором производном классе.
- **static.** Обсудим далее.
- **final.** Методы, обозначенные как final, не допускают переопределения в производных классах.
- **synchronized.** Метод, помеченный как synchronized, обладает дополнительной семантикой, касающейся проблемы управления вычислительными потоками, одновременно выполняющимися в контексте программного приложения.
- **native.** Обсудим далее.
- **strict floating point.** Метод, объявленный как strictfp, гарантирует, что все предусмотренные им операции с плавающей запятой, будут выполняться точно и единообразно всеми виртуальными машинами Java. Признак strictfp, содержащийся в объявлении класса, неявно распространяется на все методы класса, манипулирующими числами с плавающей запятой.

Метод abstract не может быть одновременно помечен любым из модификаторов – static, final, synchronized, native или strictfp.

Метод, обозначенный как static, служит атрибутом класса в целом, а не частных экземпляров этого класса. Поэтому подобные методы принято называть методами класса. Статические методы обычно предназначаются для выполнения задач, общих для класса, – таких как, скажем, получение очередного доступного серийного номера объекта и т.п. Статический метод способен обращаться только к полям и другим методам класса, обозначенным модификатором static, поскольку для доступа к нестатическим членам необходимо наличие ссылки на конкретный объект класса, а в пределах статического

метода подобной ссылки не существует, так как возможность обращения к `this` отсутствует.

Методы вызываются посредством задания ссылки на объект класса (скажем, *reference*) и наименования метода со списком аргументов (`method(arguments)`). Ссылка на объект и наименование метода объекта разделяются оператором точки (`.`):

```
reference.method(arguments);
```

В примере класса `BodyPrint` (пример 10) мы обращались к методу `println`, используя статическую ссылку `System.out` и передавая в качестве аргумента объект типа `String`, сформированный с помощью оператора сцепления (`+`) из нескольких строковых операндов.

Каждый метод объявляется с указанием конкретного числа параметров простых или объектных типов. Java не допускает объявления методов с переменным числом параметров, хотя подобное ограничение вполне преодолимо – достаточно передать методу в качестве аргумента ссылку на массив объектов. При обращении к методу вызывающий код обязан предоставить набор аргументов, соответствующих по количеству и совместимых по типам со списком параметров в объявлении метода.

В объявлении метода необходимо задавать наименование типа (простого или ссылочного) возвращаемого значения. Если семантика метода такова, что он не должен возвращать каких бы то ни было значений, вместо имени типа возвращаемого значения указывается служебное слово ***void***.

Пример `BodyPrint` (пример 10) иллюстрирует типичную ситуацию, связанную с проверкой состояния объекта класса. Однако более предпочтительным следует считать решение, в котором вместо прямого обращения к полям объекта в классе предусмотрен некоторый метод, который способен возвратит строковое представление состояния. Ниже представлен метод в составе класса `Body`, возвращающий ссылку на объект типа `String`, который описывает совокупность требуемых полей конкретного экземпляра `Body`.

Пример 11.

```
public String toString() {  
    String desc = idNum + " (" + name + ") ";  
    if (orbits != null)  
        desc += "вращается вокруг " + orbits.toString();  
    return desc;  
}
```

Операторы `+` и `+=`, используемые в тексте метода, выполняют функции сцепления (конкатенации) строк. Сначала создаётся строка `desc`, содержащая номер небесного тела и его наименование. Если тело является спутником другого тела, в `desc` добавляется текстовое описание тела более высокого уровня, получаемое с помощью того же метода `toString`. Подобный рекурсивный процесс повторяется несколько раз, если иерархия небесных тел сложна (например, Луна-Земля-Солнце) – где каждое предыдущее тело служит спутником следующего, – и завершается по достижении «самого» центрального тела системы, когда поле `orbits` соответствующего объекта класса `Body` содержит значение `null`. Наконец, с помощью команды `return` строка `desc` возвращается в то место кода, откуда был осуществлён вызов метода.

Метод `toString` объекта имеет специальное назначение – он вызывается для получения строкового представления состояния объекта в тех ситуациях, когда в выражениях конкатенации присутствует ссылка на сам объект. Рассмотрим следующие выражения:

```
System.out.println("Тело " + sun);  
System.out.println("Тело " + earth);
```

Методы `toString` объектов `sun` и `earth` будут вызваны неявно и обеспечат вывод на экран следующих результатов:

```
Тело 0 (Солнце)  
Тело 1 (Земля) вращается вокруг 0 (Солнце)
```

Метод `toString` заведомо присутствует в составе любого объекта, независимо от того, был ли он (метод) объявлен в соответствующем классе или нет, поскольку все классы Java наследуют класс `Object`, а в нём метод `toString` предусмотрен заранее. Класс `Object` и вопросы наследования классов будут рассмотрены далее.

При вызове метода управление передаётся из того места кода, откуда был осуществлён вызов, в тело метода. Выражения тела метода выполняются в порядке, предусмотренном его семантикой. Метод завершает работу и передаёт управление обратно в код-инициатор в результате возникновения одного из трёх возможных событий: выполнение команды `return`, достижения конца тела метода (только для методов, «возвращающих» `void`) или генерации исключения. **Если метод предусматривает возврат какого-либо значения, им может быть только одно значение простого или объектного типа.** Методы, которые по смыслу должны возвращать несколько значений, способны решить эту задачу одним из следующих способов:

- вернуть ссылку на объект, который содержит поля с требуемыми значениями;
- разместить результаты работы в одном или нескольких объектах, на которые указывают ссылки, переданные методу в качестве аргументов;
- вернуть массив объектов нужного типа.
- разместить результат в `public`-полях этого же объекта.

Предположим, например, что нам необходимо создать метод, который должен вернуть информацию о том, какие операции с конкретным банковским счётом разрешено выполнять его владельцу. Следует принять во внимание, что владелец счёта может быть наделён несколькими полномочиями (по выполнению приходных, расходных операций и т.п.) одновременно, поэтому метод должен обладать способностью возвращать группу значений. Решение задачи можно начать с объявления класса `Permissions`, объекты которого позволяют хранить булевы значения, подтверждающие права владельца на выполнение тех или иных банковских операций:

Пример 11.

```
public class Permissions {  
    public boolean canDeposit,      // Приход  
                    canWithdraw, // Расход  
                    canClose;      // Закрытие счёта  
}
```

А вот так может выглядеть метод, предусматривающий заполнение полей объекта типа `Permissions` и возврат его:

```
public class BankAccount {  
    private long number; // Номер счёта  
    private long balance; // Остаток на счёте  
    public Permissions permissionsFor(Person who) {  
        Permissions perm = new Permissions();  
        perm.canDeposit = canDeposit(who);  
        perm.canWithdraw = canWithdraw(who);  
        perm.canClose = canClose(who);  
        return perm;  
    }  
    // Объявление методов canDeposit и т.д...  
}
```

Любой вызов метода, предусматривающего возврат некоторого значения, должен завершаться либо выполнением соответствующей команды `return`, передающей в код-инициатор значение, которое может быть присвоено переменной соответствующего типа, либо выбрасыванием объекта исключения. Методом `permissionsFor` не должен возвращаться, например, объект класса `String`, поскольку тот не удастся присвоить переменной типа `Permissions` (в таких ситуациях говорят о несовместимости типов). Но ничто не запрещает объявить в качестве наименования типа, возвращаемого методом `permissionsFor`, класс `Object`, совершенно не затрагивая выражения `return`, так как ссылка

на объект `Permissions` может быть присвоена переменной типа `Object` (если вспомнить, что класс `Object` является базовым по отношению ко всем остальным классам Java).

При вызове метода аргументы передаются «по значению». Другими словами, значения переменных-параметров в теле метода – это копии тех значений, которые код-инициатор передал методу в виде аргументов. Если, например, метод получает в качестве аргументов значение типа `double`, в теле метода соответствующий параметр сохраняет копию значения, и возможные изменения этой копии в процессе работы метода никоим образом не воздействуют на содержимое переменных в коде-инициаторе. Рассмотрим следующий пример:

Пример 12.

```
class PassByValue { // Передача аргумента по значению
    public static void main (String[] args) {
        double one = 1.0;
        System.out.println("до: one = " + one);
        halveIt(one);
        System.out.println("после: one = " + one);
    }

    public static void halveIt(double arg) {
        arg /= 2.0; //Разделить значение arg пополам
        System.out.println("половина: arg = " + arg);
    }
}
```

Ниже показан результат работы программы – операция деления значения параметра `arg` в методе `halveIt` не влияет на содержимое переменной `one` в теле метода `main`.

```
до: one = 1.0
половина: arg = 0.5
после: one = 1.0
```

Следует заметить, что если аргумент представляет собой ссылку на объект, то «по значению» передаётся именно ссылка, но не объект как таковой. Таким образом, внутри тела метода можно присвоить ссылке другой объект, не воздействуя на содержимое исходной ссылки. Но если, пользуясь переданной ссылкой, попробовать изменить значение поля объекта, на который она указывает, или вызвать какой-либо из методов, изменяющих состояние объекта, тот действительно изменит своё состояние с точки зрения любого блока программы, в котором имеются ранее созданные ссылки на объект. Чтобы продемонстрировать названные нюансы, приведём пример:

Пример 13.

```
class PassRef { // Передача аргумента-ссылки
    public static void main (String[] args) {
        Body venus = new Body ("Венера", null);
        System.out.println("до: " + venus);
        commonName(venus);
        System.out.println("после: " + venus);
    }

    public static void commonName (Body bodyRef) {
        bodyRef.name = "Утренняя звезда";
        bodyRef = null;
    }
}
```

Результат работы программы выглядит так:

```
до: 0 (Венера)
после: 0 (Утренняя звезда)
```

Обратите внимание, что состояние «внешнего» по отношению к методу `commonName` объекта поддавалось изменению «изнутри» этого метода; кроме того,

переменная *venus* всё ещё сохранила ссылку на тот же объект класса *Body*, а в методе *commonName* копия *bodyRef* ссылочной переменной *venus* получила другое значение *null*.

Существует ещё одно средство языка, имеющее прямое отношение к обсуждаемой теме: в объявлении метода параметры позволено пометить модификатором *final*, имея в виду, что значение параметра не может быть изменено в ходе выполнения тела метода. Будь *bodyRef* объявлен как *final*, компилятор не позволил бы изменить его значение на *null*. Если логика программы обуславливает недопустимость изменения значения параметра, достаточно снабдить его признаком *final*.

Способы запуска программ на выполнение в большой степени зависят от особенностей той или иной операционной системы, но в любом случае, чтобы активизировать приложение, вы обязаны указать имя некоторого класса. При запуске программы система пытается обнаружить в указанном классе метод *main* и передать ему управление. В объявлении метода *main* должны присутствовать модификаторы *public* и *static*, а также служебное слово *void*, а в списке параметров – единственный параметр типа *String[]* (ссылка на массив ссылок на объекты типа *String*). Метод *main* объявляется как *public*, с тем, чтобы обратиться к нему мог каждый субъект (в данном случае, виртуальная машина Java) и как *static* – последнее означает, что метод принадлежит самому классу и не «привязан» ни к одному из частных экземпляров этого класса. Метод *main* объявлен как *void* - это значит, что он не возвращает никаких значений. В примере 14 приведён метод *main*, который выводит на экран значения переданных ему аргументов:

Пример 14.

```
class Echo {  
    public static void main(String[] args) {  
        for (int i = 0; i < args.length; i++)  
            System.out.print(args[i] + " ");  
        System.out.println();  
    }  
}
```

Строковые аргументы, передаваемые методу *main*, играют роль параметров командной строки, которые могут быть введены пользователем при старте программы. Например, из командной строки приложение *Echo* может быть запущено следующим образом:

```
java Echo Здесь был Вася
```

В данном случае слово *java* обозначает наименование программы-интерпретатора байт-кода, *Echo* соответствует имени класса, подлежащего выполнению, а остальные слова – это аргументы программы. Команда *java* находит откомпилированный код класса *Echo*, загружает его в виртуальную машину Java и вызывает метод *Echo.main*, передавая ему полученные извне параметры командной строки, которые сохраняются в массиве *args* объектов типа *String*. Результат работы программы выглядит так:

```
Здесь был Вася
```

Приложение как совокупность классов может содержать несколько методов *main* – таковые допустимо объявлять в любом классе, входящем в приложение, но в каждом конкретном случае запуска приложения используется только один метод *main* - он принадлежит классу, наименование которого указано в командной строке (как, например, *Echo*).

3.10 Лексические основы языка

Программы на языке Java - это набор пробелов, комментариев, ключевых слов, идентификаторов, литеральных констант, операторов и разделителей.

Хотя комментарии никак не влияют на исполняемый код программы, при правильном использовании они оказываются весьма существенной частью исходного текста. Существует три разновидности комментариев: комментарии в одной строке, комментарии в нескольких строках и, наконец, комментарии для документирования. Комментарии, занимающие одну строку, начинаются с символов *//* и заканчиваются в

конец строки. Такой стиль комментирования полезен для размещения кратких пояснений к отдельным строкам кода [3]:

```
a = 42; // это комментарий
```

Для более подробных пояснений вы можете воспользоваться комментариями, размещенными на нескольких строках, начав текст комментариев символами `/*` и закончив символами `*/`. При этом весь текст между этими парами символов будет расценен как комментарий и транслятор его проигнорирует.

```
/*
это тоже комментарий
*/
```

Третья, особая форма комментариев, предназначена для сервисной программы `javadoc`, которая использует компоненты Java-компилятора для автоматической генерации документации по типам. Соглашение, используемое для комментариев этого вида, таково: для того, чтобы разместить перед объявлением класса, метода или переменной документирующий комментарий, нужно начать его с символов `/**`. Заканчивается такой комментарий точно так же, как и обычный комментарий - символами `*/`. Программа `javadoc` умеет различать в документирующих комментариях некоторые специальные элементы, имена которых начинаются с символа `@`. Вот пример такого комментария:

```
/**
 * Этот класс умеет делать замечательные вещи.
 * @see java.applet.Applet
 * @author Patrick Naughton
 * @version 1. 2
 */
class CoolApplet extends Applet { /**
 * У этого метода два параметра:
 * @param key - это имя параметра.
 * @param value - это значение параметра с именем key.
 */ void put (String key, Object value) {
```

Тэг `@see` создаёт перекрёстную ссылку, указывающую на другой документ `javadoc`. Тэг `@author` позволяет включить в комментарий сведения об авторе кода. Тэг `@version` служит для обозначения версии программного обеспечения. Тэг `@param` служит для документирования одного параметра метода. Для обозначения каждого параметра надлежит задавать отдельный тэг `@param`. Первое слово, следующее за тэгом, трактуется как наименование параметра, а остальная часть строки – как его описание.

Служебные слова. Служебные слова не могут применяться в качестве идентификаторов, поскольку правила их употребления строго регламентированы в самом языке. В таблице 3.1 перечислены служебные слова языка программирования Java.

Таблица 3.1 - Зарезервированные слова Java

abstract	assert	boolean	break	byte
case	catch	char	class	const
continue	default	do	double	else
extends	final	finally	float	for
goto	if	implements	import	instanceof
int	interface	long	native	new
package	private	protected	public	return
short	static	strictfp	super	switch

synchronized	this	throw	throws	transient
try	void	volatile	while	

null, true, и false хотя и выглядят как служебные слова, формально относятся к числу литералов и поэтому в таблице не включены.

Отметим, что слова const и goto зарезервированы в Java, но пока не используются. Кроме этого, в Java есть зарезервированные имена методов (см. таблицу 3.2). Эти методы наследуются каждым классом.

Таблица 3.2 - Зарезервированные имена методов Java

clone	equals	finalize	getClass	hashCode
notify	notifyAll	toString	wait	

Идентификаторы. Идентификаторы используются для именования классов, методов и переменных. В качестве идентификатора может использоваться любая последовательность строчных и прописных букв, цифр и символов _ (подчеркивание) и \$ (доллар). Идентификаторы не должны начинаться с цифры, чтобы транслятор не перепутал их с числовыми литеральными константами, которые будут описаны ниже. Java - язык, чувствительный к регистру букв. Это означает, что, к примеру, Value и VALUE - различные идентификаторы.

Константы. Константы в Java задаются их литеральным представлением. Целые числа, числа с плавающей точкой, логические значения, символы и строки можно располагать в любом месте исходного кода.

Целые числа. Целые числа - это тип, используемый в обычных программах наиболее часто. Любое целочисленное значение, например, 1, 2, 3, 42 - это целый литерал. В данном примере приведены десятичные числа, то есть именно те, которые мы повседневно используем вне мира компьютеров. Кроме десятичных, в качестве целых литералов могут использоваться также восьмеричные и шестнадцатеричные числа. Java распознает восьмеричные числа по стоящему впереди нулю. Нормальные десятичные числа не могут начинаться с нуля, так что использование в программе внешне допустимого числа 09 приведет к сообщению об ошибке при трансляции, поскольку 9 не входит в диапазон 0..7, допустимый для знаков восьмеричного числа. Шестнадцатеричная константа различается по стоящим впереди символам нуль-х (0x или 0X). Диапазон значений шестнадцатеричной цифры - 0.. 15, причем в качестве цифр для значений 10.. 15 используются буквы от A до F (или от a до f). С помощью шестнадцатеричных чисел вы можете в краткой и ясной форме представить значения, ориентированные на использование в компьютере, например, написав 0xffff вместо 65535.

Целые литералы. Целые литералы являются значениями типа int, которое в Java хранится в 32-битовом слове. Если вам требуется значение, которое по модулю больше, чем приблизительно 2 миллиарда, необходимо воспользоваться константой типа long. При этом число будет храниться в 64-битовом слове. К числам с любым из названных выше оснований вы можете приписать справа строчную или прописную букву L, указав таким образом, что данное число относится к типу long. Например, 0x7fffffffffffffffL или 9223372036854775807L - это значение, наибольшее для числа типа long.

Литералы с плавающей точкой. Числа с плавающей точкой представляют десятичные значения, у которых есть дробная часть. Их можно записывать либо в обычном, либо в экспоненциальном форматах. В обычном формате число состоит из некоторого количества десятичных цифр, стоящей после них десятичной точки, и следующих за ней десятичных цифр дробной части. Например, 2.0, 3.14159 и .6667 - это допустимые значения чисел с плавающей точкой, записанных в стандартном формате. В экспоненциальном формате после перечисленных элементов дополнительно указывается

десятичный порядок. Порядок определяется положительным или отрицательным десятичным числом, следующим за символом E или e. Примеры чисел в экспоненциальном формате: 6.022e23, 314159E-05, 2e+100. В Java числа с плавающей точкой по умолчанию рассматриваются, как значения типа double. Если вам требуется константа типа float, справа к литералу надо приписать символ F или f. Если вы любите избыточные определения - можете добавлять к литералам типа double символ D или d. Значения используемого по умолчанию типа double хранятся в 64-битовом слове, менее точные значения типа float - в 32-битовых.

Логические литералы. У логической переменной может быть лишь два значения - true (истина) и false (ложь). Логические значения true и false не преобразуются ни в какое числовое представление. В отличие от C и C++, ключевое слово true в Java не равно 1, а false не равно 0. В Java эти значения могут присваиваться только переменным типа boolean либо использоваться в выражениях с логическими операторами.

Символьные литералы. Символы в Java - это индексы в таблице символов UNICODE. Они представляют собой 16-битовые значения, которые можно преобразовать в целые числа и к которым можно применять операторы целочисленной арифметики, например, операторы сложения и вычитания. Символьные литералы помещаются внутри пары апострофов (' '). Все видимые символы таблицы ASCII можно прямо вставлять внутрь пары апострофов: - 'a', 'z', '@'. Для символов, которые невозможно ввести непосредственно, предусмотрено несколько управляющих последовательностей (см. таблицу 3.3).

Таблица 3.3 - Управляющие последовательности символов

Управляющая последовательность	Описание
\ddd	Восьмеричный символ (ddd)
\uxxxx	Шестнадцатеричный символ UNICODE (xxxx)
\'	Апостроф
\"	Кавычка
\\	Обратная косая черта
\r	Возврат каретки (carriage return)
\n	Перевод строки (line feed, new line)
\f	Перевод страницы (form feed)
\t	Горизонтальная табуляция (tab)
\b	Возврат на шаг (backspace)

Строчные литералы. Строчные литералы в Java выглядят точно также, как и во многих других языках - это произвольный текст, заключенный в пару двойных кавычек (""). Хотя строчные литералы в Java реализованы весьма своеобразно (Java создает объект для каждой строки), внешне это никак не проявляется. Примеры строчных литералов: "Hello World!"; "две\строки; \ A это в кавычках\"". Все управляющие последовательности и восьмеричные / шестнадцатеричные формы записи, которые определены для

символьных литералов, работают точно так же и в строках. Строчные литералы в Java должны начинаться и заканчиваться в одной и той же строке исходного кода. В этом языке, в отличие от многих других, нет управляющей последовательности для продолжения строкового литерала на новой строке.

3.11 Типы данных

Оператор - это нечто, выполняющее некоторое действие над одним или двумя аргументами и выдающее результат. Синтаксически операторы чаще всего размещаются между идентификаторами и литералами. Детально операторы будут рассмотрены далее, их перечень приведен в таблице 3.4 [3].

Таблица 3.4 - Операторы языка Java

+	+=	-	-=
*	*=	/	/=
	=	^	^=
&	&=	%	%=
>	>=	<	<=
!	!=	++	--
>>	>>=	<<	<<=
>>>	>>>=	&&	
==	=	~	?:
	instanceof	[]	

Разделители. Лишь несколько групп символов, которые могут появляться в синтаксически правильной Java-программе, все еще остались незазванными. Это - простые разделители, которые влияют на внешний вид и функциональность программного кода (см. таблицу 3.5).

Таблица 3.5 - Разделители

Символы	Название	Для чего применяются
()	круглые скобки	Выделяют списки параметров в объявлении и вызове метода, также используются для задания приоритета операций в выражениях, выделения выражений в операторах управления выполнением программы, и в операторах приведения типов.
{ }	фигурные скобки	Содержат значения автоматически инициализируемых массивов, также используются для ограничения блока кода в классах, методах и локальных областях видимости.
[]	квадратные скобки	Используются в объявлениях массивов и при доступе к отдельным элементам массива.

;	точка с запятой	Разделяет операторы.
,	запятая	Разделяет идентификаторы в объявлениях переменных, также используется для связи операторов в заголовке цикла for.
.	точка	Отделяет имена пакетов от имен подпакетов и классов, также используется для отделения имени переменной или метода от имени переменной.

Переменные. Переменная характеризуется комбинацией идентификатора, типа и области действия. В зависимости от того, где вы объявили переменную, она может быть локальной, например, для кода внутри цикла for, либо это может быть переменная экземпляра класса, доступная всем методам данного класса. Локальные области действия объявляются с помощью фигурных скобок.

Простые типы. Простые типы в Java не являются объектно-ориентированными, они аналогичны простым типам большинства традиционных языков программирования. В Java имеется восемь простых типов: - byte, short, int, long, char, float, double и boolean. Их можно разделить на четыре группы:

Целые. К ним относятся типы byte, short, int и long. Эти типы предназначены для целых чисел со знаком.

Типы с плавающей точкой - float и double. Они служат для представления чисел, имеющих дробную часть.

Символьный тип char. Этот тип предназначен для представления элементов из таблицы символов, например, букв или цифр.

Логический тип boolean. Это специальный тип, используемый для представления логических величин.

В Java отсутствует автоматическое приведение типа с сужением области определения. Несовпадение типов приводит не к предупреждению при трансляции, а к сообщению об ошибке. Для каждого типа строго определены наборы допустимых значений и разрешенных операций.

Целые числа. В языке Java понятие беззнаковых чисел отсутствует. Все числовые типы этого языка - знаковые. Отсутствие в Java беззнаковых чисел вдвое сокращает количество целых типов. В языке имеется 4 целых типа, занимающих 1, 2, 4 и 8 байтов в памяти. Для каждого типа - byte, short, int и long, есть свои естественные области применения.

Byte. Тип byte - это знаковый 8-битовый тип. Его диапазон - от -128 до 127. Он лучше всего подходит для хранения произвольного потока байтов, загружаемого из сети или из файла.

```
byte b;
byte c = 0x55;
```

Если речь не идет о манипуляциях с битами, использования типа byte, как правило, следует избегать. Для нормальных целых чисел, используемых в качестве счетчиков и в арифметических выражениях, гораздо лучше подходит тип int.

Short. short - это знаковый 16-битовый тип. Его диапазон - от -32768 до 32767. Это, вероятно, наиболее редко используемый в Java тип, поскольку он определен, как тип, в котором старший байт стоит первым.

```
short s;
short t = 0x55aa;
```

Случилось так, что на ЭВМ различных архитектур порядок байтов в слове различается, например, старший байт в двухбайтовом целом short может храниться первым, а может и последним. Первый случай имеет место в архитектурах SPARC и Power PC, второй - для микропроцессоров Intel x86. Переносимость программ Java

требует, чтобы целые значения одинаково были представлены на ЭВМ разных архитектур.

Int. Тип `int` служит для представления 32-битных целых чисел со знаком. Диапазон допустимых для этого типа значений - от -2147483648 до 2147483647. Чаще всего этот тип данных используется для хранения обычных целых чисел со значениями, достигающими двух миллиардов. Этот тип прекрасно подходит для использования при обработке массивов и для счетчиков. Всякий раз, когда в одном выражении фигурируют переменные типов `byte`, `short`, `int` и целые литералы, тип всего выражения перед завершением вычислений приводится к `int`.

```
int i;
int j = 0x55aa0000;
```

Long. Тип `long` предназначен для представления 64-битовых чисел со знаком.

```
long m;
long n = 0x55aa000055aa0000;
```

Не надо отождествлять разрядность целочисленного типа с занимаемым им количеством памяти. Исполняющий код Java может использовать для ваших переменных то количество памяти, которое сочтет нужным, лишь бы только их поведение соответствовало поведению типов, заданных вами. Ниже приведена таблица разрядностей и допустимых диапазонов для различных типов целых чисел.

Таблица 3.6 – Разрядность и диапазон значений целых чисел

Имя	Разрядность	Диапазон
<code>long</code>	64	-9, 223, 372, 036, 854, 775, 808.. 9, 223, 372, 036, 854, 775, 807
<code>int</code>	32	-2, 147, 483, 648.. 2, 147, 483, 647
<code>short</code>	16	-32, 768.. 32, 767
<code>byte</code>	8	-128.. 127

Числа с плавающей точкой. Числа с плавающей точкой, часто называемые в других языках вещественными числами, используются при вычислениях, в которых требуется использование дробной части. В Java реализован стандартный (IEEE-754) набор типов для чисел с плавающей точкой - `float` и `double` и операторов для работы с ними. Характеристики этих типов приведены в таблице 3.7.

Таблица 3.7 – Разрядность и диапазон значений чисел с плавающей точкой

Имя	Разрядность	Диапазон
<code>double</code>	64	1.7e-308.. 1. 7e+ 308
<code>float</code>	32	3.4e-038.. 3. 4e+ 038

Float. В переменных с обычной, или одинарной точностью, объявляемых с помощью ключевого слова `float`, для хранения вещественного значения используется 32 бита.

```
float f;
float f2 = 3. 14F; // обратите внимание на F,
//т.к. по умолчанию все литералы double
```

Double. В случае двойной точности, задаваемой с помощью ключевого слова `double`, для хранения значений используется 64 бита. Все трансцендентные математические функции, такие, как `sin`, `cos`, `sqrt`, возвращают результат типа `double`.

```
double d;
```



```
double pi = 3. 14159265358979323846;
```

Приведение типа. Приведение типов (type casting) - одно из неприятных свойств C++, тем не менее приведение типов сохранено и в языке Java. Иногда возникают ситуации, когда у вас есть величина какого-то определенного типа, а вам нужно ее присвоить переменной другого типа. Для некоторых типов это можно проделать и без приведения типа, в таких случаях говорят об автоматическом преобразовании типов. В Java автоматическое преобразование возможно только в том случае, когда точности представления чисел переменной-приемника достаточно для хранения исходного значения. Такое преобразование происходит, например, при занесении значения переменной типа byte или short в переменную типа int. Это называется расширением (widening) или повышением (promotion), поскольку тип меньшей разрядности расширяется (повышается) до большего совместимого типа. Размера типа int всегда достаточно для хранения чисел из диапазона, допустимого для типа byte, поэтому в подобных ситуациях оператора явного приведения типа не требуется. Обратное, в большинстве случаев, неверно, поэтому для занесения значения типа int в переменную типа byte необходимо использовать оператор приведения типа. Эту процедуру иногда называют сужением (narrowing), поскольку вы явно сообщаете транслятору, что величину необходимо преобразовать, чтобы она уместилась в переменную нужного вам типа. Для приведения величины к определенному типу перед ней нужно указать этот тип, заключенный в круглые скобки. В приведенном ниже фрагменте кода демонстрируется приведение типа источника (переменной типа int) к типу приемника (переменной типа byte). Если бы при такой операции целое значение выходило за границы допустимого для типа byte диапазона, оно было бы уменьшено путем деления по модулю на допустимый для byte диапазон (результат деления по модулю на число - это остаток от деления на это число).

```
int a = 100;  
byte b = (byte) a;
```

Автоматическое преобразование типов в выражениях. Когда вы вычисляете значение выражения, точность, требуемая для хранения промежуточных результатов, зачастую должна быть выше, чем требуется для представления окончательного результата.

```
byte a = 40;  
byte b = 50;  
byte c = 100;  
int d = a * b / c;
```

Результат промежуточного выражения (a * b) вполне может выйти за диапазон допустимых для типа byte значений. Именно поэтому Java автоматически повышает тип каждой части выражения до типа int, так что для промежуточного результата (a * b) хватает места.

Автоматическое преобразование типа иногда может оказаться причиной неожиданных сообщений транслятора об ошибках. Например, показанный ниже код, хотя и выглядит вполне корректным, приводит к сообщению об ошибке на фазе компиляции. В нем мы пытаемся записать значение 50 * 2, которое должно прекрасно уместиться в тип byte, в байтовую переменную. Но из-за автоматического преобразования типа результата в int мы получаем сообщение об ошибке от транслятора - ведь при занесении int в byte может произойти потеря точности.

```
byte b = 50;  
b = b * 2;
```

Исправленный текст :

```
byte b = 50;  
b = (byte) (b * 2);
```

что приводит к занесению в b правильного значения 100.

Если в выражении используются переменные типов byte, short и int, то во избежание переполнения тип всего выражения автоматически повышается до int. Если же

в выражении тип хотя бы одной переменной - long, то и тип всего выражения тоже повышается до long. Не забывайте, что все целые литералы, в конце которых не стоит символ L (или l), имеют тип int.

По умолчанию Java рассматривает все литералы с плавающей точкой, как имеющие тип double. Приведенная ниже программа показывает, как повышается тип каждой величины в выражении для достижения соответствия со вторым операндом каждого бинарного оператора.

```
class Promote {
    public static void main (String[] args) { byte b = 42;
        char c = 'a';
        short s = 1024;
        int i = 50000;
        float f = 5.67f;
        double d = .1234;
        double result = (f* b) + (i/ c) - (d* s);
        System.out.println
            ((f* b)+ " + " + (i / c)+ " - " + (d* s));
        System.out.println ("result = "+ result);
    }
}
```

Подвыражение $f * b$ - это число типа float, умноженное на число типа byte. Поэтому его тип автоматически повышается до float. Тип следующего подвыражения i / c (int, деленный на char) повышается до int. Аналогично этому тип подвыражения $d * s$ (double, умноженный на short) повышается до double. На следующем шаге вычислений мы имеем дело с тремя промежуточными результатами типов float, int и double. Сначала при сложении первых двух тип int повышается до float и получается результат типа float. При вычитании из него значения типа double тип результата повышается до double. Окончательный результат всего выражения - значение типа double.

Символы. Поскольку в Java для представления символов в строках используется кодировка Unicode, разрядность типа char в этом языке - 16 бит. В нем можно хранить десятки тысяч символов интернационального набора символов Unicode. Диапазон типа char - 0..65536. Unicode - это объединение десятков кодировок символов, он включает в себя латинский, греческий, арабский алфавиты, кириллицу и многие другие наборы символов.

```
char c;
char c2 = 0xf132;
char c3 = ' a';
char c4 = '\n';
```

Хотя величины типа char и не используются, как целые числа, вы можете оперировать с ними так, как если бы они были целыми числами. Это дает вам возможность сложить два символа вместе, или инкрементировать значение символьной переменной. В приведенном ниже фрагменте кода мы, располагая базовым символом, прибавляем к нему целое число, чтобы получить символьное представление нужной нам цифры.

```
int three = 3;
char one = '1';
char four = (char) (three+ one);
```

В результате выполнения этого кода в переменную four заносится символьное представление нужной нам цифры - '4'. Обратите внимание - тип переменной one в приведенном выше выражении повышается до типа int, так что перед занесением результата в переменную four приходится использовать оператор явного приведения типа.

Тип boolean. В языке Java имеется простой тип boolean, используемый для хранения логических значений. Переменные этого типа могут принимать всего два значения - true (истина) и false (ложь). Значения типа boolean возвращаются в качестве

результата всеми операторами сравнения, например ($a < b$). Кроме того, `boolean` - это тип, требуемый всеми условными операторами управления - такими, как `if`, `while`, `do`.

```
boolean done = false;
```

3.12 Массивы

Массивы (`arrays`) – это упорядоченные наборы элементов одного типа. Элементами массива могут служить объекты простых и ссылочных типов, в том числе и ссылки на другие массивы. Массивы сами по себе являются объектами и наследуют класс `Object` [3].

Объявление

```
int[] ia = new int[3];
```

создаёт массив с именем `ia`, который изначально указывает на набор из трёх элементов типа `int`. В объявлении массива его размерность не указывается. Количество элементов массива задаётся при его создании посредством оператора `new`. Длина массива фиксируется в момент создания и в дальнейшем изменению не поддаётся. Впрочем, переменной типа массива (в нашем примере - `ia`) в любой момент может быть поставлен в соответствие новый массив с другой размерностью.

Доступ к элементам массива осуществляется по значениям их номеров-индексов. Первый элемент массива имеет индекс, равный 0, а последний – `length-1`. Обращение к элементу массива выполняется посредством задания имени массива и значения индекса, заключённого в квадратные скобки. При каждом обращении к элементу массива по индексу исполняющая система Java проверяет, находится ли значение индекса в допустимых пределах, и генерирует исключение типа `ArrayIndexOutOfBoundsException`, если результат проверки ложен. Выражение индекса должно относиться к типу `int` – только этим и ограничивается максимальное количество элементов массива.

Длину массива можно определить с помощью поля `length` объекта массива (которое неявно снабжено признаками `public` и `final`). Ниже приведён дополненный код прежнего примера, в котором предусмотрено выполнение цикла, обеспечивающего вывод на экран содержимого каждого элемента массива `ia`:

```
for (int i = 0; i < ia.length; i++)
    System.out.println(i+ ": " + ia[i]);
```

Массив нулевой длины (т.е. такой, в котором нет элементов) принято называть пустым. Обратите внимание, что ссылка на массив, равная значению `null`, и ссылка на пустой массив – это совершенно разные вещи. Пустой массив – это реальный массив, в котором попросту отсутствуют элементы. Пустой массив представляет собой удобную альтернативу значению `null` при возврате из метода. Если метод способен возвращать `null`, прикладной код, в котором выполняется обращение к методу, должен сравнить возвращённое значение с `null` прежде, чем перейти к выполнению оставшихся операций. Если же метод возвращает массив (возможно, пустой), никакие дополнительные проверки не нужны – разумеется, помимо тех случаев, которые касаются длины массива и должны выполняться в любом случае.

Допускается и иная форма объявления массива, в которой квадратные скобки задаются после идентификатора массива, а не после наименования его типа:

```
int ia[] = new int[3];
```

Первый вариант синтаксиса считается более предпочтительным, поскольку описание типа в таком случае выглядит более компактным.

Правила употребления в объявлениях массивов тех или иных модификаторов обычны. Существует единственная особенность, которую важно помнить, - модификаторы применяются к массиву как таковому, но не к его отдельным элементам. Если в объявлении массива указан признак `final`, это значит только то, что ссылка на массив не может быть изменена после его создания, но никак не запрещает возможность изменения содержимого отдельных элементов массива. Язык не позволяет задавать никаких модификаторов (скажем, `final` или `volatile`) для элементов массива.

Многомерные массивы. В Java поддерживается возможность объявления многомерных массивов (multidimensional arrays), т.е. массивов, элементами которых служат другие массивы. Код, предусматривающий объявление двумерной матрицы и вывод на экран содержимого её элементов, может выглядеть, например, так:

```
float[][] mat = new float [4][4];
for (int y = 0; y < mat.length; y++) {
    for (int x = 0; x < mat[y].length; x++)
        System.out.print(mat[y][x] + " ");
    System.out.println();
}
```

При создании массива должна быть указана, по меньшей мере, его первая, «самая левая», размерность. Другие размерности разрешается не задавать – в этом случае их придётся определить позже. Указание в операторе new одновременно всех размерностей – это самый лаконичный способ создания массива, позволяющий избежать необходимости использования дополнительных операторов new. Выражение объявления и создания массива mat, приведённое выше, равнозначно следующему фрагменту кода:

```
float[][] mat = new float [4][];
for (int y = 0; y < mat.length; y++)
    mat[y] = new float[4];
```

Такая форма объявления обладает тем преимуществом, что позволяет наряду с получением массивов с одинаковыми размерностями (скажем, 4x4) строить и массивы массивов различных размерностей, необходимых для хранения тех или иных последовательностей данных.

Инициализация массивов. При создании массива каждый его элемент получает значение, предусмотренное по умолчанию и зависящее от типа массива: нуль – для числовых типов, '\u0000' – для char, false – для boolean и null – для ссылочных типов. Объявляя массив ссылочного типа, мы на самом деле определяем массив переменных этого типа. Рассмотрим следующий фрагмент кода:

```
Attr[] attrs = new Attr [12];
for (int i = 0; i < attrs.length; i++)
    attrs[i] = new Attr(names[i], values[i]);
```

После выполнения первого выражения, содержащего оператор new, переменная attrs получит ссылку на массив из 12 переменных, которые инициализированы значением null. Объекты Attr как таковые, будут созданы только в процессе прохождения цикла. Массив может инициализироваться (одновременно с объявлением) посредством конструкции в фигурных скобках, в которой перечислены исходные значения его элементов:

```
String[] dangers = {"Львы", "Тигры", "Медведи"};
```

Следующий фрагмент кода даст тот же результат:

```
String[] dangers = new String[3];
dangers[0] = "Львы";
dangers[1] = "Тигры";
dangers[2] = "Медведи";
```

Первая форма, предусматривающая задание списка инициализаторов в фигурных скобках, не требует явного использования оператора new – Он вызывается косвенно исполняющей системой. Длина массива в этом случае определяется количеством значений – инициализаторов. Допускается и возможность явного задания оператора new, Но размерность всё равно следует опускать – она, как и раньше, определяется исполняющей системой:

```
String[] dangers = new String[] {"Львы", "Тигры", "Медведи"};
```

Подобную форму объявления и инициализации массива разрешается применять в любом месте кода, например, в выражении вызова метода:

```
printStrings(new String[] {"раз", "два", "три"});
```

Массив без названия (т.е. без явно сохранённой ссылки), который создаётся таким образом, называют анонимным (anonymous).

Массивы массивов могут инициализироваться посредством вложенных последовательностей исходных значений. Ниже приведён пример объявления массива, содержащего несколько первых строк так называемого треугольника Паскаля, где каждая строка описана собственным массивом значений.

```
int[][] pascalsTriangle = {
    {1};
    {1,1};
    {1,2,1};
    {1,3,3,1};
    {1,4,6,4,1};
};
```

Индексы многомерных массивов следуют в порядке от внешнего к внутренним. Так, например, `pascalsTriangle[0]` ссылается на массив значений типа `int`, содержащий один элемент, `pascalsTriangle[1]` указывает на массив с двумя элементами и т.д.

Класс `System` содержит метод `arrayscopy`, который позволяет присваивать значения элементов одного массива другому, избегая необходимости использования вложенных циклов.

3.13 Операторы

Операторы в языке Java - это специальные символы, которые сообщают транслятору о том, что вы хотите выполнить операцию с некоторыми операндами. Некоторые операторы требуют одного операнда, их называют унарными. Одни операторы ставятся перед операндами и называются префиксными, другие - после, их называют постфиксными операторами. Большинство же операторов ставят между двумя операндами, такие операторы называются инфиксными бинарными операторами. Существует тернарный оператор, работающий с тремя операндами [3].

В Java имеется 46 операторов. Их можно разбить на 4 класса - арифметические, битовые, операторы сравнения и логические.

Арифметические операторы. Арифметические операторы используются для вычислений так же как в алгебре (см. таблицу 3.8). Допустимые операнды должны иметь числовые типы. Например, использовать эти операторы для работы с логическими типами нельзя, а для работы с типом `char` можно, поскольку в Java тип `char` - это подмножество типа `int`.

Таблица 3.8 – Арифметические операторы

Оператор	Результат	Оператор	Результат
+	Сложение	+ =	Сложение с присваиванием
-	Вычитание (также унарный минус)	-=	Вычитание с присваиванием
*	Умножение	*=	Умножение с присваиванием
/	Деление	/=	Деление с присваиванием
%	Деление по модулю	%=	Деление по модулю с присваиванием
++	Инкремент	--	Декремент

Четыре арифметических действия. Ниже, в качестве примера, приведена простая программа, демонстрирующая использование операторов. Обратите внимание на то, что операторы работают как с целыми литералами, так и с переменными.

```

class BasicMath {
    public static void main(String args[]) {
        int a = 1 + 1;
        int b = a * 3;
        int c = b / 4;
        int d = b - a;
        int e = -d;
        System.out.println("a = " + a);
        System.out.println("b = " + b);
        System.out.println("c = " + c);
        System.out.println("d = " + d);
        System.out.println("e = " + e);
    }
}

```

Исполнив эту программу, вы должны получить приведенный ниже результат:

```

a = 2
b = 6
c = 1
d = 4
e = -4

```

Оператор деления по модулю. Оператор деления по модулю, или оператор `mod`, обозначается символом `%`. Этот оператор возвращает остаток от деления первого операнда на второй. В отличие от C++, функция `mod` в Java работает не только с целыми, но и с вещественными типами. Приведенная ниже программа иллюстрирует работу этого оператора.

```

class Modulus {
    public static void main (String[] args) {
        int x = 42;
        double y = 42.3;
        System.out.println("x mod 10 = " + x % 10);
        System.out.println("y mod 10 = " + y % 10);
    }
}

```

Выполнив эту программу, вы получите следующий результат:

```

x mod 10 = 2
y mod 10 = 2.3

```

Арифметические операторы присваивания. Для каждого из арифметических операторов есть форма, в которой одновременно с заданной операцией выполняется присваивание. Ниже приведен пример, который иллюстрирует использование подобной разновидности операторов.

```

class OpEquals {
    public static void main(String[] args) {
        int a = 1;
        int b = 2;
        int c = 3;
        a += 5;
        b *= 4;
        c += a * b;
        c %= 6;
        System.out.println("a = " + a);
        System.out.println("b = " + b);
        System.out.println("c = " + c);
    }
}

```

А вот и результат, полученный при запуске этой программы:

```

a = 6
b = 8
c = 3

```

Целочисленная арифметика. Арифметические операции с целочисленными операндами выполняются в соответствии с правилами дополнения по модулю, т.е. если

значение выходит за границы допустимого диапазона изменения числовых величин соответствующего типа (например, `int` или `long`), оно уменьшается по модулю, отвечающему размеру этого диапазона. Поэтому результаты целочисленных операций никогда не приводят к переполнению разрядной сетки (overflow) переменной или исчезновению значащих разрядов (потере значимости - underflow) в ней – разряды числа могут быть только перенесены.

При целочисленном делении остаток отбрасывается (например, $7/2$ равно 3 и $(-7)/2$ равно -3). Результаты применения операторов деления и вычисления остатка по отношению к аргументам целых типов подчиняются следующему правилу:

$$(x/y)*y + x\%y == x$$

Так что, $7\%2$ есть 1, а $-7\%2$ есть -1. Если в качестве операнда-делителя в выражениях целочисленного деления и вычисления остатка задаётся нуль, операция считается недопустимой и генерируется исключение типа `ArithmeticException`.

Арифметика с плавающей запятой. В процессе выполнения арифметических операций над аргументами с плавающей запятой возможны переполнение разрядной сетки до бесконечности (результат превышает верхнюю границу диапазона изменения значений типа `float` или `double`) и потеря значимости – нередко до нуля (результат слишком мал для типа `float` или `double`). В результате вычисления некорректных выражений (таких как, например, деление бесконечности на бесконечность) получается значение NaN – «не число» (not a number).

Арифметические операции над конечными операндами с плавающей запятой, удовлетворяющими диапазонам точности представления значений `float` и `double`, выполняется вполне предсказуемо. Правило присваивания знака результату также традиционно: при умножении и делении чисел с одним знаком результат положителен, а если знаки различны – отрицателен.

Сложение двух бесконечностей даёт в результате такую же бесконечность, если их знаки одинаковы, и NaN – если знаки различны. При вычитании бесконечностей одного знака будет получено значение NaN; вычитание бесконечностей с разными знаками даёт в результате бесконечность с тем знаком, который имеется у левого операнда. Например, $(\infty - (-\infty))$ есть ∞ . Результат вычисления арифметического выражения, одним из операндов которого является NaN, всегда равен NaN. Переполнение даёт в итоге бесконечность соответствующего знака, а потеря значимости – значение (возможно, нулевое) соответствующего знака. При выполнении арифметических операций с плавающей запятой поддерживается отрицательный нуль, -0.0 , который в контексте операторов сравнения равен положительному нулю, $+0.0$. Хотя оба нуля считаются равноценными, в конкретных выражениях они способны приводить к различным результатам. Так, например, результат вычисления выражения $1f/0f$ – равен положительной бесконечности, а выражения $1f/-0f$ – отрицательной бесконечности.

Если итогом исчезновения значащих разрядов является -0.0 и если $-0.0 == 0.0$, каким образом можно выявить факт получения отрицательного нуля? Следует поместить тестируемое нулевое значение в выражение, где знак способен себя проявить, и проверить результат. Пусть, например, `x` содержит нулевое значение; тогда выражение $1/x$ будет равно отрицательной бесконечности, если `x` - отрицательный нуль, и положительной бесконечности – в противном случае.

Правила выполнения операций с бесконечными величинами совпадают с теми, которые приняты в математике. Алгебраическое сложение любой конечной величины с бесконечностью даёт в результате ту же бесконечность. Например, $(-\infty + x)$ равно $-\infty$ при любом конечном значении `x`.

Бесконечные значения в Java-программе задаются с помощью символических констант `POSITIVE_INFINITY` (положительная бесконечность) и `NEGATIVE_INFINITY` (отрицательная бесконечность), объявленных в классах-оболочках `Float` и `Double`.

Например, Double. NEGATIVE_INFINITY указывает на версию значения отрицательной бесконечности, определённую в классе Double.

Умножение бесконечности на нуль даёт в результате NaN. При умножении бесконечности на ненулевое конечное значение будет получена бесконечность соответствующего знака.

Операции деления и вычисления остатка в применении к аргументам с плавающей запятой способны давать в результате бесконечные значения или NaN, но никогда не приводят к выбрасыванию исключений. Результаты операций деления и вычисления остатка при различных комбинациях значений аргументов представлены в таблице 3.9.

Таблица 3.9

x	y	x/y	x%y
Конечное значение	± 0.0	$\pm\infty$	NaN
Конечное значение	$\pm\infty$	± 0.0	x
± 0.0	± 0.0	NaN	NaN
$\pm\infty$	Конечное значение	$\pm\infty$	NaN
$\pm\infty$	$\pm\infty$	NaN	NaN

В остальных случаях оператор вычисления остатка (%) от деления аргументов с плавающей запятой действует аналогично случаю, когда он применяется к целочисленным аргументам.

Инкремент и декремент. В C существует 2 оператора, называемых операторами инкремента и декремента (++ и --) и являющихся сокращённым вариантом записи для сложения или вычитания из операнда единицы. Эти операторы уникальны в том плане, что могут использоваться как в префиксной, так и в постфиксной форме. Следующий пример иллюстрирует использование операторов инкремента и декремента.

```
class IncDec {
    public static void main(String[] args) {
        int a = 1;
        int b = 2;
        int c = ++b;
        int d = a++;
        c++;
        System.out.println("a = " + a);
        System.out.println("b = " + b);
        System.out.println("c = " + c);
        System.out.println("d = " + d);
    }
}
```

Результат выполнения данной программы будет таким:

```
a = 2
b = 3
c = 4
d = 1
```

Целочисленные битовые операторы. Для целых числовых типов данных - long, int, short, char и byte, определен дополнительный набор операторов, с помощью которых можно проверять и модифицировать состояние отдельных битов соответствующих значений. В таблице 10 приведена сводка таких операторов. Операторы битовой арифметики работают с каждым битом как с самостоятельной величиной.

Таблица 3.10 – Битовые операторы

Оператор	Результат	Оператор	Результат
~	Побитовое унарное отрицание (NOT)		
&	Побитовое И (AND)	&=	Побитовое И (AND) с присваиванием
	Побитовое ИЛИ (OR)	=	Побитовое ИЛИ (OR) с присваиванием
^	Побитовое исключающее ИЛИ (XOR)	^=	Побитовое исключающее ИЛИ (XOR) с присваиванием
>>	Сдвиг вправо	>> =	Сдвиг вправо с присваиванием
>>>	Сдвиг вправо с заполнением нулями	>>>=	Сдвиг вправо с заполнением нулями с присваиванием
<<	сдвиг влево	<<=	Сдвиг влево с присваиванием

В таблице 3.11, приведенной ниже, показано, как каждый из операторов битовой арифметики воздействует на возможные комбинации битов своих операндов. Приведенный после таблицы пример иллюстрирует использование этих операторов в программе на языке Java.

Таблица 3.11

A	B	OR	AND	XOR	NOT A
0	0	0	0	0	1
1	0	1	0	1	0
0	1	1	0	1	1
1	1	1	1	0	0

```

class Bitlogic {
    public static void main(String[] args) {
        String binary[] = { "0000", "0001", "0010",
                            "0011", "0100", "0101",
                            "0110", "0111", "1000",
                            "1001", "1010", "1011",
                            "1100", "1101", "1110",
                            "1111" };

        int a = 3;    // 0+2+1  или двоичное 0011
        int b = 6;    // 4+2+0  или двоичное 0110
        int c = a | b;
        int d = a & b;
        int e = a ^ b;
        int f = (~a & b) | (a & ~b);
    }
}

```

```

        int g = ~a & 0x0f;
        System.out.println(" a = " + binary[a]);
        System.out.println(" b = " + binary[b]);
        System.out.println("a | b = " + binary[c]);
        System.out.println("a & b = " + binary[d]);
        System.out.println("a ^ b = " + binary[e]);
        System.out.println("~a & b | a & ~b = " + binary[f]);
        System.out.println("~a = " + binary[g]);
    }
}

```

Ниже приведен результат, полученный при выполнении этой программы:

```

a = 0011
b = 0110
a | b = 0111
a & b = 0010
a ^ b = 0101
~a & b | a & ~b = 0101
~a = 1100

```

Сдвиги влево и вправо. Оператор << выполняет сдвиг влево всех битов своего левого операнда на число позиций, заданное правым операндом. При этом часть битов в левых разрядах выходит за границы и теряется, а соответствующие правые позиции заполняются нулями. Ранее уже говорилось об автоматическом повышении типа всего выражения до int в том случае, если в выражении присутствуют операнды типа int или целых типов меньшего размера. Если же хотя бы один из операндов в выражении имеет тип long, то и тип всего выражения повышается до long.

Оператор >> означает в языке Java сдвиг вправо. Он перемещает все биты своего левого операнда вправо на число позиций, заданное правым операндом. Когда биты левого операнда выдвигаются за самую правую позицию слова, они теряются. При сдвиге вправо освобождающиеся старшие (левые) разряды сдвигаемого числа заполняются предыдущим содержимым знакового разряда. Такое поведение называют расширением знакового разряда.

В следующей программе байтовое значение преобразуется в строку, содержащую его шестнадцатеричное представление. Обратите внимание - сдвинутое значение приходится маскировать, то есть логически умножать на значение 0x0f, для того, чтобы очистить заполняемые в результате расширения знака биты и понизить значение до пределов, допустимых при индексировании массива шестнадцатеричных цифр.

```

class HexByte {
    static public void main(String[] args) {
        char hex[] = { '0', '1', '2', '3', '4', '5', '6', '7', '8',
            '9', 'a', 'b', 'c', 'd', 'e', 'f' };
        byte b = (byte) 0xf1;
        System.out.println("b = 0x" + hex[(b >> 4) & 0x0f] + hex[b &
            0x0f]);
    }
}

```

Ниже приведен результат работы этой программы:

```
b = 0xf1
```

Беззнаковый сдвиг вправо. Часто требуется, чтобы при сдвиге вправо расширение знакового разряда не происходило, а освобождающиеся левые разряды просто заполнялись бы нулями.

```

class ByteUShift {
    static public void main(String[] args) {
        char hex[] = { '0', '1', '2', '3', '4', '5', '6', '7', '8',
            '9', 'a', 'b', 'c', 'd', 'e', 'f' };
        byte b = (byte) 0xf1;
        byte c = (byte) (b >> 4);
        byte d = (byte) (b >> 4);
        byte e = (byte) ((b & 0xff) >> 4);
    }
}

```

```

        System.out.println(" b = 0x" + hex[(b >> 4) & 0x0f] + hex[b &
0x0f]);
        System.out.println(" b >> 4 = 0x" + hex[(c >> 4) & 0x0f] +
        hex[c & 0x0f]);
        System.out.println("b >>> 4 = 0x" + hex[(d >> 4) & 0x0f] +
        hex[d & 0x0f]);
        System.out.println("(b & 0xff) >> 4 = 0x" + hex[(e >> 4) &
0x0f] +
        hex[e & 0x0f]);
    }
}

```

Для этого примера переменную `b` можно было бы инициализировать произвольным отрицательным числом, мы использовали число с шестнадцатеричным представлением `0xf1`. Переменной `c` присваивается результат знакового сдвига `b` вправо на 4 разряда. Как и ожидалось, расширение знакового разряда приводит к тому, что `0xf1` превращается в `0xff`. Затем в переменную `d` заносится результат беззнакового сдвига `b` вправо на 4 разряда. Можно было бы ожидать, что в результате `d` содержит `0x0f`, однако на деле мы снова получаем `0xff`. Это - результат расширения знакового разряда, выполненного при автоматическом повышении типа переменной `b` до `int` перед операцией сдвига вправо. Наконец, в выражении для переменной `e` нам удастся добиться желаемого результата - значения `0x0f`. Для этого нам пришлось перед сдвигом вправо логически умножить значение переменной `b` на маску `0xff`, очистив таким образом старшие разряды, заполненные при автоматическом повышении типа. Обратите внимание, что при этом уже нет необходимости использовать беззнаковый сдвиг вправо, поскольку мы знаем состояние знакового бита после операции AND.

```

b = 0xf1
b >> 4 = 0xff
b >>> 4 = 0xff
b & 0xff) >> 4 = 0x0f

```

Битовые операторы присваивания. Так же, как и в случае арифметических операторов, у всех бинарных битовых операторов есть родственная форма, позволяющая автоматически присваивать результат операции левому операнду. В следующем примере создаются несколько целых переменных, с которыми с помощью операторов, указанных выше, выполняются различные операции.

```

class OpBitEquals {
    public static void main(String[] args) {
        int a = 1;
        int b = 2;
        int c = 3;
        a |= 4;
        b >>= 1;
        c <<= 1;
        a ^= c;
        System.out.println("a = " + a);
        System.out.println("b = " + b);
        System.out.println("c = " + c);
    }
}

```

Результаты исполнения программы таковы:

```

a = 3
b = 1
c = 6

```

Операторы отношения. Для того, чтобы можно было сравнивать два значения, в Java имеется набор операторов, описывающих отношение и равенство. Список таких операторов приведен в таблице 3.12.

Таблица 3.12 – Операторы отношений

Оператор	Результат
==	равно
!=	не равно
>	больше
<	меньше
>=	больше или равно
<=	меньше или равно

Значения любых типов, включая целые и вещественные числа, символы, логические значения и ссылки, можно сравнивать, используя оператор проверки на равенство == и неравенство !=. Обратите внимание - в языке Java, так же, как в C и C++ проверка на равенство обозначается последовательностью (==). Один знак (=) - это оператор присваивания.

Булевы логические операторы. Булевы логические операторы, сводка которых приведена в таблице 3.13, оперируют только с операндами типа boolean. Все бинарные логические операторы воспринимают в качестве операндов два значения типа boolean и возвращают результат того же типа.

Таблица 3.13 – Булевы логические операторы

Оператор	Результат	Оператор	Результат
&	Логическое И (AND)	&=	И (AND) с присваиванием
	Логическое ИЛИ (OR)	=	ИЛИ (OR) с присваиванием
^	Логическое исключающее ИЛИ (XOR)	^=	Исключающее ИЛИ (XOR) с присваиванием
	Оператор OR быстрой оценки выражений (short circuit OR)	==	Равно
&&	Оператор AND быстрой оценки выражений (short circuit AND)	!=	Не равно
!	Логическое унарное отрицание (NOT)	?:	Тернарный оператор if-then-else

Результаты воздействия логических операторов на различные комбинации значений операндов показаны в таблице 3.14.

Таблица 3.14

A	B	OR	AND	XOR	NOT A
false	false	false	false	false	true
true	false	true	false	true	false

false	true	true	false	true	true
true	true	true	true	false	false

Программа, приведенная ниже, практически полностью повторяет уже знакомый вам пример BitLogic. Только на этот раз мы работаем с булевыми логическими значениями.

```
class BoolLogic {
    public static void main(String[] args) {
        boolean a = true;
        boolean b = false;
        boolean c = a | b;
        boolean d = a & b;
        boolean e = a ^ b;
        boolean f = (!a & b) | (a & !b);
        boolean g = !a;
        System.out.println(" a = " + a);
        System.out.println(" b = " + b);
        System.out.println(" a|b = " + c);
        System.out.println(" a&b = " + d);
        System.out.println(" a^b = " + e);
        System.out.println("!a&b|a&!b = " + f);
        System.out.println(" !a = " + g);
    }
}
```

Результаты исполнения программы таковы:

```
a = true
b = false
a|b = true
a&b = false
a^b = true
!a&b|a&!b = true
!a = false
```

Операторы быстрой оценки логических выражений (short circuit logical operators). Существуют два интересных дополнения к набору логических операторов. Это альтернативные версии операторов AND и OR, служащие для быстрой оценки логических выражений. Вы знаете, что если первый операнд оператора OR имеет значение true, то независимо от значения второго операнда результатом операции будет величина true. Аналогично, в случае оператора AND, если первый операнд - false, то значение второго операнда на результат не влияет - он всегда будет равен false. Если вы используете операторы && и || вместо обычных форм & и |, то Java не производит оценку правого операнда логического выражения, если ответ ясен из значения левого операнда. Общепринятой практикой является использование операторов && и || практически во всех случаях оценки булевых логических выражений. Версии этих операторов & и | применяются только в битовой арифметике.

Оператор instanceof. Оператор instanceof проверяет, принадлежит ли объект некоторому типу. В левой части выражения instanceof указывается ссылка, а в правой – имя класса или интерфейса. Оператор instanceof возвращает true, если выражение левой части совместимо с типом, название которого указано в правой части, и false – в противном случае.

Следует заметить, что данный оператор не позволяет выяснить реальный класс, экземпляром которого является объект. С применением оператора instanceof это возможно только если заранее известна иерархия классов, но код программы при этом будет несколько специфичен.

Тернарный оператор if-then-else. Общая форма оператора if-then-else такова:
выражение1? выражение2: выражение3

В качестве первого операнда - <выражение1> - может быть использовано любое выражение, результатом которого является значение типа boolean. Если результат равен true, то выполняется оператор, заданный вторым операндом, то есть, <выражение2>. Если же первый операнд равен false, то выполняется третий операнд - <выражение3>. Второй и третий операнды, то есть <выражение2> и <выражение3>, должны возвращать значения совместимых типов и не должны иметь тип void.

В приведенной ниже программе этот оператор используется для проверки делителя перед выполнением операции деления. В случае нулевого делителя возвращается значение 0.

```
class Ternary {
    public static void main(String[] args) {
        int a = 42;
        int b = 2;
        int c = 99;
        int d = 0;
        int e = (b == 0) ? 0 : (a / b);
        int f = (d == 0) ? 0 : (c / d);
        System.out.println("a = " + a);
        System.out.println("b = " + b);
        System.out.println("c = " + c);
        System.out.println("d = " + d);
        System.out.println("a / b = " + e);
        System.out.println("c / d = " + f);
    }
}
```

При выполнении этой программы исключительной ситуации деления на нуль не возникает и выводятся следующие результаты:

```
a = 42
b = 2
c = 99
d = 0
a / b = 21
c / d = 0
```

Приоритеты операторов. В Java действует определенный порядок, или приоритет, операций. В элементарной алгебре вас учили тому, что у умножения и деления более высокий приоритет, чем у сложения и вычитания. В программировании также приходится следить и за приоритетами операций. В таблице 3.15 указаны в порядке убывания приоритеты всех операций языка Java. Операторы, обладающие одинаковым приоритетом, указаны в пределах одной строки.

Таблица 3.15 - Приоритеты операторов

Постфиксные операторы	[]	.	(params)	expr+ +	expr-- -							
Унарные операторы	++expr --expr	--expr	+expr	-expr	~	!						
Операторы создания объектов и преобразования	new	(type) expr										

типов												
Операторы умножения, деления и вычисления остатка	*	/	%									
Операторы сложения и вычитания	+	-										
Операторы побитового сдвига	<<	>>	>>>									
Операторы сравнения	<	>	>=	<=	instanceof							
Операторы равенства и неравенства	==	!=										
И (AND)	&											
Исключающее ИЛИ (XOR)	^											
Включающее ИЛИ (OR)												
Условное И (AND)	&&											
Условное ИЛИ (OR)												
Условные	?:											

й оператор												
Оператор ы присваив ания	=	+=	-=	*=	/=	%=	>> =	<< =	>>> =	&=	^=	=

В первой строке таблицы приведены три оператора, о которых мы пока не говорили. Круглые скобки () используются для явной установки приоритета. Квадратные скобки [] используются для индексирования переменной-массива. Оператор . (точка) используется для выделения элементов из ссылки на объект. Все же остальные операторы уже обсуждались ранее.

Явные приоритеты. Поскольку высший приоритет имеют круглые скобки, вы всегда можете добавить в выражение несколько пар скобок, если у вас есть сомнения по поводу порядка вычислений или вам просто хочется сделать свой код более читабельным.

`a >> b + 3`

Какому из двух выражений, `a >> (b + 3)` или `(a >> b) + 3`, соответствует первая строка? Поскольку у оператора сложения более высокий приоритет, чем у оператора сдвига, правильный ответ - `a >> (b + a)`. Так что если вам требуется выполнить операцию `(a >> b) + 3` без скобок не обойтись.

3.14 Управление выполнением программы

Управление в Java почти идентично средствам, используемым в C и C++ [3].

Условные операторы:

If-else. В обобщенной форме этот оператор записывается следующим образом:

`if (логическое выражение) оператор1; [ else оператор2;]`

Раздел `else` необязателен. На месте любого из операторов может стоять составной оператор, заключенный в фигурные скобки. Логическое выражение - это любое выражение, возвращающее значение типа `boolean`.

Ниже приведена программа, в которой для определения, к какому времени года относится тот или иной месяц, используются операторы `if-else`.

```
class IfElse {
    public static void main(String[] args) {
        int month = 4;
        String season;
        if (month == 12 || month == 1 || month == 2) {
            season = "Winter";
        }
        else if (month == 3 || month == 4 || month == 5) {
            season = "Spring";
        }
        else if (month == 6 || month == 7 || month == 8) {
            season = "Summer";
        }
        else if (month == 9 || month == 10 || month == 11) {
            season = "Autumn";
        }
        else {season = "Bogus Month";}
        System.out.println( "April is in the " + season + ".");
    }
}
```

После выполнения программы вы должны получить следующий результат:

`April is in the Spring.`

Break. В языке Java отсутствует оператор `goto`. Для того, чтобы в некоторых случаях заменять `goto`, в Java предусмотрен оператор `break`. Этот оператор сообщает

исполняющей среде, что следует прекратить выполнение блока и передать управление оператору, следующему за данным блоком. Для именования блоков в языке Java используются метки. Оператор break при работе с циклами и в операторах switch может использоваться без метки. В таком случае подразумевается выход из текущего блока.

Например, в следующей программе имеется три вложенных блока, и у каждого своя уникальная метка. Оператор break, стоящий во внутреннем блоке, вызывает переход на оператор, следующий за блоком b. При этом пропускаются два оператора println.

```
class Break {
    public static void main(String[] args) {
        boolean t = true;
        a:      { b:      { c:      {
            System.out.println("Before the break"); // Перед break
            if (t)
                break b;
            System.out.println("This won't execute");
            // Не будет выполнено
        }
        System.out.println("This won't execute");
        // Не будет выполнено
    }
    System.out.println("This is after b");
    //После b
}
}
```

В результате исполнения программы вы получите следующий результат:

```
Before the break
This is after b
```

Switch. Оператор switch обеспечивает ясный способ переключения между различными частями программного кода в зависимости от значения одной переменной или выражения. Общая форма этого оператора такова:

```
switch ( выражение ) { case значение1:
    break;
case значение2:
    break;
case значение m:
    break;
default:
}
```

Результатом вычисления выражения может быть значение любого простого типа, при этом каждое из значений, указанных в операторах case, должно быть совместимо по типу с выражением в операторе switch. Все эти значения должны быть уникальными литералами. Если же вы укажете в двух операторах case одинаковые значения, транслятор выдаст сообщение об ошибке.

Если же значению выражения не соответствует ни один из операторов case, управление передается коду, расположенному после ключевого слова default. Отметим, что оператор default необязателен. В случае, когда ни один из операторов case не соответствует значению выражения и в switch отсутствует оператор default выполнение программы продолжается с оператора, следующего за оператором switch.

Внутри оператора switch (а также внутри циклических конструкций, но об этом - позже) break без метки приводит к передаче управления на код, стоящий после оператора switch. Если break отсутствует, после текущего раздела case будет выполняться следующий. Иногда бывает удобно иметь в операторе switch несколько смежных разделов case, не разделенных оператором break.

```
class SwitchSeason {
    public static void main(String[] args) {
        int month = 4;
        String season;
```

```

switch (month) {
    case 12: // проходит дальше
    case 1: // проходит дальше
    case 2:
        season = "Winter";
        break;
    case 3: // проходит дальше
    case 4: // проходит дальше
    case 5:
        season = "Spring";
        break;
    case 6: // проходит дальше
    case 7: // проходит дальше
    case 8:
        season = "Summer";
        break;
    case 9: // проходит дальше
    case 10: // проходит дальше
    case 11:
        season = "Autumn";
        break;
    default:
        season = "Bogus Month";
}
System.out.println("April is in the " + season + ".");
}
}

```

Ниже приведен еще более полезный пример, где оператор switch используется для передачи управления в соответствии с различными кодами символов во входной строке. Программа подсчитывает число строк, слов и символов в текстовой строке.

```

class WordCount {
    static String text = "Now is the time\n" +
        "for all good men\n" +
        "to come to the aid\n" +
        "of their country\n" +
        "and pay their due taxes\n";
    static int len = text.length();
    public static void main(String[] args) {
        boolean inWord = false;
        int numChars = 0;
        int numWords = 0;
        int numLines = 0;
        for (int i=0; i < len; i++) {
            char c = text.charAt(i);
            numChars++;
            switch (c) {
                case '\n': numLines++; // проходит дальше
                case '\t': // проходит дальше
                case ' ': if (inWord) {
                    numWords++;
                    inWord = false;
                }
                break;
                default: inWord = true;
            }
        }
        System.out.println("\t" + numLines + "\t" +
            numWords + "\t" + numChars);
    }
}

```

В этой программе для подсчета слов использовано несколько концепций, относящихся к обработке строк. В результате исполнения программы вы получите следующий результат:

5 21 94

Return. Подпрограмма `main`, которую мы использовали до сих пор - это статический метод соответствующего класса-примера. В любом месте программного кода метода можно поставить оператор `return`, который приведет к немедленному завершению работы и передаче управления коду, вызвавшему этот метод. Ниже приведен пример, иллюстрирующий использование оператора `return` для немедленного возврата управления, в данном случае - исполняющей среде Java.

```
class ReturnDemo {
    public static void main(String[] args) {
        boolean t = true;
        System.out.println("Before the return");
        //Перед оператором return
        if (t) return;
        System.out.println("This won't execute");
        //Это не будет выполнено
    }
}
```

Замечание. Зачем в этом примере использован оператор `if (t)`? Дело в том, не будь этого оператора, транслятор Java догадался бы, что последний оператор `println` никогда не будет выполнен. Такие случаи в Java считаются ошибками, поэтому без оператора `if` оттранслировать этот пример нам бы не удалось.

Оператор запятая. Иногда возникают ситуации, когда разделы инициализации или итерации цикла `for` требуют нескольких операторов. Поскольку составной оператор в фигурных скобках в заголовок цикла `for` вставлять нельзя, Java предоставляет альтернативный путь. Применение запятой (,) для разделения нескольких операторов допускается только внутри круглых скобок оператора `for`. Ниже приведен тривиальный пример цикла `for`, в котором в разделах инициализации и итерации стоит несколько операторов.

```
class Comma {
    public static void main(String[] args) {
        int a, b;
        for (a = 1, b = 4; a < b; a++, b--) {
            System.out.println("a = " + a);
            System.out.println("b = " + b);
        }
    }
}
```

Вывод этой программы показывает, что цикл выполняется всего два раза.

```
a = 1
b = 4
a = 2
b = 3
```

Continue. В некоторых ситуациях возникает потребность досрочно перейти к выполнению следующей итерации, проигнорировав часть операторов тела цикла, еще не выполненных в текущей итерации. Для этой цели в Java предусмотрен оператор `continue`. Ниже приведен пример, в котором оператор `continue` используется для того, чтобы в каждой строке печатались два числа.

```
class ContinueDemo {
    public static void main(String args[]) {
        for (int i=0; i < 10; i++) {
            System.out.print(i + " ");
            if (i % 2 == 0) continue;
            System.out.println("");
        }
    }
}
```

```
    }
}
```

Если индекс четный, цикл продолжается без вывода символа новой строки. Результат выполнения этой программы таков:

```
0 1
2 3
4 5
6 7
8 9
```

Как и в случае оператора `break`, в операторе `continue` можно задавать метку, указывающую, в каком из вложенных циклов вы хотите досрочно прекратить выполнение текущей итерации. Для иллюстрации служит программа, использующая оператор `continue` с меткой для вывода треугольной таблицы умножения для чисел от 0 до 9:

```
class ContinueLabel {
    public static void main(String[] args) {
        outer: for (int i=0; i < 10; i++) {
            for (int j = 0; j < 10; j++) {
                if (j > i) {
                    System.out.println("");
                    continue outer;
                }
                System.out.print(" " + (i * j));
            }
        }
    }
}
```

Оператор `continue` в этой программе приводит к завершению внутреннего цикла со счетчиком `j` и переходу к очередной итерации внешнего цикла со счетчиком `i`. В процессе работы эта программа выводит следующие строки:

```
0
0 1
0 2 4
0 3 6 9
0 4 8 12 16
0 5 10 15 20 25
0 6 12 18 24 30 36
0 7 14 21 28 35 42 49
0 8 16 24 32 40 48 56 64
0 9 18 27 36 45 54 63 72 81
```

3.15 Циклы

Любой цикл можно разделить на 4 части - инициализацию, тело, итерацию и условие завершения. В Java есть три циклические конструкции: `while` (с предусловием), `do-while` (с постусловием) и `for` (с параметром) [3].

While. Этот цикл многократно выполняется до тех пор, пока значение логического выражения равно `true`. Ниже приведена общая форма оператора `while`:

```
[ инициализация; ]
while ( завершение ) {
    тело;
    [итерация;] }
```

Инициализация и итерация необязательны. Ниже приведен пример цикла `while` для печати десяти строк.

```
class WhileDemo {
    public static void main(String[] args) {
        int n = 10;
        while (n > 0) {
            System.out.println("tick " + n);
            n--;
        }
    }
}
```

```

    }
}

```

Do-while. Иногда возникает потребность выполнить тело цикла, по крайней мере, один раз - даже в том случае, когда логическое выражение с самого начала принимает значение false. Для таких случаев в Java используется циклическая конструкция do-while. Её общая форма записи такова:

```

[ инициализация; ] do { тело; [итерация;] } while
( завершение );

```

В следующем примере тело цикла выполняется до первой проверки условия завершения. Это позволяет совместить код итерации с условием завершения:

```

class DoWhile {
    public static void main(String[] args) {
        int n = 10;
        do {
            System.out.println("tick " + n);
        } while (--n < 0);
    }
}

```

For. В этом операторе предусмотрены места для всех четырех частей цикла. Ниже приведена общая форма оператора записи for.

```

for ( инициализация; завершение; итерация ) тело;

```

Любой цикл, записанный с помощью оператора for, можно записать в виде цикла while, и наоборот. Если начальные условия таковы, что при входе в цикл условие завершения не выполнено, то операторы тела и итерации не выполняются ни одного раза. В канонической форме цикла for происходит увеличение целого значения счетчика с минимального значения до определенного предела.

```

class ForDemo {
    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++)
            System.out.println("i = " + i);
    }
}

```

Следующий пример - вариант программы, ведущей обратный отсчет.

```

class ForTick {
    public static void main(String[] args) {
        for (int n = 10; n > 0; n--)
            System.out.println("tick " + n);
    }
}

```

Обратите внимание - переменные можно объявлять внутри раздела инициализации оператора for. Переменная, объявленная внутри оператора for, действует в пределах этого оператора.

А вот - новая версия примера с временами года, в которой используется оператор for.

```

class Months {
    static String months[] = {
        "January", "February", "March", "April",
        "May", "June", "July", "August", "September",
        "October", "November", "December" };
    static int monthdays[] = { 31, 28, 31, 30, 31,
        30, 31, 31, 30, 31, 30, 31 };
    static String spring = "spring";
    static String summer = "summer";
    static String autumn = "autumn";
    static String winter = "winter";
    static String seasons[] = { winter, winter, spring,
        spring, spring, summer, summer,
        summer, autumn, autumn, autumn,

```

```

        winter };
    public static void main(String[] args) {
        for (int month = 0; month < 12; month++) {
            System.out.println(months[month] + " is a " +
                seasons[month] + " month with " +
                monthdays[month] + " days.");
        }
    }
}

```

При выполнении эта программа выводит следующие строки:

```

January is a winter month with 31 days.
February is a winter month with 28 days.
March is a spring month with 31 days.
April is a spring month with 30 days.
May is a spring month with 31 days.
June is a summer month with 30 days.
July is a summer month with 31 days.
August is a summer month with 31 days.
September is a autumn month with 30 days.
October is a autumn month with 31 days.
November is a autumn month with 30 days.
December a winter month with 31 days.

```

4 Создание приложения Android с помощью Java

4.1 Введение

Приложение Tip Calculator (рис. 4.1) вычисляет и отображает чаевые на основании суммы счета в ресторане. После ввода пользователем суммы счета приложение вычисляет и отображает размер чаевых и величину итогового счета (с учетом чаевых). При этом используются три процентных ставки для расчета чаевых: 10 %, 15 % и 20 %. Пользователь может указать собственную ставку, используемую при расчете чаевых. Для этого нужно переместить ползунок компонента Seekbar, после чего обновляется величина процентной ставки, которая отображается в правой части компонента Seekbar. Мы выбрали величину 18 % в качестве заданной по умолчанию пользовательской процентной ставки для чаевых. Это связано с тем, что многие рестораны используют эту процентную ставку для компаний, состоящих из шести человек и более. Предполагаемые величины чаевых и общего счета могут быть изменены пользователем [1].

ПРИМЕЧАНИЕ. Цифровая клавиатура, показанная на рис. 4.1, может отличаться от вашей в зависимости от используемого устройства AVD либо версии Android, установленной на устройстве.

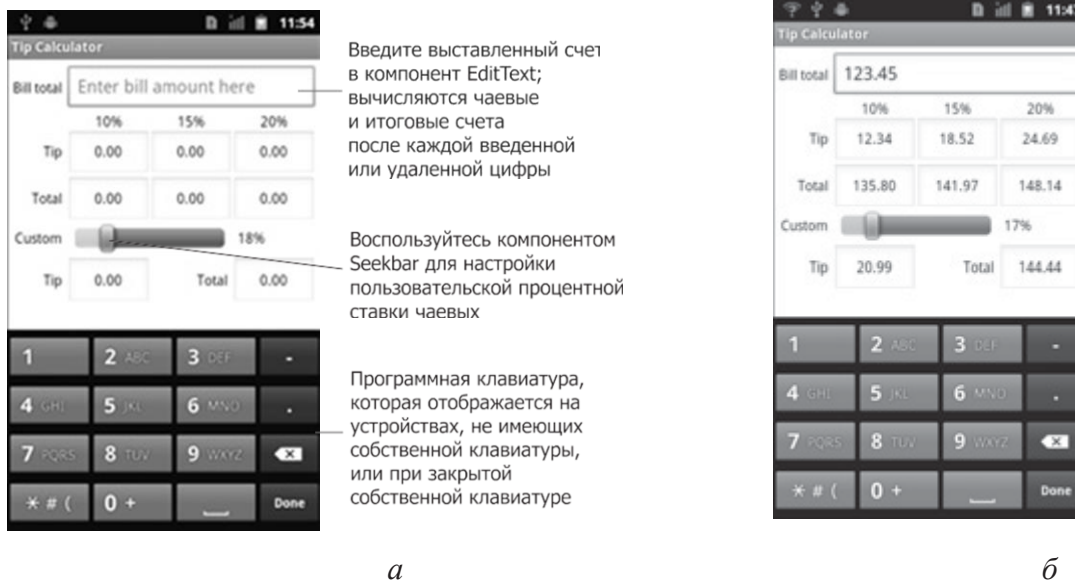


Рис. 4.1. Ввод суммы счета и подсчет чаевых: а — начальный интерфейс пользователя, появляющийся после касания пользователем компонента EditText Bill total и отображения цифровой клавиатуры; б — графический интерфейс после ввода выставленного счета на \$123,45 и изменения процентной ставки чаевых на 17 %.

Начнем с тестирования приложения, заключающегося в вычислении стандартной и заданной пользователем ставок чаевых. Затем вас ожидает обзор технологий, применяемых для создания приложения. После этого мы создадим графический интерфейс пользователя (GUI) с помощью окна Outline в Eclipse. Это окно будет использоваться для добавления компонентов GUI. В графическом редакторе Visual Layout Editor мы увидим, как выглядит будущий GUI. Большая часть XML-кода для этого GUI будет сгенерирована с помощью инструментов плагина ADT. Потом мы отредактируем XML-код, что позволит настроить свойства, недоступные в окне Properties. И напоследок рассмотрим окончательный код приложения и проведем его подробный анализ.

Откройте интегрированную среду разработки Eclipse, импортируйте проект Tip Calculator и выполните следующие действия:

1. Откройте диалоговое окно Import (Импорт). Чтобы открыть диалоговое окно Import,

выполните команды File<Import... (Файл<Импорт...).

2. *Импортируйте проект приложения* Tip Calculator. В диалоговом окне Import раскройте узел General (Общие), выберите параметр Existing Projects into Workspace (Существующие проекты в рабочую область), а потом щелкните на кнопке Next> (Далее>) для выполнения шага Import Projects (Импорт проектов). Установите флажок Select root directory (Выбрать корневой каталог) и щелкните на кнопке Browse... (Просмотр...). В диалоговом окне Browse For Folder (Просмотр папок) найдите папку TipCalculator (в папке примеров книги), выделите ее и щелкните на кнопке OK. Щелкните на кнопке Finish (Готово) для выполнения импорта проекта в среду Eclipse. Проект появится в окне Package Explorer, находящемся в левой части окна Eclipse.
3. *Запустите приложение* Tip Calculator. В окне среды Eclipse щелкните правой кнопкой мыши на проекте TipCalculator в окне Package Explorer, затем в появившемся меню выберите команды Run As<Android Application (Выполнить как<Приложение Android). В результате приложение Tip Calculator начнет выполняться на виртуальном устройстве AVD, которое было создано так, как описано в разделе «Подготовительные действия» из вводной части книги.

ПРИМЕЧАНИЕ. Если было создано несколько устройств AVD либо к компьютеру подключено какое-либо физическое устройство Android, выберите одно из них для выполнения приложения.

Коснитесь компонента EditText, который называется Bill Total, чтобы отобразить числовую клавиатуру. С помощью этой клавиатуры введите число 123,45.

ПРИМЕЧАНИЕ. Если нажатие клавиш приводит к появлению японских иероглифов, коснитесь пальцем элемента EditText под названием Bill Total и удерживайте палец пару секунд. В появившемся списке параметров выберите параметр Input method (Метод ввода). Потом выберите клавиатуру Android из второго списка параметров.

Если при вводе допущена ошибка, нажмите клавишу удаления, чтобы удалить последнюю введенную цифру. Компоненты EditText, имеющие заголовки 10 %, 15 % и 20%, отображают размеры чаевых и общий счет для заранее заданных процентов чаевых (рис. 4.1, б). Также на экране отображаются компоненты EditText для пользовательских чаевых, общей суммы чаевых и общего счета для заданного по умолчанию значения 18 %, используемого при вычислении пользовательских чаевых. Все компоненты EditText, используемые для вычисления чаевых и счета, обновляются после каждого ввода или удаления числа.

Выбор процентной ставки для пользовательских чаевых. Воспользуйтесь компонентом Seekbar для ввода процентной ставки, используемой при вычислении пользовательских чаевых. Перетаскивайте ползунок Seekbar до тех пор, пока процентная ставка для пользовательских чаевых не будет равна 17 %. Сумма чаевых и общего счета для выбранной процентной ставки появится в компонентах EditText, находящихся ниже компонента Seekbar. По умолчанию компонент Seekbar позволяет выбирать значения, находящиеся в диапазоне от 0 до 100.

В этой главе будут использоваться многие объектно-ориентированные свойства Java, включая классы, анонимные внутренние классы, объекты, методы, интерфейсы и наследование. Мы создадим подкласс класса Android Activity, с помощью которого определяется, что произойдет в случае запуска приложения на выполнение, а также логика приложения Tip Calculator. Будет реализовано программное взаимодействие с компонентами EditText, TextView и SeekBar. Эти компоненты создаются с помощью Visual Layout Editor и окна Outline в Eclipse, а также ручного редактирования XML-разметки, используемой для создания GUI. Компонент EditText, который часто называется текстовым полем или текстовым окном в других GUI-технологиях, — это

подкласс компонента `TextView`. Компонент `EditText` отображает текст либо принимает текст, вводимый пользователем. Компонент `SeekBar`, в других GUI-технологиях называемый ползунком, представляет целое число, заданное по умолчанию в диапазоне 0–100, которое может быть выбрано пользователем. Для организации взаимодействия между пользователем и интерфейсом пользователя применяется обработка событий и анонимные внутренние классы.

4.2 Создание графического интерфейса приложения

В этом разделе создается графический интерфейс приложения `Tip Calculator` с помощью инструментов модуля `ADT Plugin`. В конце этого раздела представлен код XML, сгенерированный модулем `ADT Plugin` для макета этого приложения. Мы подробно рассмотрим шаги, используемые для создания графического интерфейса. В следующих главах книги будут описаны новые свойства графического интерфейса каждого приложения, а также представлена завершенная XML-разметка интерфейса приложения с выделенными измененными фрагментами [1].

ПРИМЕЧАНИЕ. После выполнения операций, описанных в этом разделе, будет сформирован графический интерфейс, который будет отличаться от интерфейса на рис. 4.1 до тех пор, пока не будут выполнены шаги, описанные в разделах 4.4.2–4.4.4.

В приложении, создаваемом в этой главе, используется объект `TableLayout` (см. рис. 4.2) для расположения компонентов GUI в шести строках и четырех столбцах. Каждая ячейка, определенная объектом `TableLayout`, может быть пустой или включать один компонент, в качестве которого может выступать макет, содержащий другие компоненты. В строках 0 и 4 на рис. 4.2 показан компонент, который может охватывать несколько столбцов. Для создания строк используются объекты `TableRow`. Количество столбцов в `TableLayout` определяется с помощью объекта `TableRow`, который содержит большинство компонентов. Высота каждой строки определяется наиболее длинным компонентом в этой строке (обратите внимание на рис. 4.2, где строки 1 и 4 короче других строк). Аналогично ширина столбца определяется наиболее широким элементом этого столбца (если не разрешить «растягивание» столбцов таблицы для заполнения экрана по ширине, в результате чего столбцы становятся шире). По умолчанию компоненты добавляются в строку в направлении слева направо. Можно указать точное положение компонента с помощью нумерации, которая по умолчанию начинается с 0. Дополнительные сведения о классе `TableLayout` можно найти на веб-сайте: developer.android.com/reference/android/widget/TableLayout.html, а сведения о классе `TableRow` — на веб-сайте: developer.android.com/reference/android/widget/TableRow.html.

Строки и столбцы объекта `TableLayout`

	Столбец 0	Столбец 1	Столбец 2	Столбец 3
Строка 0	Bill total	0.00		
Строка 1		10%	15%	20%
Строка 2	Tip	0.00	0.00	0.00
Строка 3	Total	0.00	0.00	0.00
Строка 4	Custom			
Строка 5	Tip	0.00	Total	0.00

Рис. 4.2. Класс `TableLayout` с подписанными строками и столбцами,

используемый для создания GUI приложения Tip Calculator

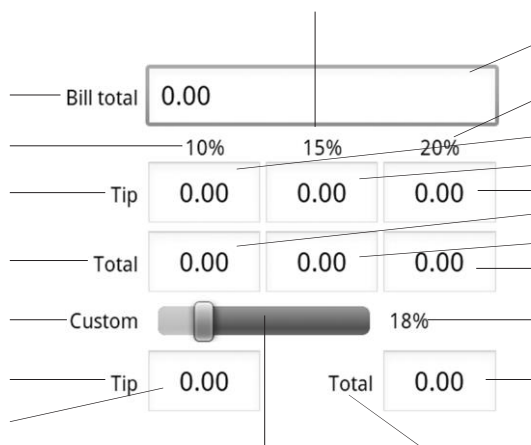


Рис. 4.3. Компоненты GUI приложения Tip Calculator, в качестве названий которых используются значения свойства Id

На рис. 4.3 показаны имена всех компонентов GUI, применяемых для формирования графического интерфейса приложения. Используемое соглашение о наименовании предусматривает использование имени класса компонента GUI в качестве значения свойства Id каждого компонента в XML-разметке и в имени переменной каждого компонента в коде Java.

```
fifteenTextView  
  
billEditText  
billTextView tenTextView tipTextView  
  
totalTextView customTextView  
twentyTextView tip10EditText  
tip15EditText tip20EditText total10EditText total15EditText  
total20EditText  
  
customTipTextViewtipCustomTextView  
totalCustomEditTexttipCustomEditText  
  
customSeekBar totalCustomTextView
```

А теперь приступим к созданию GUI, показанного на рис. 4.2. Начнем с формирования базового макета и элементов управления, потом настроим свойства элементов управления, завершив тем самым проект. После добавления компонентов в каждую строку объекта `TableLayout` можно настроить свойства `Id` и `Text` этих компонентов (см. рис. 4.3). В разделе 4.5 предыдущей главы упоминалось о том, что при разработке проекта значения литеральных строк должны включаться в файл `strings.xml`, который находится в папке `res/values` приложения. Эта методика особенно полезна в тех случаях, когда вы намереваетесь локализовать приложение для его использования на разных языках. Для компонентов `TextView` с названиями 10 %, 15 % и 20 % строковые ресурсы не используются. Убедитесь в том, что выполнили шаги, требуемые для создания GUI, компоненты которого расположены в заданном порядке. Если заданный порядок нарушен, измените расположение компонентов в окне `Outline` либо в файле `main.xml`.

При выполнении следующих шагов с помощью окна `Outline` выполняется добавление компонентов в соответствующие объекты `TableRow` из класса `TableLayout`.

При работе с более сложными структурами, например TableLayouts, с помощью Visual Layout Editor довольно трудно просматривать сложную многоуровневую структуру макета и помещать компоненты на нужный уровень. В окне Outline отображается разветвленная структура GUI, в связи с чем, задача проектирования существенно упрощается. Например, можно выбрать соответствующую строку в TableLayout и добавить в эту строку выбранный вами компонент GUI.

Шаг 1. Создание проекта TipCalculator. В среде Eclipse допускается создание лишь одного проекта с заданным именем для одной рабочей области. Поэтому перед выполнением этого шага удалите из рабочей области существующее приложение Tip Calculator, которое ранее уже тестировалось. Для выполнения этого действия щелкните на названии приложения правой кнопкой мыши и в контекстном меню выберите параметр Delete (Удалить). В появившемся диалоговом окне отмените установку флажка Delete project contents on disk is not selected (Удалить содержимое проекта на диске) и щелкните на кнопке OK. В результате произойдет удаление проекта из рабочей области, а папка проекта, находящаяся на диске, останется нетронутой. Затем создайте новый проект Android под названием TipCalculator. В диалоговом окне New Android Project (Новый проект Android) укажите следующие значения и щелкните на кнопке Finish (Готово):

- Build Target (Операционная система): выберите Android 2.3.3.
- Application name (Имя приложения): Tip Calculator.
- Package name (Название пакета): com.deitel.tipcalculator.
- Create Activity (Создать деятельность): TipCalculator.
- Min SDK Version (Минимальная версия SDK): 10.

ПРИМЕЧАНИЕ. Эта версия SDK соответствует версии Android 2.3.3, хотя в этом приложении не будут использоваться функции, присущие именно Android 2.3.3. Если планируется выполнять это приложение на виртуальном устройстве Android (AVD) либо на реальном устройстве, на котором выполняется более ранняя версия Android, параметру Min SDK присвойте меньшее значение. Например, если выбрано значение 8 для этого параметра, приложение будет выполняться на версии Android 2.2 или более старшей.

Шаг 2. Удаление и повторное создание файла main.xml. Чтобы удалить файл main.xml, щелкните на нем (в папке проектов /res/layout) и в контекстном меню выберите параметр Delete (Удалить).

1. Правой кнопкой мыши щелкните на папке layout и выберите команды New<Other....
2. (Создать<Другое...). Отобразится диалоговое окно New (Создать).
3. В узле Android выберите параметр Android XML File (XML-файл Android). После щелчка на кнопке Next> (Далее>) появится диалоговое окно New Android XML File (Создать XML-файл Android).
4. Выберите название файла main.xml и выделите TableLayout, потом щелкните на кнопке Finish.

Шаг 3. Конфигурирование Visual Layout Editor для использования подходящей библиотеки Android SDK. После выполнения предыдущего шага в окне Visual Layout Editor открывается новый файл main.xml. Если установлено несколько библиотек Android SDK, модуль ADT Plugin выбирает последнюю библиотеку в качестве стандартной, используемой для создания интерфейса (на вкладке Graphical Layout), причем этот выбор независим от SDK, выбранной при создании проекта. В раскрывающемся списке селектора SDK, находящемся в правой верхней части вкладки Graphical Layout, выберите параметр Android 2.3.3 (см. рис. 4.7). В результате разрабатываемый графический интерфейс будет совместим с устройствами Android 2.3.3.

Шаг 4. Настройка размеров и разрешения окна Visual Layout Editor. В раскрывающемся списке Device Configurations (Конфигурации устройств), находящемся в левой верхней части вкладки Graphical Layout (см. рис. 4.11), выберите параметр 3.7in WVGA (Nexus One). В результате будет настроена область конфигурирования для устройств с экраном, имеющим разрешение 480×800 (WVGA).

Шаг 5. Конфигурирование параметров объекта TableLayout. В окне Outline выберите параметр TableLayout. На экране появится окно Properties, в котором представлены следующие свойства.

1. Background (Фон): #FFF.
2. Id (Идентификатор): @+id/tableLayout.
3. Padding (Отступ): 5 dp.
4. Stretch columns (Растяжение столбцов): 1, 2, 3.

По умолчанию параметрам Layout width и Layout height присваивается значение match_parent, в результате чего макет занимает весь экран. Параметру Padding присваивается значение 5 dp, в результате чего вдоль границы всего макета создается кайма шириной в 5 независимых от плотности пикселей. С помощью параметра Stretch columns (представлен в XML атрибутом android:stretchColumns, см. листинг 4.1, строка 8) определяется растяжение по горизонтали столбцов 1–3, чтобы заполнить макет по ширине. Ширина столбца 0 соответствует ширине наиболее широкого содержащегося в нем элемента вместе с отступом вокруг этого элемента.

Шаг 6. Добавление компонентов TableRow. На этом шаге с помощью окна Outline добавим шесть компонентов TableRow в объект TableLayout. Выполните следующие действия:

1. В окне Outline щелкните правой кнопкой мыши на компоненте tableLayout и выберите добавляемый компонент Outline.
2. Повторите этот процесс еще пять раз.

При каждом повторении процесса не забудьте щелкнуть правой кнопкой мыши на tableLayout с тем, чтобы TableRow корректно вкладывались в TableLayout. Свойству Id для каждого TableRow автоматически присваиваются значения от tableRow1 до tableRow6 соответственно. Поскольку нумерация столбцов начинается с 0, соответствующим образом были изменены свойства Id для TableRow, которые получили значения от tableRow0 до tableRow5 соответственно. Также выберите каждый объект TableRow и присвойте его свойству Layout width значение match_parent. В результате строки займут все пространство макета. Чтобы выполнить эту операцию для всех компонентов TableRow одновременно, щелкните на первом компоненте TableRow, отображенном в окне Outline. Затем, удерживая клавишу Shift, щелкните на последнем компоненте TableRow в окне Outline, чтобы выделить все шесть компонентов. Затем присвойте значения свойствам.

Шаг 7. Добавление компонентов в tableRow. На этом шаге компоненты TextView и EditText будут добавлены в tableRow0. Выполните следующие действия:

1. Перетащите компонент TextView (billTextView) из раздела Form Widgets (Виджеты форм) палитры в tableRow0, находящийся в окне Outline.
2. Перетащите компонент EditText (billEditText) из раздела Form Widgets (Виджеты форм) палитры в tableRow0, находящийся в окне Outline.
3. Присвойте значения свойствам Id и Text для каждого компонента. Чтобы получить быстрый доступ к этим свойствам, щелкните правой кнопкой мыши на компоненте в окне Outline и в контекстном меню выберите команды Edit ID...(Изменить идентификатор...) и Edit Text...(Изменить текст...) соответственно.

Важно перетащить указанные выше компоненты на подходящие объекты TableRow в окне Outline, чтобы они были вложены в требуемый объект TableRow.

Шаг 8. Добавление компонентов в tableRow1. Добавьте три компонента TextView в объект tableRow1, выполнив следующие действия:

1. Перетащите компонент TextView (tenTextView) на объект tableRow1, находящийся в окне Outline.
2. Повторите этот процесс, добавив компоненты fifteenTextView и twentyTextView.
3. Установите значения свойств Id и Text для каждого компонента.

Шаг 9. Добавление компонентов в tableRow2. Добавьте компонент TextView и три компонента EditText в tableRow2, выполнив следующие действия:

1. В окне Outline перетащите компонент TextView (tipTextView) на tableRow2.
2. В окне Outline перетащите три компонента EditText на tableRow2 (tip10EditText, tip15EditText и tip20EditText).
3. Установите значения свойств Id и Text для каждого компонента.

Шаг 10. Добавление компонентов в tableRow3. Добавьте компонент TextView и три компонента EditText в tableRow3, выполнив следующие действия:

1. В окне Outline перетащите компонент TextView (totalTextView) на tableRow3.
2. В окне Outline перетащите три компонента EditText на tableRow3 (total10EditText, total15EditText и total20EditText).
3. Установите значения свойств Id и Text для каждого компонента.

Шаг 11. Добавление компонентов в tableRow4. Добавьте компоненты TextView, SeekBar и другой компонент TextView в tableRow4, выполнив следующие действия:

1. В окне Outline перетащите компонент TextView (customTextView) на tableRow4.
2. В окне Outline перетащите компонент SeekBar (customSeekBar) на tableRow4.
3. В окне Outline перетащите компонент TextView (customTipTextView) на tableRow4.
4. Установите значения свойств Id и Text для компонентов TextView.

Шаг 12. Добавление компонентов в tableRow5. Добавьте компонент TextView, EditText, другой компонент TextView и еще один компонент EditText в tableRow5, выполнив следующие действия:

1. В окне Outline перетащите компонент TextView (tipCustomTextView) на tableRow5.
2. В окне Outline перетащите компонент EditText (tipCustomEditText) на tableRow5.
3. В окне Outline перетащите компонент TextView (totalCustomTextView) на tableRow5.
4. В окне Outline перетащите компонент EditText (totalCustomEditText) на tableRow5.
5. Установите значения свойств Id и Text для каждого компонента.

Начиная с этого момента, графический интерфейс пользователя будет выглядеть так, как показано на рис. 4.4.

Рис. 4.4. Графический интерфейс пользователя Tip Calculator перед настройкой свойств, отличающихся от свойств Id и Text для каждого компонента

При сравнении с рис. 4.2 заметны следующие различия:

- Компоненты `billEditText` и `customSeekBar` еще не охватывают несколько столбцов.
- Текст для всех компонентов `TextView` выделен светло-серым цветом, что затрудняет его чтение.
- Некоторые из компонентов не «попадают» в свой столбец.
- Например, компоненты `TextView`, имеющие заголовки 10 %, 15 % и 20 %, находятся в столбце `tableRow1`, а компонент `TextView` с заголовком 18 % — в столбце `tableRow4`. Последний компонент автоматически «попадет» в свой столбец, если объект `customSeekBar` будет охватывать два столбца.

Большая часть текста на рис. 4.2 выровнена по центру или по правому краю, а текст на рис. 4.4 выровнен по левому краю.

В результате выполнения дальнейших шагов мы завершим разработку приложения путем настройки его свойств.

Шаг 13. Изменение свойства `Text color` для всех элементов `TextView`. В окне Outline можно одновременно выбрать несколько компонентов. Для этого нужно щелкать на каждом компоненте при нажатой клавише `Ctrl` (либо `Control`). После завершения выделения в окне Properties отобразятся только общие свойства для выделенных компонентов. При изменении значения свойства для нескольких выделенных компонентов значение свойства изменится для каждого выделенного компонента. Чтобы улучшить читаемость текста для компонентов `TextView`, измените цвет текста на черный. Для этого нужно одновременно изменить свойство `Text color` для всех компонентов `TextView`, выполнив следующие действия:

1. Удерживая клавишу `Ctrl` (или `Control`), щелкайте на каждом компоненте `TextView` для его выделения.
2. В окне Properties найдите свойство `Text color` и присвойте ему значение `#000`.

Шаг 14. Перемещение компонентов `TextView` с заголовками 10 %, 15 % и 20 % в требуемые столбцы. Как показано на рис. 4.2, заголовки столбцов 10 %, 15 % и 20 % находятся во втором, третьем и четвертом столбцах соответственно. По умолчанию при добавлении компонентов в `TableRow` первый компонент помещается в первый столбец, второй компонент — во второй столбец и т. д. Чтобы переместить компонент в другой столбец, укажите номер столбца компонента.

К сожалению, это свойство по умолчанию не отображается в окне Properties. Поэтому для указания столбца компонента придется изменить соответствующий ему XML- код.

1. Чтобы перейти к просмотру XML-разметки, выберите вкладку `main.xml` в окне редактора Visual Layout Editor.

2. Выберите элемент `<TextView>` с атрибутом `android:id`, имеющим значение `"@+id/ tenTextView"`.
3. В открывающий XML-тег компонента `TextView` добавьте следующую пару «атрибут/ значение»: `android:layout_column="1"`.

В результате выполнения этой операции компонент `TextView` с заголовком 10 % переместится во второй столбец (нумерация столбцов будет начинаться с 0). При этом остальные компоненты строки автоматически переместятся в последующие столбцы. Если нужно пропустить другие столбцы, установите значение компонента `android:layout_column` для каждого компонента в строке таким образом, чтобы определить каждый столбец, в котором будет отображаться компонент. Если в XML-код будет добавлен атрибут вручную, атрибут и его значение будут отображаться в разделе `Misc` окна `Properties`.

Шаг 15. Центрирование текста в компонентах `TextView` объекта `tableRow1` и в компонентах `EditText` объектов `tableRow2`, `tableRow3` и `tableRow5`, а также установка размера шрифта для компонентов `EditText`. Как показано на рис. 4.2, текст, отображаемый во многих компонентах, является центрированным. Для выполнения центрирования текста, отображаемого в компонентах, применяется свойство `Gravity`. Вернитесь обратно на вкладку `Graphical Layout` в окне `Visual Layout Editor` и выполните следующие действия:

1. В окне `Outline` выберите три компонента `TextView` объекта `tableRow1`.
2. В окне `Properties` свойству `Gravity` присвойте значение `center`.
3. Выберите все компоненты `EditText`, относящиеся к объектам `tableRow2`, `tableRow3` и `tableRow5`.
4. В окне `Properties` свойству `Gravity` присвойте значение `center`.
5. Свойству `Text size` присвойте значение `14sp`. В результате будет уменьшен заданный по умолчанию размер шрифта в компонентах `EditText`, что позволит отображать больше цифр без переноса текста.

Шаг 16. Охват нескольких столбцов путем настройки компонентов `billEditText` и `customSeekBar`. На рис. 4.2 компонент `billEditText` занимает столбцы 1–3, а компонент `customSeekBar` — столбцы 1–2. Атрибут, определяющий охват нескольких столбцов, можно добавить непосредственно в XML-код:

1. Чтобы просмотреть разметку макета, выберите вкладку `main.xml` в окне редактора `Visual Layout Editor`.
2. Выберите элемент `<EditText>` с атрибутом `android:id`, который имеет значение `"@+id/ billEditText"`.
3. В открывающий XML-тег `EditText` добавьте следующую пару «атрибут/значение»: `android:layout_span="3"`.
4. Выберите элемент `<SeekBar>`.
5. В открывающий XML-тег `SeekBar` добавьте следующую пару «атрибут/значение»: `android:layout_span="2"`.

Теперь компонент `billEditText` будет охватывать столбцы 1–3, а компонент `customSeekBar` — столбцы 1–2.

Шаг 17. Выравнивание компонентов `TextView` по правому краю. Компоненты `TextView` в столбце 0 выровнены по правому краю подобно компоненту `TextView`, который выровнен по правому краю в третьем столбце объекта `tableRow5`. Также каждый из элементов `TextView` имеет отступ справа, равный `5dp`. Этот отступ отделяет данный компонент от элемента управления, находящегося справа от него.

1. Вернитесь обратно на вкладку `Graphical Layout` в окне редактора `Visual Layout Editor`.
2. В окне `Outline` выберите все компоненты `TextView` в столбце 0, а в

последней строке — второй компонент TextView.

3. Свойству Gravity присвойте значение right, а свойству Padding right — значение 5dp.

Шаг 18. Центрирование по вертикали компонентов TextView в tableRow4 А теперь настроим свойство Gravity, чтобы выровнять по вертикали компоненты TextView, относящиеся к объекту tableRow4:

1. В окне Outline выберите компонент customTextView объекта tableRow4.
2. Выберите свойство Gravity и щелкните на кнопке с многоточием, которая находится справа от значения свойства, чтобы отобразить список возможных значений свойства Gravity.
3. Выберите значение свойства center_vertical. Следует установить значения right и center_vertical.
4. Чтобы применить значение, щелкните на кнопке OK.
5. В окне Outline выберите компонент customTipTextView, относящийся к объекту tableRow4.
6. Свойству Gravity присвойте значение center_vertical.
7. Для применения значения щелкните на кнопке OK.
8. В окне Outline выберите два компонента TextView объекта tableRow4, свойствам Layout height присвойте значения match_parent, а свойству Padding bottom присвойте значение 5dp. В результате два компонента TextView будут иметь ту же высоту, что и компонент SeekBar, а с помощью свойства Gravity текст будет выравниваться по вертикали вместе с компонентом SeekBar. Свойству Padding bottom также будет присвоено значение SeekBar. Благодаря настройке значения этого свойства для свойства TextView текст будет выровнен относительно компонента SeekBar.
9. И наконец, присвоим свойству Padding left объекта customTipTextView значение 5dp, чтобы визуально отделить компонент TextView от компонента SeekBar.

Шаг 19. Установка свойств Progress Property и Padding объекта customSeekBar. Чтобы завершить создание графического интерфейса пользователя, присвойте значения свойствам Progress, Padding left и Padding right компонента SeekBar. Сначала настройте положение ползунка SeekBar, чтобы представить значение 18 %, которое отображается в TextView справа от SeekBar. Также добавьте небольшой отступ слева и справа от SeekBar. Если ползунок перемещается в крайнее левое или крайнее правое положение компонента SeekBar (представляя значения 0 и 100 соответственно), его довольно трудно захватывать, поскольку недостаточно свободного места между компонентом SeekBar и другими компонентами, находящимися слева и справа от него.

1. В окне Outline выберите компонент customSeekBar.
2. Свойству Progress присвойте значение 18.
3. Свойствам Padding left и Padding right присвойте значение 8dp.
4. Свойству Padding bottom присвойте значение 5dp, чтобы отделить с помощью отступа последнюю строку компонентов.
5. Свойству Focusable присвойте значение false. После этого в случае изменения пользователем значения SeekBar компонент billEditText будет сохранять фокус. Это полезно в том случае, если нужно отображать на экране клавиатуру (для устройств, отображающих программную клавиатуру).

Шаг 20. Предотвращение возможности изменения пользователем

результатов вычислений, отображаемых компонентом EditText. За исключением компонента `billEditText`, находящегося в верхней части графического интерфейса пользователя, все другие компоненты `EditText` приложения применяются лишь для отображения результатов вычислений. Поэтому пользователь не может изменять текст, соответствующий этим компонентам. Контролировать получение фокуса компонента `EditText` пользователем можно путем установки свойства `Focusable`. Можно также запретить длинное нажатие на компоненте `EditText`, а также предотвратить отображение курсора в области `EditText`. Это приведет к блокированию доступа пользователя к тексту. Настройте следующие параметры:

1. В окне `Outline` выберите все компоненты `EditText` за исключением компонента `billEditText`.
2. Свойствам `Focusable`, `Long clickable` и `Cursor visible` присвойте значения `false`.

Шаг 21. Указание типа числовой клавиатуры с помощью `billEditText`.

Можно настроить параметры, обеспечивающие ввод пользователем чисел с плавающей запятой в компонент `billEditText`. Выполните следующие действия:

1. В окне `Outline` выберите компонент `billEditText`.
2. Свойству `Input type` присвойте значение `numberDecimal`.

Шаг 22. Настройка значений «веса» макета для различных компонентов.

Свойство `Layout weight` компонента определяет его важность (вес) относительно других компонентов. По умолчанию свойству `Layout weight` для всех компонентов присваивается значение 0. Это свойство определяет относительную величину данного компонента по отношению к другим компонентам. В данном случае свойству `Layout weight` присваивается 1 для всех компонентов за исключением компонентов `TextView`, находящихся в левом столбце. Если макет растягивается для заполнения экрана, компоненты `TextView`, находящиеся в левом столбце экрана, занимают область, ширина которой соответствует ширине наиболее широкого компонента `TextView` в этом столбце. Другие компоненты со свойством `Layout weight`, которому присвоено значение 1, растягиваются для заполнения оставшейся области экрана. При этом пространство экрана делится равномерно. Если свойству `Layout weight` для компонента строки присвоено значение 2, этот компонент будет занимать в два раза больше места, чем компонент со свойством `Layout weight`, значение которого равно 1 (в этой же строке).

На этом разработка графического интерфейса пользователя завершена. В следующем разделе будет представлена XML-разметка, сгенерированная редактором `Visual Layout Editor`. Ниже приводится код приложения.

Теперь разработанный вами GUI будет выглядеть так, как показано на рис. 4.2. В листинге 4.1 представлена завершенная XML-разметка графического интерфейса пользователя приложения `Tip Calculator`. Сгенерированный XML-код был переформатирован, а также добавлены комментарии, облегчающие читабельность. Выделены области кода, определяющие новые функции GUI, которые были рассмотрены в разделах 4.4.2 и 4.4.4.

Листинг 4.1. XML-разметка приложения `Tip Calculator`.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <!-- main.xml -->

3 <!-- XML-разметка приложения Tip Calculator -->
4

5 <TableLayout
6     xmlns:android=http://schemas.android.com/apk/res/android
    android:layout_width="match_parent"
    android:layout_height="match_parent"
```

```

7   android:background="#FFF" android:id="@+id/tableLayout"
8   android:stretchColumns="1,2,3"      android:padding="5dp">
9
10  <!-- tableRow0 -->
11  <TableRow    android:layout_height="wrap_content"
12    android:layout_width="match_parent"
13    android:id="@+id/tableRow0">
14      <TextView    android:id="@+id/billTextView"
15        android:layout_width="wrap_content"
16        android:layout_height="wrap_content"
17        android:text="@string/billTotal"
18        android:textColor="#000"
19        android:gravity="right"
20        android:paddingRight="5dp"></TextView>
21      <EditText    android:layout_width="wrap_content"
22        android:id="@+id/billEditText"
23        android:layout_height="wrap_content"
24        android:layout_span="3"
25        android:inputType="numberDecimal"
26        android:layout_weight="1">
27      </EditText>
28    </TableRow>
29
30  <!-- tableRow1 -->
31  <TableRow    android:layout_height="wrap_content"
32    android:layout_width="match_parent"
33    android:id="@+id/tableRow1">
34      <TextView    android:id="@+id/tenTextView"
35        android:layout_width="wrap_content"
36        android:layout_height="wrap_content"      android:text="10
37        %"
38        android:textColor="#000"  android:layout_column="1"
39        android:gravity="center"
40        android:layout_weight="1"></TextView>
41      <TextView    android:id="@+id/fifteenTextView"
42        android:layout_width="wrap_content"
43        android:layout_height="wrap_content"      android:text="15
44        %"
45        android:textColor="#000"  android:gravity="center"
46        android:layout_weight="1"></TextView>
47      <TextView    android:id="@+id/twentyTextView"
48        android:layout_width="wrap_content"
49        android:layout_height="wrap_content"      android:text="20
50        %"
51        android:textColor="#000"  android:gravity="center"
52        android:layout_weight="1"></TextView>
53    </TableRow>
54
55  <!-- tableRow2 -->
56  <TableRow    android:layout_height="wrap_content"
57    android:layout_width="match_parent"
58    android:id="@+id/tableRow2">
59      <TextView    android:id="@+id/tipTextView"
60        android:layout_width="wrap_content"
61        android:layout_height="wrap_content"
62        android:text="@string/tip" android:textColor="#000"
63        android:gravity="right"

```

```

        android:paddingRight="5dp"></TextView>
53 <EditText    android:layout_width="wrap_content"
54     android:id="@+id/tip10EditText"
55     android:layout_height="wrap_content"
        android:text="@string/zero"
56     android:gravity="center"        android:focusable="false"
57     android:layout_weight="1"        android:textSize="14sp"
58     android:cursorVisible="false"
        android:longClickable="false">
59 </EditText>
60 <EditText    android:layout_width="wrap_content"
61     android:id="@+id/tip15EditText"
62     android:layout_height="wrap_content"
        android:text="@string/zero"
63     android:gravity="center"        android:focusable="false"
64     android:layout_weight="1"        android:textSize="14sp"
65     android:cursorVisible="false"
        android:longClickable="false">
66 </EditText>
67 <EditText    android:layout_height="wrap_content"
68     android:layout_width="wrap_content"
69     android:id="@+id/tip20EditText"
        android:text="@string/zero"
70     android:gravity="center"        android:focusable="false"
71     android:layout_weight="1"        android:textSize="14sp"
72     android:cursorVisible="false"
        android:longClickable="false">
73 </EditText>
74 </TableRow>
75
76 <!-- tableRow3 -->
77 <TableRow    android:layout_height="wrap_content"
78     android:layout_width="match_parent"
        android:id="@+id/tableRow3">
79 <TextView    android:layout_width="wrap_content"
80     android:layout_height="wrap_content"
81     android:id="@+id/totalTextView"
        android:text="@string/total"
82     android:textColor="#000"        android:gravity="right"
83     android:paddingRight="5dp"></TextView>
84 <EditText    android:layout_width="wrap_content"
85     android:text="@string/zero"
        android:layout_height="wrap_content"
86     android:id="@+id/total10EditText"
        android:gravity="center"
87     android:focusable="false"    android:layout_weight="1"
88     android:textSize="14sp"    android:cursorVisible="false"
89     android:longClickable="false"></EditText>
90 <EditText    android:layout_width="wrap_content"
91     android:text="@string/zero"
        android:layout_height="wrap_content"
92     android:id="@+id/total15EditText"
        android:gravity="center"
93     android:focusable="false"    android:layout_weight="1"
94     android:textSize="14sp"    android:cursorVisible="false"
95     android:longClickable="false"></EditText>
96 <EditText    android:layout_width="wrap_content"
97     android:text="@string/zero"
        android:layout_height="wrap_content"

```

```

98 android:id="@+id/total20EditText"
    android:gravity="center"
99 android:focusable="false" android:layout_weight="1"
100 android:textSize="14sp" android:cursorVisible="false"
101 android:longClickable="false"></EditText>
102</TableRow>
103
104<!-- tableRow4 -->
105<TableRow android:layout_height="wrap_content"
106 android:layout_width="match_parent"
    android:id="@+id/tableRow4">
107<TextView android:id="@+id/customTextView"
108 android:layout_width="wrap_content"
    android:text="@string/custom"
109 android:textColor="#000" android:paddingRight="5dp"
110 android:gravity="right|center_vertical"
111 android:layout_height="match_parent"
    android:paddingBottom="5dp"
112 android:focusable="false"></TextView>
113<SeekBar android:layout_height="wrap_content"
114 android:layout_width="match_parent"
115 android:id="@+id/customSeekBar" android:layout_span="2"
116 android:progress="18" android:paddingLeft="8dp"
117 android:paddingRight="8dp" android:paddingBottom="5dp"
118 android:layout_weight="1"></SeekBar>
119<TextView android:id="@+id/customTipTextView"
120 android:layout_width="wrap_content" android:text="18
    %"
121 android:textColor="#000"
    android:gravity="center_vertical"
122 android:layout_height="match_parent"
    android:paddingLeft="5dp"
123 android:paddingBottom="5dp" android:focusable="false"
124 android:layout_weight="1"></TextView>
125</TableRow>
126
127<!-- tableRow5 -->
128<TableRow android:layout_height="wrap_content"
129 android:layout_width="match_parent"
    android:id="@+id/tableRow5">
130<TextView android:layout_width="wrap_content"
131 android:layout_height="wrap_content"
132 android:id="@+id/tipCustomTextView"
    android:text="@string/tip"
133 android:textColor="#000" android:gravity="right"
134 android:paddingRight="5dp"></TextView>
135<EditText android:layout_width="wrap_content"
136 android:layout_height="wrap_content"
137 android:id="@+id/tipCustomEditText"
    android:text="@string/zero"
138 android:gravity="center" android:focusable="false"
139 android:layout_weight="1" android:textSize="14sp"
140 android:cursorVisible="false"
    android:longClickable="false">
141</EditText>
142<TextView android:id="@+id/totalCustomTextView"
143 android:layout_width="wrap_content"
144 android:layout_height="wrap_content"
    android:text="@string/total"
145 android:textColor="#000" android:gravity="right"

```

```

146android:paddingRight="5dp"
    android:layout_weight="1"></TextView>
147<EditText    android:layout_height="wrap_content"
148android:layout_width="wrap_content"
149android:id="@+id/totalCustomEditText"
    android:text="@string/zero"
150android:gravity="center"    android:focusable="false"
151android:layout_weight="1"    android:textSize="14sp"
152android:cursorVisible="false"
    android:longClickable="false">
153</EditText>
154</TableRow>
155</TableLayout>
156

```

В листинге 4.2 приведены строковые ресурсы, используемые в листинге 4.1.

Листинг 4.2. Строковые ресурсы, которые хранятся в файле strings.xml

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3      <string name="app_name">Tip Calculator</string>
4      <string name="billTotal">Bill total</string>
5      <string name="tip">Tip</string>
6      <string name="total">Total</string>
7      <string name="custom">Custom</string>
8      <string name="zero">0.00</string>
9  </resources>

```

4.3 Включение дополнительных функций в приложение

В листингах 4.3–4.11 приведена реализация приложения Tip Calculator в единственном классе TipCalculator. Этот класс вычисляет чаевые на основе процентной ставки, равной 10 %, 15 %, 20 %, а также на базе пользовательской процентной ставки. Чаевые вычисляются на основании выставленного счета, затем вычисленные чаевые добавляются к выставленному счету, образуя счет к оплате (итоговый счет) [1].

Операторы package и import. В листинге 4.3 приводится код операторов package и import, который включен в класс TipCalculator.java. Оператор package, находящийся в строке 3, указывает на то, что класс в этом файле является частью пакета package com.deitel.tipcalculator. Эта строка была вставлена при создании проекта на шаге 1 в разделе 4.2.

Листинг 4.3. Операторы package и import, входящие в состав класса TipCalculator.

```

1 // TipCalculator.java
2 //  Вычисляет чаевые на основе счета с 5, 10, 15 или
   //  введенной пользователем процентной ставки.

3 package com.deitel.tipcalculator;
4
5 import android.app.Activity;
6 import android.os.Bundle;
7 import android.text.Editable;
8 import android.text.TextWatcher;
9 import android.widget.EditText;
10 import android.widget.SeekBar;
11 import android.widget.SeekBar.OnSeekBarChangeListener;

```

```

12 import android.widget.TextView;
13

```

Операторы **import**, находящиеся в строках 5–12, импортируют различные интерфейсы и классы, используемые приложением.

Класс **Activity**, относящийся к пакету `android.app` (строка 5), поддерживает базовые методы жизненного цикла, используемые в приложении (эти методы будут рассмотрены ниже).

Класс **Bundle** из пакета `android.os` (строка 6) представляет информацию о состоянии приложения. Приложение может сохранять свое состояние в случае перевода в фоновый режим операционной системой (например, если пользователь запускает другое приложение или принимает телефонный звонок).

Интерфейс **Editable** из пакета `android.text` (строка 7) позволяет изменять содержимое и разметку текста в GUI.

Интерфейс **TextWatcher** из пакета `android.text` (строка 8) служит для реагирования на события, возникающие при взаимодействии пользователя с компонентом **EditText**.

Пакет `android.widget` (строки 9–12) содержит виджеты (то есть компоненты GUI) и макеты, используемые в Android GUI, такие как `EditText` (строка 9), `SeekBar` (строка 10) и `TextView` (строка 12).

Интерфейс **SeekBar.OnSeekBarChangeListener** из пакета `android.widget` (строка 11) предназначен для реагирования на события, возникающие при перемещении ползунка `SeekBar`.

Класс Activity и жизненный цикл деятельности приложения Tip Calculator. Приложения Android не включают главный метод (`main method`). Они могут включать четыре типа компонентов: деятельности, службы, провайдеры контента и широковещательные приемники. Порядок инициализации этих компонентов рассмотрим позднее. В этой главе мы обсуждаем только деятельности. Взаимодействие пользователей с деятельностями осуществляется с помощью представлений (или видов), которые являются компонентами GUI. Каждая отдельная деятельность обычно связана с отдельным экраном приложения.

Класс `TipCalculator` (см. листинги 4.4–4.11) представляет собой класс `Activity` приложения `Tip Calculator`. В следующих главах будут рассмотрены примеры приложений, включающих несколько деятельностей, представляющих отдельные экраны приложения. Класс `TipCalculator` является результатом расширения (наследуется) класса `Activity` (строка 15). В процессе создания проекта `TipCalculator` модуль ADT Plugin генерирует этот класс как подкласс класса `Activity` и поддерживает оболочку для переопределенного метода `onCreate`, с помощью которого переопределяется каждый подкласс `Activity`. Ниже мы рассмотрим этот метод.

Листинг 4.4. Класс `TipCalculator` является подклассом класса `Activity`

```

14 // Главный класс Activity для приложения
    TipCalculator
15 public class TipCalculator extends
    Activity 16 {

```

На протяжении своего жизненного цикла деятельность может находиться в одном из следующих состояний: активное (выполняется), приостановленное или остановленное. Переход между различными состояниями деятельности осуществляется в ответ на различные события.

Активная (или выполняющаяся) деятельность отображается на экране и «владеет фокусом» (то есть выполняется на переднем плане). Именно с этой

деятельностью взаимодействует пользователь.

Приостановленная деятельность отображается на экране, но не обладает фокусом. Эта деятельность может быть удалена из памяти операционной системой (как правило, с целью освобождения места, требуемого для выполнения другого приложения). Обратите внимание, что в первую очередь из памяти удаляются остановленные деятельности.

Остановленная деятельность не отображается на экране и будет удалена системой при нехватке памяти.

В процессе перехода деятельности между различными состояниями она принимает вызовы для различных методов жизненного цикла, определенных в классе Activity (developer.android.com/reference/android/app/Activity.html). В приложении Tip Calculator реализованы два метода жизненного цикла: onCreate и onSaveInstanceState. К другим важнейшим методам жизненного цикла относятся: onStart, onPause, onRestart, onResume, onStop и onDestroy. Все эти методы будут рассмотрены в следующих главах.

Метод onCreate вызывается системой после запуска класса Activity. После вызова этого метода отображается графический интерфейс пользователя, а пользователь получает возможность взаимодействия с классом Activity.

Метод onSaveInstanceState вызывается системой в случае изменения конфигурации устройства в процессе выполнения приложения (например, при вращении устройства пользователем или выдвигании физической клавиатуры устройства, как в оригинальном смартфоне Motorola Droid). Этот метод может применяться для сохранения информации о состоянии, которое может быть восстановлено в случае вызова метода onCreate (в качестве составной части процесса изменения конфигурации). Если приложение находится в фоновом режиме, обеспечивающем возможность ответа пользователя на телефонные звонки, либо в случае запуска пользователем другого приложения, компоненты GUI приложения будут автоматически сохранять свой контент. В результате обеспечивается возврат к выполнению приложения на переднем плане (эта возможность будет доступной в том случае, если система не «убивает» приложение).

Каждый метод жизненного цикла деятельности, переопределенный пользователем, должен сначала вызывать версию суперкласса этого метода. В противном случае вызов этого метода приведет к генерированию исключения.

Переменные класса и экземпляры переменных. В строках 18–32 листинга 4.5 объявляются переменные класса TipCalculator. Большинство из этих переменных представляют собой компоненты EditText, используемые для ввода выставленного счета пользователем, а также для отображения вычисленной суммы чаевых и общего счета (с учетом чаевых). Статические строки (строки 18–19) используются в качестве ключей в парах «ключ/значение» для представления текущего итогового счета и пользовательских процентных ставок чаевых. Пары «ключ/значение» хранятся и выбираются с помощью методов onSaveInstanceState и onCreate соответственно (при изменении конфигурации приложения).

Листинг 4.5. Переменные экземпляра класса TipCalculator.

```
17 // константы, используемые при сохранении/восстановлении состояния
18 private static final String BILL_TOTAL = "BILL_TOTAL";
19 private static final String CUSTOM_PERCENT =
"CUSTOM_PERCENT"; 20
21 private double currentBillTotal; // счет, вводимый
    пользователем
22 private int currentCustomPercent; // % чаевых, выбранный
    SeekBar
23 private EditText tip10EditText; // 10%-чаевые
24 private EditText total10EditText; // общий счет, включая 10%-
```

```

чаевые
25 private EditText tip15EditText;      // 15%-чаевые
26 private EditText total15EditText; // общий счет, включая 15%-
чаевые
27 private EditText billEditText;      // ввод счета
пользователем
28 private EditText tip20EditText;      // 20%-чаевые
29 private EditText total20EditText; // общий счет, включая 20%-
чаевые
30 private TextView customTipTextView; // % пользовательских
чаевых
31 private EditText tipCustomEditText;    // пользовательские
чаевые
32 private EditText totalCustomEditText; // общий счет
33                                     // с пользовательскими чаевыми

```

Выставленный счет, введенный пользователем в EditText billEditText, считывается и хранится в виде строки в компоненте currentBillTotal. При этом требуется выполнение определенного преобразования, суть которого будет объяснена позже. Пользовательская процентная ставка чаевых, выбираемая пользователем с помощью ползунка Seekbar (целое число, находящееся в диапазоне 0–100), будет сохранена в currentCustomPercent. Это значение обычно умножается на 0,01 для преобразования в тип данных double, используемый в вычислениях. Сумма пользовательских чаевых и общий счет, включая пользовательские чаевые, присваиваются компонентам tipCustomEditText и totalCustomEditText соответственно. В строке 30 объявляется переменная TextView, в которой хранится процентная ставка пользовательских чаевых, соответствующая отображаемой позиции ползунка SeekBar (см. значение 18 % на рис. 4.1, а).

Фиксированные процентные ставки чаевых, равные 10 %, 15 % и 20 %, а также итоговые счета (с учетом чаевых) отображаются в EditText. Сумма чаевых, подсчитанная с учетом ставки 10 %, и сумма итогового счета (вместе с чаевыми, подсчитанными по ставке 10 %) хранятся в переменных tip10EditText и total10EditText соответственно. Сумма чаевых, подсчитанная с учетом ставки 15 %, и сумма итогового счета (вместе с чаевыми, подсчитанными по ставке 15 %) хранятся с помощью переменных tip15EditText и total15EditText соответственно. Сумма чаевых, подсчитанная с учетом ставки 20 %, и сумма итогового счета (вместе с чаевыми, подсчитанными по ставке 20 %) хранятся с помощью переменных tip20EditText и total20EditText соответственно.

Переопределение метода onCreate из класса Activity. Метод onCreate (см. листинг 4.6) генерируется автоматически при создании проекта приложения и вызывается системой после запуска класса Activity. Метод onCreate обычно инициализирует переменные экземпляра класса Activity и компоненты GUI. Этот метод упрощен до предела, поэтому приложение загружается быстро. Фактически в случае, если загрузка приложения занимает более пяти секунд, операционная система отображает диалоговое окно ANR (Application Not Responding, приложение не отвечает). В этом окне отображается параметр, с помощью которого пользователь может принудительно завершить приложение. Операции по инициализации, занимающие много времени, следует выполнять в фоновом процессе, а не с помощью метода onCreate.

Листинг 4.6. Переопределение метода onCreate класса Activity

```

34 // Вызывается при первом создании класса activity.
35 @Override
36 public void onCreate(Bundle savedInstanceState)
37 {
38     super.onCreate(savedInstanceState); // вызов версии

```



```

суперкласса
39 setContentView(R.layout.main);    // «раздувание» GUI
40

41 // Приложение запущено впервые или восстановлено из памяти?
42 if ( savedInstanceState == null ) // приложение запущено
    впервые
43 {
44     currentBillTotal = 0.0; // инициализация суммы счета нулем
45     currentCustomPercent = 18; // инициализация
        пользовательских
            // чаевых значением 18 %
        конец структуры
    if

46 else // приложение восстановлено из памяти
47 {
48     // инициализация суммы счета сохраненной в памяти суммой
49     currentBillTotal =
50     savedInstanceState.getDouble(BILL_TOTAL);
51
52     // инициализация пользовательских чаевых сохраненным
        // процентом чаевых

53     currentCustomPercent =
54     savedInstanceState.getInt(CUSTOM_PERCENT);
55 } // конец структуры else
56 // ссылки на чаевые 10 %, 15 %, 20 % и итоговые EditTexts
57 tip10EditText = (EditText)
58 findViewById(R.id.tip10EditText);
59 total10EditText = (EditText)
60 findViewById(R.id.total10EditText);
61 tip15EditText = (EditText)
62 findViewById(R.id.tip15EditText);
63 total15EditText = (EditText)
64 findViewById(R.id.total15EditText);
65 tip20EditText = (EditText)
66 findViewById(R.id.tip20EditText);
67 total20EditText = (EditText)
68 findViewById(R.id.total20EditText);
69 // TextView, отображающий процент пользовательских чаевых
70 customTipTextView = (TextView)
71 findViewById(R.id.customTipTextView);
72 // пользовательские чаевые и итоговые EditTexts
73 tipCustomEditText = (EditText)
74 findViewById(R.id.tipCustomEditText);
75 totalCustomEditText =
76 (EditText) findViewById(R.id.totalCustomEditText);

77 // получение billEditText
78 billEditText = (EditText)
79 findViewById(R.id.billEditText);
80 // billEditTextWatcher обрабатывает событие onTextChanged
81 // из billEditText

82 billEditText.addTextChangedListener(billEditTextWatcher);
83
84 // получение SeekBar, используемого для подсчета суммы

```

```

// пользовательских чаевых

80     SeekBar customSeekBar = (SeekBar)
        findViewById(R.id.customSeekBar);
81     customSeekBar.setOnSeekBarChangeListener(customSeekBarListener
        );
82 } // конец метода onCreate
83

```

Во время выполнения приложения пользователь может изменить конфигурацию устройства путем его поворота либо выдвижением физической клавиатуры. Приложение без проблем переключается в различные выбранные конфигурации. Если система вызывает метод `onCreate`, он передает значение `Bundle` параметру `savedInstanceState`. Это значение содержит сохраненное состояние деятельности (при наличии подобного состояния). Обычно информация, относящаяся к этому состоянию, сохраняется с помощью метода `onSaveInstanceState` из класса `Activity` (см. листинг 4.9). (В данном случае использовался метод `savedInstanceState`, определенный в строках 42–55.) В строке 38 вызывается метод `onCreate` суперкласса, который используется при переопределении любого метода из класса `Activity`.

В процессе создания графического интерфейса пользователя и добавления ресурсов (например, строк в файл `strings.xml` или GUI-компонентов в файл `main.xml`) в приложение инструменты плагина ADT генерируют класс `R`. Этот класс содержит вложенные статические классы, представляющие каждый тип ресурса, находящийся в папке проекта `res`. Класс `R` находится в папке проекта `gen`, в которой находятся сгенерированные файлы исходного кода. В классах, вложенных в класс `R`, инструменты создают константы `static final int`, с помощью которых можно ссылаться на эти ресурсы программным образом из кода приложения (методы программного доступа будут обсуждаться далее). Рассмотрим некоторые классы, вложенные в класс `R`:

- Класс `drawable`. Этот класс включает все элементы `drawable`, например изображения, которые находятся в различных папках `drawable`, расположенных в папке приложения `res`.
- Класс `id`. Данный класс включает константы для GUI-компонентов, используемые в файлах XML-разметки.
- Класс `layout`. Здесь находятся константы, которые представляют файлы разметки проекта (например, `main.xml`).
- Класс `string`. В этом классе находятся константы, используемые для определения строк в файле `strings.xml`.

В результате вызова `setContentView` (строка 39) принимается значение константы `R.layout.main`, с помощью которой выбирается XML-файл, который представляет GUI деятельности. В данном случае константа представляет файл `main.xml`. Метод `setContentView` использует эту константу для загрузки соответствующего XML-документа, который потом анализируется и преобразуется в графический интерфейс пользователя приложения. Этот процесс известен под названием «раздувание» GUI.

В строках 42–55 определяется, было ли приложение запущено в первый раз либо восстановлено после изменения конфигурации. Если значение переменной `savedInstanceState` будет `null` (строка 42), приложение запускается первый раз. В этом случае в строках 44–45 переменные `currentBillTotal` и `currentCustomPercent` инициализируются значениями, необходимыми в случае первой загрузки приложения. Если же приложение было восстановлено из памяти, в строке 50 вызывается метод `getString` объекта `savedInstanceState`, с помощью которого считывается сумма сохраненного итогового счета (в виде значения `double`). В строках 53–54 вызывается метод `getInt` объекта `savedInstanceState`, позволяющий получить доступ к сохраненной процентной ставке пользовательских чаевых (в виде значения `int`).

После «раздувания» XML-разметки с помощью метода `findViewById` класса `Activity` обеспечивается получение ссылок на отдельные виджеты. Этот метод получает константу типа `int` для специфического представления (GUI-компонент) и возвращает ссылку на нее. Название каждой константы из компонента GUI в классе `R.id` определяется с помощью атрибута `android:id` компонента GUI из файла `main.xml`. Например, значение константы `billEditText` будет `R.id.billEditText`.

В строках 58–63 обеспечивается получение ссылок на шесть компонентов `EditText`, в которых хранятся вычисленные чаевые (с процентными ставками 10 %, 15 % и 20 %),

а также итоговые суммы по счетам (с учетом чаевых). В строке 66 содержится ссылка на компонент `TextView`, которая обновляется в случае изменения пользователем процентной ставки пользовательских чаевых. В строках 69–71 находятся ссылки на компоненты `EditText`, в которых отображаются вычисленные пользовательские чаевые и итоговые суммы по счетам.

В строке 74 находится ссылка на `billEditText`, а в строке 77 вызывается метод `addTextChangedListener`. Этот метод регистрирует объект `TextChangedListener`, который отвечает на события, сгенерированные в случае изменения пользователем текста `billEditText`. Код этого объекта `listener` приведен в листинге 4.11.

В строке 80 находится ссылка на объект `customSeekBar`, а в строке 81 вызывается метод `setOnSeekBarChangeListener`, регистрирующий объект `OnSeekBarChangeListener`. Этот объект отвечает на события, которые генерируются в случае перемещения пользователем ползунка `customSeekBar`, с помощью которого изменяется процентная ставка пользовательских чаевых. Код этого объекта `listener` приводится в листинге 4.10.

Метод `updateStandard` из класса `TipCalculator`. Метод `updateStandard` (см. листинг 4.7) обновляет значения `EditText`, в которых находятся чаевые, вычисленные в соответствии с процентными ставками 10 %, 15 % и 20 %, а также итоговые чаевые. Обновление осуществляется при каждом изменении итогового счета. Метод использует значение `currentBillTotal` для вычисления сумм чаевых и итогового счета при выборе процентной ставки 10 % (строки 88–95), 15 % (строки 98–106) и 20 % (строки 109–116). Метод `static format` из класса `String` применяется для преобразования сумм чаевых и счетов в строки, которые отображаются соответствующими компонентами `EditText`.

Листинг 4.7. Метод `updateStandard` из класса `TipCalculator` вычисляет и отображает чаевые и итоговые счета на основе стандартных процентных ставок чаевых (10 %, 15 % и 20 %).

```
84 // вычисляет чаевые, хранящиеся в EditTexts, по ставкам 10, 15
85 и 20 %
86 private void updateStandard()
87 {
88     // вычисляет итоговый счет, включающий чаевые со ставкой 10 %
89     double tenPercentTip = currentBillTotal * .1;
90     double tenPercentTotal = currentBillTotal +
91     tenPercentTip;
92     // настройка текста tipTenEditText в соответствии с
93     tenPercentTip
94     tip10EditText.setText(String.format("    %.02f",
95     tenPercentTip));
96     // настройка текста totalTenEditText в соответствии с
97     tenPercentTotal
98     total10EditText.setText(String.format("    %.02f",
99     tenPercentTotal));
100    // вычисление общего итога с чаевыми 15 %
```

```

98  double fifteenPercentTip = currentBillTotal * .15;
99  double  fifteenPercentTotal  =  currentBillTotal  +
fifteenPercentTip; 100
101  // настройка текста tipFifteenEditText в соответствии
    // с fifteenPercentTip

102  tip15EditText.setText(String.format("    %.02f",
fifteenPercentTip)); 103
104  // настройка текста totalFifteenEditText в соответствии
    // с fifteenPercentTotal

105  total15EditText.setText(
106  String.format("    %.02f",
fifteenPercentTotal)); 107
108  // вычисление общего итога с чаевыми 20 %
109  double twentyPercentTip = currentBillTotal * .20;
110  double  twentyPercentTotal  =  currentBillTotal  +
twentyPercentTip; 111
112  // настройка текста tipTwentyEditText в соответствии
    //с twentyPercentTip

113  tip20EditText.setText(String.format("    %.02f",
twentyPercentTip)); 114
115  // настройка текста totalTwentyEditText в соответствии
    // с twentyPercentTotal

116  total20EditText.setText(String.format(" %.02f",
twentyPercentTotal));
117 } // конец метода updateStandard
118

```

Метод updateCustom из класса TipCalculator. Метод updateCustom (см. листинг 4.8) обновляет значения компонентов EditText, со- держащих значения пользовательских чаевых и итогового сета. При этом использу- ется процентная ставка, выбранная пользователем с помощью customSeekBar. В строке 123 настраивается текст customTipTextView таким образом, чтобы достигалось соот- ветствие с положением ползунка SeekBar. В строках 126–127 вычисляется значение customTipAmount. В строке 130 вычисляется customTotalAmount. В строках 133–135 выполняется преобразование customTipAmount и customTotalAmount в строки, которые отображаются в tipCustomEditText и totalCustomEditText соответственно.

Листинг 4.8. Метод updateCustom из класса TipCalculator вычисляет и отображает чаевые и итоговый счет в случае пользовательской ставки чаевых, выбираемой пользователем с помощью customSeekBar.

```

119 // обновляет компоненты EditText, включающие пользовательские
    // чаевые и итоги

120 private void updateCustom()
121 {
122  // настройка текста customTipTextView в соответствии с
    положением
    // SeekBar

123  customTipTextView.setText(currentCustomPercent + "
%"); 124
125  // вычисление суммы пользовательских чаевых
126  double customTipAmount =
127  currentBillTotal * currentCustomPercent * .01;

```

128

```
129    // вычисление итогового счета, включая пользовательские чаевые
130    double customTotalAmount = currentBillTotal +
customTipAmount; 131
132    // отображение суммы чаевых и итогового счета
133    tipCustomEditText.setText(String.format(" %.02f",
customTipAmount));
134    totalCustomEditText.setText(
135    String.format(" %.02f", customTotalAmount));
136 } // конец метода updateCustom
137
```

Переопределение метода onSaveInstanceState из класса Activity. В строках 139–146 (см. листинг 4.9) переопределяется метод onSaveInstanceState из класса Activity. Этот метод вызывается системой в случае изменения конфигурации устройства при выполнении приложения (например, при вращении устройства пользователем или выдвижении клавиатуры устройства). Чтобы сгенерировать этот метод в Eclipse, щелкните правой кнопкой мыши в области исходного кода и в контекстном меню выберите команды Source/Override/Implement Methods... (Исходный код/Переопределение/реализация методов). В появившемся диалоговом окне отображаются методы, которые могут быть переопределены или реализованы в классе. Установите флажок onSaveInstanceState, укажите место, в которое IDE включит код класса, и щелкните на кнопке ОК для создания оболочки метода.

Листинг 4.9. Переопределение метода onSaveInstanceState из класса Activity для сохранения состояния при изменении конфигурации приложения

```
138 // сохранение значений billEditText и customSeekBar
139 @Override
140 protected void onSaveInstanceState(Bundle outState)
141 {
142     super.onSaveInstanceState(outState);
143
144     outState.putDouble( BILL_TOTAL, currentBillTotal );
145     outState.putInt( CUSTOM_PERCENT, currentCustomPercent );
146 } // конец метода onSaveInstanceState
147
```

В разрабатываемом в этой главе приложении сначала вызывается метод onSaveInstanceState суперкласса, потом сохраняются пары «ключ/значение» в Bundle, которые потом передаются методу. В строке 144 сохраняется текущая сумма итогового счета, а в строке 145 сохраняется процентная ставка пользовательских чаевых (соответствует текущему положению ползунка SeekBar). Эти значения используются методом onCreate, который вызывается для восстановления приложения после изменения конфигурации. При рассмотрении других приложений из этой книги мы изучим множество других методов жизненного цикла Activity. Эти методы описаны на веб-сайте: bit.ly/ActivityLifecycle.

Анонимный внутренний класс, реализующий интерфейс OnSeekBarChangeListener. В строках 149–171 (см. листинг 4.10) создается анонимный объект внутреннего класса customSeekBarListener, который реагирует на события customSeekBar. Дополнительные сведения, касающиеся анонимных внутренних классов, можно найти на странице Oracle Java Tutorial bit.ly/AnonymousInnerClasses.

В строке 81 регистрируется `customSeekBarListener` в качестве объекта, выполняющего обработку событий и относящегося к `customSeekBar`. В строках 153–170 реализованы методы интерфейса `OnSeekBarChangeListener`.

Листинг 4.10. Анонимный внутренний класс, реализующий интерфейс `OnSeekBarChangeListener` в ответ на события, вызываемые `customSeekBar`.

```
148 // вызывается при изменении пользователем положения ползунка
SeekBar
149 private OnSeekBarChangeListener customSeekBarListener =
150     new OnSeekBarChangeListener()
151 {
152     // обновление currentCustomPercent, потом вызов updateCustom
153     @Override
154     public void onProgressChanged(SearchBar seekBar, int
        progress,
155         boolean fromUser)
156     {
157         // присваивание currentCustomPercent положения ползунка
        SeekBar
158         currentCustomPercent = seekBar.getProgress();
159         updateCustom(); // обновление EditTexts пользовательскими
            // чаевыми и итоговыми счетами
160     } // конец метода onProgressChanged
161
162     @Override
163     public void onStartTrackingTouch(SearchBar seekBar)
164     {
165     } // конец метода
onStartTrackingTouch 166
167     @Override
168     public void onStopTrackingTouch(SearchBar seekBar)
169     {
170     } // конец метода onStopTrackingTouch
171 }; // конец OnSeekBarChangeListener
172
```

Переопределение метода `onProgressChanged` интерфейса `OnSeekBarChangeListener`. В строках 153–160 переопределяется метод `onProgressChanged`. В строке 158 метод `getProgress` объекта `SeekBar` возвращает целочисленное значение, находящееся в диапазоне 0–100. Это значение соответствует положению ползунка объекта `SeekBar` и присваивается переменной `currentCustomPercent`. В строке 159 вызывается метод `updateCustom`, который используется в `customCurrentPercent` для вычисления и отображения пользовательских чаевых и итогового счета.

Переопределение методов `onStartTrackingTouch` и `onStopTrackingTouch` интерфейса `OnSeekBarChangeListener`. В Java нужно переопределять каждый реализованный метод интерфейса. И хотя в создаваемом приложении не используются эти методы интерфейса, чтобы выполнить правила Java, создадим пустую оболочку для каждого метода (строки 162–170).

Анонимный внутренний класс, реализующий интерфейс `TextWatcher`. В строках 174–206 листинга 4.11 создается объект внутреннего класса `billEditTextWatcher`, реагирующий на события, порождаемые `billEditText`. В строке 77 регистрируется `billEditTextWatcher` для прослушивания событий, порождаемых `billEditText`. В строках 177–205 реализуются методы интерфейса `TextWatcher`.

Листинг 4.11. Анонимный внутренний класс, реализующий интерфейс

TextWatcher, который используется для реагирования на события, порождаемые billEditText

```
173 // объект обработки событий, реагирующий на события
billEditText
174 private TextWatcher billEditTextWatcher = new TextWatcher()
175 {
176     // вызывается после ввода пользователем числа
177     @Override
178     public void onTextChanged(CharSequence s, int start,
179         int before, int count)
180     {
181         // преобразование текста billEditText в double
182         try
183         {
184             currentBillTotal =
185                 Double.parseDouble(s.toString());
186         } // завершение блока try
187         catch (NumberFormatException e)
188         {
189             currentBillTotal = 0.0; // по умолчанию в случае
190                                     // исключения
191         } // завершение блока catch
192
193         // обновление EditTexts, содержащих стандартные
194         // и пользовательские чаевые
195
196         updateStandard(); // обновление 10, 15 и 20 % EditTexts
197         updateCustom(); // обновление EditTexts
198                             // с пользовательскими чаевыми
199
200     } // конец метода onTextChanged
201
202     @Override
203     public void afterTextChanged(Editable s)
204     {
205         // конец метода afterTextChanged
206     }
207
208     @Override
209     public void beforeTextChanged(CharSequence s, int
210         start, int count,
211         int after)
212     {
213         // завершение метода beforeTextChanged
214     }
215 } // завершение billEditTextWatcher
216
217 } // завершение класса TipCalculator
```

Переопределение метода onTextChanged интерфейса TextWatcher. Метод onTextChanged (строки 177–194) вызывается в случае изменения текста, отображаемого компонентом billEditText. Этот метод принимает четыре параметра (строки 178–179). В рассматриваемом примере используется лишь параметр CharSequence s, который содержит копию текста billEditText. Другие параметры указывают на то, что текст, количество символов которого определено параметром count, начинающийся в позиции, указанной параметром start, заменил фрагмент прежнего текста, длина которого определяется параметром before.

В строке 184 выполняется преобразование текста, введенного пользователем в

компонент `billEditText`, в тип `double`. В строке 192 вызывается метод `updateStandard`, который выполняет обновление компонентов `EditText` 10 %, 15 % и 20 %, включающих сумму чаевых и итоговую сумму по счету (включая сумму чаевых). В строке 193 вызывается метод `updateCustom`, который обновляет компоненты `EditText`, включающие пользовательские чаевые и итоговый счет. При этом используется процентная ставка пользовательских чаевых, которая берется из `SeekBar`.

Методы `beforeTextChanged` и `afterTextChanged` интерфейса `billEditTextWatcherTextWatcher`. В этой главе не используются методы интерфейса `TextWatcher`. Поэтому эти методы переопределены пустыми методами (строки 196–205) для соответствия условиям разработки интерфейса.

4.4 Резюме

В этой главе вы создали свое первое интерактивное приложение Android — `Tip Calculator`. Сначала были вкратце рассмотрены функции этого приложения, потом в целях тестирования вычислялось значение стандартных и пользовательских чаевых на основе введенного счета. Вы выполнили подробные пошаговые инструкции по созданию графического интерфейса пользователя приложения с помощью инструментов ADT Plugin и среды Eclipse (редактор `Visual Layout Editor`, окна `Outline` и `Properties`).

При разработке графического интерфейса приложения используется компонент `TableLayout`, с помощью которого выполняется распределение компонентов GUI по строкам и столбцам. Вы узнали о том, что каждая ячейка компонента `TableLayout` может быть пустой или содержать один компонент. Одновременно с этим каждая ячейка может быть структурой, содержащей другие компоненты. Вы использовали компоненты `TableRow` для создания строк в макете и освоили методику определения количества столбцов с помощью `TableRow`, содержащих большинство компонентов. Также вы узнали о том, что высота каждой строки определяется высотой самого «длинного» компонента в строке, а ширина столбца — шириной самого «широкого» элемента столбца (если не выбрано растяжение столбцов). Компоненты `TextView` применялись для создания подписей GUI-компонентов, а `EditText` — для получения суммы итогового счета, введенной пользователем. Нефокусируемые компоненты `EditText` использовались для отображения различных значений чаевых и итогов, вычисляемых после выбора разных процентных ставок, а с помощью компонента `SeekBar` вводились пользовательские процентные ставки чаевых. Большая часть XML-разметки графического интерфейса пользователя была сгенерирована инструментами модуля ADT Plugin, но допускалось и непосредственное редактирование XML-разметки для настройки свойств, которые недоступны в окне `Properties`.

В процессе разработки приложения из этой главы был использован целый ряд возможностей объектно-ориентированного программирования на Java, включая классы, анонимные внутренние классы, объекты, методы, интерфейсы и наследование. Мы познакомились с «раздуванием» графического интерфейса пользователя из исходного XML-файла в экранное представление, освоили класс `Android Activity` и частично изучили жизненный цикл `Activity`. Мы рассмотрели метод `onCreate`, выполняющий инициализацию приложения при его запуске, и метод `onSaveInstanceState`, сохраняющий состояние приложения при изменении конфигурации устройства. В листингах главы использовался метод `findViewById` из класса `Activity`, который вызывается из метода `onCreate` и применяется для получения ссылок на компоненты GUI, с которыми приложение взаимодействует на программном уровне. Для объекта `billEditText` был определен анонимный внутренний класс, реализующий интерфейс `TextWatcher`. В результате приложение получило возможность вычислять новые значения чаевых и итогов после изменения текста `EditText` пользователем. Для компонента `customSeekBar` был определен анонимный

внутренний класс, который реализует интерфейс `OnSeekBarChangeListener`. В результате приложение получило возможность вычислять новую сумму пользовательских чаевых и итогового счета после изменения пользователем значения процентной ставки. Это изменение осуществляется перемещением ползунка компонента `SeekBar`.

5 Разработка приложения поисковых запросов в Твиттере

5.1 Введение

Разрабатываемое в данной главе приложение Favorite Twitter Searches помогает представить избранные (возможно длинные) строки поиска Твиттера в виде простых для запоминания, выбираемых пользователем коротких имен тегов. Это позволит пользователям следовать за твитами в своих любимых темах. Поисковые запросы Твиттера могут быть настроены с помощью поисковых операторов Твиттера (dev.twitter.com/docs/using-search). Но при этом следует иметь в виду, что более сложные запросы имеют большую длину, требуют больше времени и более чувствительны к ошибкам при вводе на мобильных устройствах. Избранные поисковые запросы пользователя сохраняются на устройстве и могут быть доступны при каждом запуске приложения. На рис. 5.1, а показано приложение с несколькими сохраненными поисковыми запросами. При этом допускается сохранение пользователем большого количества поисковых запросов и их прокрутка в алфавитном порядке. Поисковые запросы и соответствующие им теги вводятся в компоненты EditText, находящиеся в верхней части экрана. С помощью компонента Save Button (Кнопка сохранения) каждый поисковый запрос добавляется в список избранного. В результате касания кнопки поиска поисковый запрос отправляется Твиттеру, а результаты поиска отображаются на экране веб-браузера. На рис. 5.1, б показан результат касания кнопки Google Button (Кнопка Google) — поиск твитов в Google, указанных поиском Твиттера `from:Google`. Чтобы изменить настройки поиска, воспользуйтесь кнопками Edit Button (Кнопки изменения), которые находятся справа от каждой кнопки поиска. С помощью этих кнопок можно настроить результаты поиска таким образом, чтобы получить лучшие результаты (в случае предварительного сохранения поисковых запросов как избранных). После касания кнопки Clear Tags Button (Очистить теги), находящейся в нижней части экрана, удаляются все ранее сохраненные поисковые запросы, находящиеся в списке избранного. Потом нужно подтвердить удаление, выбрав соответствующий параметр в появившемся диалоговом окне [1].

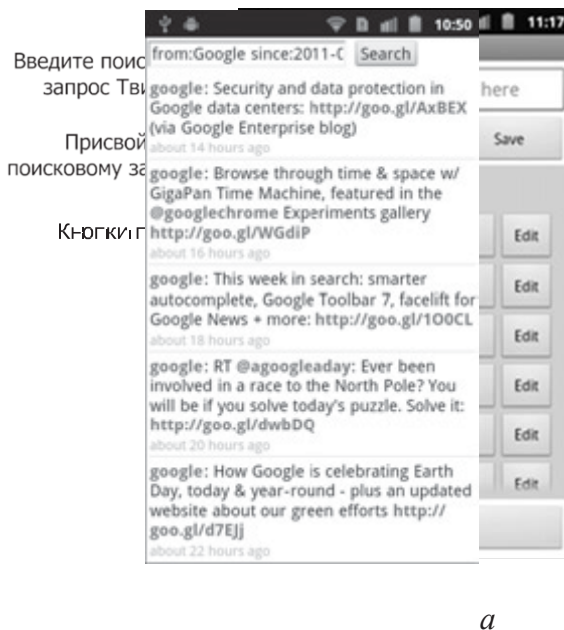


Рис. 5.1. Приложение Favorite Twitter Searches: а — приложение с несколькими сохраненными поисковыми запросами; б — результат касания кнопки

5.2 Тестирование приложения Favorite Twitter Searches

Открытие и выполнение приложения. Откройте интегрированную среду разработки Eclipse, импортируйте проект приложения Favorite Twitter Searches. Выполните следующие действия:

1. *Откройте диалоговое окно Import.* Чтобы открыть это диалоговое окно, выполните команды File→Import... (Файл→Импорт...).
2. *Импортируйте проект приложения Favorite Twitter Searches.* В диалоговом окне Import раскройте узел General (Общие) и выберите параметр Existing Projects into Workspace (Существующие проекты в рабочую область), потом щелкните на кнопке Next > (Далее >) для выполнения шага Import Projects (Импорт проектов). Установите флажок Select root directory (Выбор корневого каталога), потом щелкните на кнопке Browse... (Просмотр...). В диалоговом окне Browse For Folder (Просмотр папки) выберите папку FavoriteTwitterSearches, находящуюся в папке examples книги, и щелкните на кнопке ОК. Щелкните на кнопке Finish (Готово) для импорта проекта в Eclipse. После этого проект появится в окне Package Explorer, находящемся в левой части окна Eclipse.
3. *Запустите приложение Favorite Twitter Searches.* Для этого в среде Eclipse правой кнопкой мыши щелкните на проекте FavoriteTwitterSearches, находящемся в окне Package Explorer, потом в появившемся меню выберите команды Run As→Android Application (Выполнить как Приложение Android). В результате будет запущено на выполнение приложение Favorite Twitter Searches в среде AVD, которая создана и настроена. Окно выполняющегося приложения показано на рис. 5.2 [1].

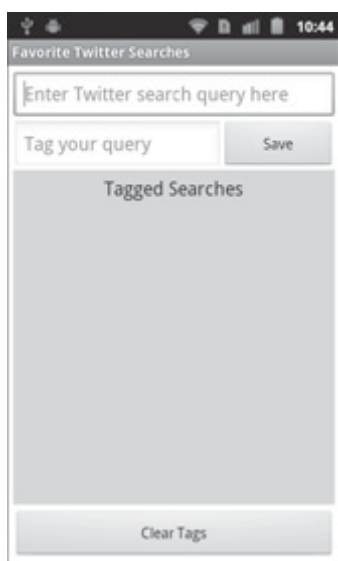
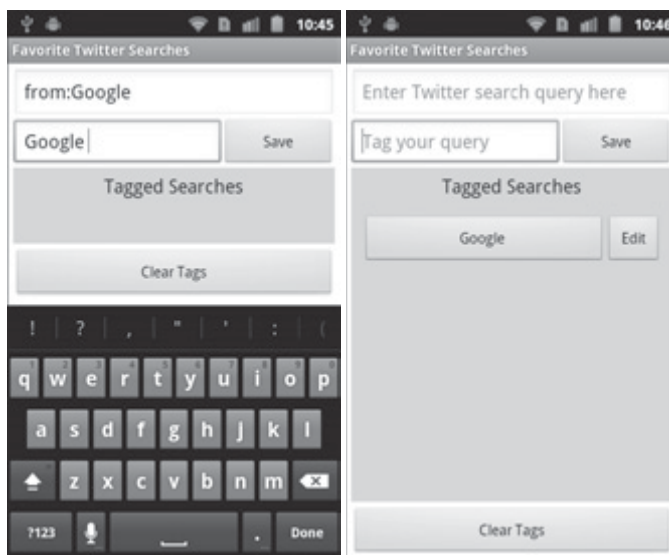


Рис. 5.2. Выполняющееся приложение Favorite Twitter Searches

Верхние два компонента EditText позволяют вводить новые поисковые запросы, а в разделе Tagged Searches (Тегированные поиски) отображаются ранее сохраненные поисковые запросы (в рассматриваемом примере подобные запросы отсутствуют).

Добавление нового избранного поиска. Введите фразу from:Google, задающую поисковый запрос, в верхний компонент EditText. В нижний компонент EditText введите Google (рис. 5.3, а). Введенное слово может использоваться в качестве краткого имени, отображающегося в разделе Tagged Searches. Чтобы сохранить поисковый запрос и скрыть экранную клавиатуру, нажмите кнопку Save Button. Под заголовком Tagged Searches появится кнопка Google Button (рис. 5.3, б). При этом также исчезнет экранная клавиатура. Это связано с тем, что данное приложение скрывает экранную клавиатуру программным образом.



а

б

Рис. 5.3. Ввод поискового запроса Твиттера: а — ввод поискового запроса Твиттера и тега поиска; б — окно приложения после сохранения поискового запроса и тега поиска

Изменение поискового запроса. Справа от каждой кнопки поиска находится кнопка редактирования (Edit Button). Коснитесь этой кнопки, и ваш запрос и тег поиска будут перезагружены в компоненты EditText, находящиеся в верхней части экрана приложения, и станут доступными для редактирования. В рассматриваемом примере ограничим поиск твитами, которые были созданы после 1 апреля 2011 года. Для этого в конец поискового запроса нужно добавить фразу `since:2011-04-01` (рис. 5.4). Нажмите кнопку Save для обновления сохраненного поискового запроса. 5.3.

ПРИМЕЧАНИЕ. Если изменить название тега, будет создана новая кнопка поиска. Эта возможность весьма полезна для создания нового запроса на основе ранее сохраненного.

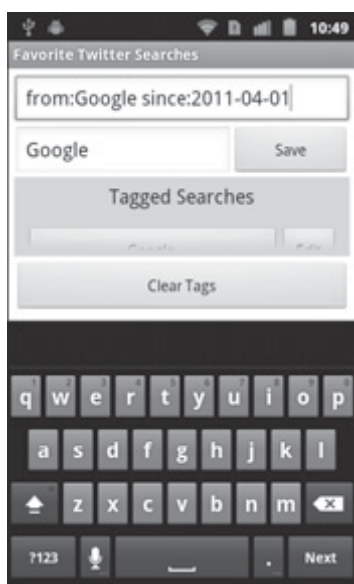


Рис. 5.4. Редактирование строки поиска в Твиттере

Просмотр результатов поиска в Твиттере. Чтобы просмотреть результаты поиска, нажмите кнопку Google search query Button. В результате откроется окно веб-

браузера, и вы получите доступ к веб-сайту Твиттера и сможете получить и отобразить результаты поиска (рис. 5.5).

5.3 Обзор применяемых технологий

Приложение использует GUI-компоненты `EditText`, `ScrollView` и `Button`. Компонент `ScrollView` представляет собой объект `ViewGroup`, который может включать другие представления (`Views`), например макет (`layout`). Этот компонент обеспечивает прокрутку контента при его просмотре на экране. С помощью компонента `ScrollView` можно отображать на экране любые по величине списки сохраненных поисковых запросов, среди которых будут избранные запросы, которые могут помещаться на экране. Каждый поисковый запрос связан с компонентом `Button` — коснитесь этого компонента, чтобы начать поиск с помощью браузера.

SharedPreferences. В вашем распоряжении может быть один или несколько файлов, включающих пары «ключ/значение», которые связаны с каждым приложением. Эта особенность используется для манипулирования файлами, которые называются поисками. В этих файлах хранятся пары тегов и поисковые запросы Твиттера, созданные пользователем. Для считывания пар «ключ/значение» из файла используются объекты `SharedPreferences` (пакет `android.content`). Чтобы изменить содержимое файла, используются объекты `SharedPreferences.Editor` (пакет `android.content`). Ключи должны иметь строковый тип, а значения могут иметь строковый или примитивный тип.

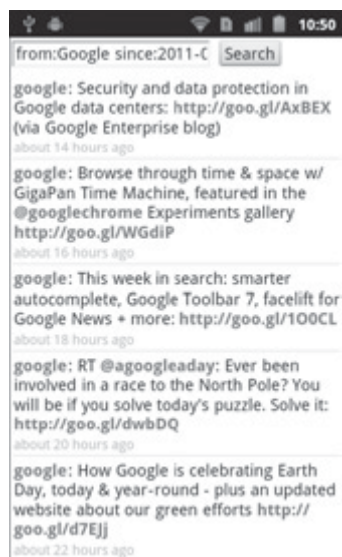


Рис. 5.5. Просмотр результатов поиска

С помощью метода `refreshButtons` осуществляется считывание данных в сохраненных поисках. Этот метод вызывается из метода `onCreate`, относящегося к классу `Activity`. Эта методика применяется, когда загруженный объем данных не слишком велик. После загрузки приложения Android создает основной поток, который называется `UI thread`. В этом потоке реализуются выполняемые GUI операции. Обратите внимание, что в потоке `UI thread` не рекомендуется выполнять интенсивные операции ввода/вывода, поскольку это может привести к резкому ухудшению отзывчивости приложения.

Интенты. Обычно интенты используются для запуска действий. Они указывают выполняемое действие и данные, по отношению к которым это действие будет выполнено. Как только пользователь коснется объекта `Button`, представляющего поиск, создается URL-ссылка, которая включает поисковый запрос Твиттера. С помощью создания нового интента (`Intent`) выполняется загрузка URL-ссылки в веб-браузер, потом интент передается методу `startActivity`, который класс `Activity` косвенным образом наследует из класса `Context`. Для просмотра URL-ссылки метод `startActivity` запускает веб-

браузер устройства, в окне которого отображается содержимое. В данном приложении в качестве содержимого используются результаты поиска в Твиттере.

LayoutInflater. После ввода пользователем нового поискового запроса в пользовательский интерфейс добавляется новый ряд объектов Button. Один объект Button представляет поиск, а второй позволяет изменять поиск. Для программного создания этих компонентов GUI на основе заранее определенной XML-разметки используется компонент LayoutInflater. Компонент LayoutInflater «раздувает» файл XML-разметки, в результате чего создаются компоненты, которые были специфицированы в XML-коде. Затем определяется поисковый текст объекта Button, регистрируются обработчики событий каждого объекта Button и новые компоненты GUI связываются с пользовательским интерфейсом.

AlertDialog. Перед сохранением нового поискового запроса нужно, чтобы пользователь ввел сам запрос и тег. Если этого не произойдет и компонент EditText останется пустым, отобразится соответствующее сообщение для пользователя. Также требуется подтверждение пользователем удаления всех поисковых запросов после касания кнопки Clear Tags (Очистить теги). Необходимые сообщения и подтверждения отображаются с помощью объектов AlertDialog. Во время отображения этого диалогового окна (модальное диалоговое окно) блокируется взаимодействие пользователя с приложением. В следующих разделах будет показано, каким образом определять настройки для подобных диалоговых окон с помощью объекта AlertDialog.Builder, а затем с его помощью создавать AlertDialog.

Файл AndroidManifest.xml. Файл AndroidManifest.xml генерируется при создании приложения с помощью подключаемого модуля ADT Plugin в среде Eclipse. В этом файле определяются настройки, например название приложения, имя пакета, целевая SDK и минимальная версия SDK, имя объекта Activity для приложения и ряд других настроек. В конце главы вас ожидает обзор структуры этого файла. Также вы узнаете, каким образом в манифест добавить новую настройку, предотвращающую отображение виртуальной клавиатуры после первой загрузки приложения.

5.4 Создание графического интерфейса приложения и файлов ресурсов

В этом разделе будет создан графический интерфейс пользователя для приложения Favorite Twitter Searches. Вашему вниманию будет представлен код XML, сгенерированный модулем ADT Plugin для макета приложения. В первую очередь будут рассмотрены новые возможности и функции GUI, представлена завершенная XML-разметка и рассмотрены ключевые фрагменты XML-кода. Будет также создана вторая XML-разметка, динамическое «раздувание» которой создаст теги и кнопки Edit для каждого поискового запроса. В результате приложение сможет загружать ранее сохраненные поисковые запросы и менять внешний вид в процессе добавления или удаления поисковых запросов пользователем.

Компонент main.xml TableLayout. Как и в главе 4, для создания основного макета приложения используется компонент TableLayout (рис. 5.6), который в данном случае включает пять строк и два столбца. Атрибуту android:stretchColumns компонента TableLayout присвоено значение "*". В результате все строки таблицы будут «растягиваемыми», то есть элементы в каждом столбце могут расширяться на весь экран.

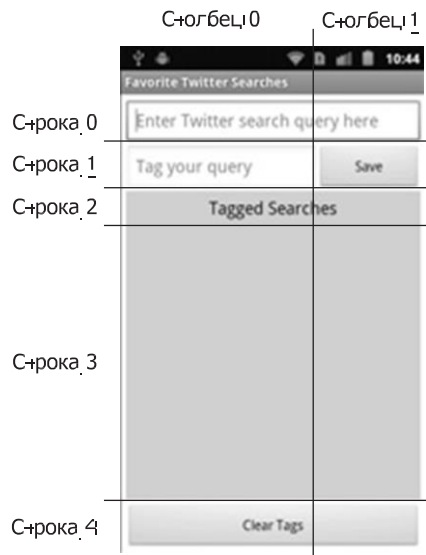


Рис. 5.6. Строки и столбцы компонента TableLayout приложения Favorite Twitter Searches

На рис. 5.7 показаны названия компонентов GUI. При создании этих имен мы руководствовались соглашением о наименовании. Суть этого соглашения заключается в использовании имени класса GUI-компонента для каждого свойства Id компонента в XML-разметке и в каждом имени переменной в коде Java.

Создание проекта. Создайте новый проект Android и присвойте ему имя FavoriteTwitterSearches. В диалоговом окне New Android Project (Новый проект Android) укажите значения следующих параметров и нажмите кнопку Finish (Готово):

- Build Target (Операционная система): Android 2.3.3.
- Application name (Имя приложения): Favorite Twitter Searches.
- Package name (Название пакета): com.deitel.favoritetwittersearches.
- Create Activity (Создать деятельность): FavoriteTwitterSearches.
- Min SDK Version (Минимальная версия SDK): 10.

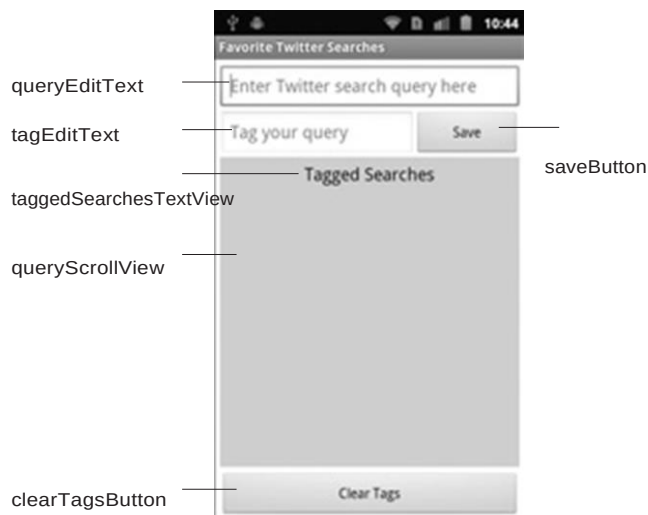


Рис. 5.7. Компоненты GUI приложения Favorite Twitter Searches именуются с помощью значений свойства Id

ПРИМЕЧАНИЕ. Эта версия SDK соответствует версии Android 2.3.3, хотя в этом приложении не будут использоваться функции, присущие именно Android 2.3.3. Если планируется выполнять это приложение на виртуальном устройстве Android (AVD) либо на реальном устройстве с более ранней версией Android, параметру Min SDK присвойте меньшее значение. Например, если выбрано значение 8 для этого параметра, приложение будет выполняться на версии Android 2.2 или более старшей.

Создание файлов ресурсов. В этом приложении литеральное значение цвета и несколько литеральных значений размеров хранятся в файлах `colors.xml` и `dimen.xml` соответственно. Подобные имена файлов используются по соглашению, а сами файлы находятся в папке приложения `res/values`. Цвет и размеры, хранящиеся в этих файлах, будут представлены в автоматически сгенерированном файле `R.java` с помощью константы, которую можно использовать в качестве ссылки на определенное значение. Чтобы создать каждый из перечисленных выше файлов, выполните следующие действия:

1. Щелкните правой кнопкой мыши на имени проекта в окне Package Explorer и выполните команды `New > Other...` (Создать > Другой...), затем в диалоговом окне `New` (Создать) в узле `Android` выберите параметр `Android XML File` (XML-файл Android). На экране появится диалоговое окно `New Android XML File` (Создать XML-файл Android).
2. В текстовое поле `File` (Файл) введите имя `colors.xml`.
3. В разделе `What type of resource would you like to create?` (Укажите тип создаваемого ресурса) выберите переключатель `Values` (Значения). После этого в папке проекта `res/values` появится новый файл.
4. Для завершения создания файла щелкните на кнопке `Finish` (Готово).
5. Повторите описанный выше процесс для создания файла `dimen.xml`.

Содержимое этих двух файлов показано в листингах 5.1–5.2. В XML-разметке проекта из этой главы используются значения цвета и размера, заданные в этих файлах. Также используются несколько предварительно определенных цветов Android из класса `R.color`. Как и в предыдущих приложениях, в файле `strings.xml` будут заданы многочисленные строковые ресурсы.

Файл `colors.xml`. XML-документы, представляющие ресурсы, должны включать элемент `resources`, с помощью которого определяются ресурсы. Внутри этого элемента (см. листинг 5.1) определяется единственное значение цвета, используемое в приложении (`light_orange`). Элемент `color` (строка 3) определяет атрибут `name`, который используется для ссылки на цвет, и шестнадцатеричное значение, определяющее цвет.

Листинг 5.1. Цвета, определенные в файле `colors.xml`

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <resources>
3   <color name="light_orange">#8f90</color>
4 </resources>
```

Файл `dimen.xml`. В листинге 5.2 определяются элементы `dimen`, представляющие значения ширины тега поиска и кнопок `Edit`. Благодаря определению размеров в форме ресурсов можно использовать пиксельные значения, независимые от плотности (`dp` или `dip`), и пиксельные значения, независимые от масштаба (`sp`). В результате Android выполняет автоматическое преобразование в соответствующие пиксельные значения для выбранного устройства. В коде используются лишь фиксированные пиксельные размеры, в результате чего у пользователя появляется возможность вычисления подходящих пиксельных значений для каждого устройства вручную.

Листинг 5.2. Размеры, определенные в файле `dimen.xml`

```
5 <?xml version="1.0" encoding="UTF-8"?>
6 <resources>
7   <dimen name="tagButtonWidth">230dp</dimen>
8   <dimen name="editButtonWidth">50dp</dimen>
9 </resources>
```

Файл `strings.xml`. В листинге 5.3 определяются строковые литеральные значения, используемые в приложении. В строке 4 определяется значение `searchURL`. Поисковые

запросы пользователя добавляются к этой URL-ссылке перед отображением поиска Твиттера на экране веб-браузера устройства.

Листинг 5.3. Строки, определенные в файле strings.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <resources>
3   <string name="app_name">Favorite Twitter Searches</string>
4   <string
5     name="searchURL">http://search.twitter.com/search?q=
6   <string name="tagPrompt">Назначьте тег запросу</string>
7   <string name="queryPrompt">Введите поисковый запрос
8   <string name="taggedSearches">Тегированные поиски</string>
9   <string name="edit">Изменить</string>
10  <string name="clearTags">Очистить теги</string>
11  <string name="save">Сохранить</string>
12  <string name="erase">Удалить</string>
13  <string name="cancel">Отмена</string>
14  <string name="OK">OK</string>
15  <string name="missingTitle">Текст отсутствует</string>
16  <string name="missingMessage">
17    Введите текст запроса и назначьте ему тег.</string>
18  <string name="confirmTitle">Вы уверены?</string>
19  <string name="confirmMessage">
20    Сейчас будут удалены все сохраненные поисковые
    запросы</string>
21 </resources>
```

5.5 Добавление класса TableLayout и компонентов

С помощью технологий, описанных в главе 4, будет создан графический интерфейс пользователя (см. рис. 5.6–5.7). Начнем с создания базовой структуры и элементов управления, потом настроим свойства элементов управления для завершения проекта. После добавления компонентов в каждую строку TableLayout установим значения свойств Id и Text этих компонентов (см. рис. 5.7). В процессе построения GUI поместите литеральные строковые значения в файл strings.xml, находящийся в папке res/values приложения. С помощью окна Outline добавьте компоненты в требуемые TableRows макета TableLayout [1].

Шаг 1. Удаление и повторное создание файла main.xml. Для создаваемого в этой главе приложения вместо стандартного файла main.xml воспользуйтесь другим подобным файлом с объектом TableLayout, позволяющим реализовать относительное расположение компонентов. Чтобы заменить стандартный файл main.xml, выполните следующие действия:

1. Чтобы удалить файл main.xml, щелкните на нем (в папке проектов /res/layout) и в контекстном меню выберите параметр Delete (Удалить).
 2. Правой кнопкой мыши щелкните на папке layout и выберите команды New > Other...
 3. (Создать > Другое...). Отобразится диалоговое окно New (Создать).
 4. В узле Android выберите параметр Android XML File (XML-файл Android). После щелчка на кнопке Next > (Далее >) появится диалоговое окно New Android XML File (Создать XML-файл Android).
 5. Выберите название файла main.xml и выделите TableLayout, потом щелкните на кнопке
1. Finish.

Шаг 2. Конфигурирование Visual Layout Editor для использования подходящей библиотеки Android SDK. В раскрывающемся списке селектора SDK, находящемся в правой верхней части вкладки Graphical Layout, выберите параметр Android 2.3.3 (как на рис. 3.7). В результате разрабатываемый графический интерфейс будет совместим с устройствами Android 2.3.3.

Шаг 3. Настройка размеров и разрешения окна Visual Layout Editor. В раскрывающемся списке Device Configurations (Конфигурации устройств), находящемся в левой верхней части вкладки Graphical Layout (см. рис. 3.11), выберите параметр 3.7in WVGA (Nexus One). В результате будет настроена область конфигурирования для устройств с экраном, имеющим разрешение 480×800 (WVGA).

Шаг 4. Конфигурирование компонента TableLayout. В окне Outline выберите параметр TableLayout и установите следующие значения:

- Background (Фон): @android:color/white;
- Id (Идентификатор): @+id/tableLayout;
- Padding (Отступ): 5dp;
- Stretch columns (Растяжение столбцов): *.

Цвет Background определяется с помощью одного из заранее заданных цветовых значений Android, white (белый), находящихся в класса R.color. Названия цветов, определенных в этом классе, можно найти на веб-сайте developer.android.com/reference/android/R.color.html.

Чтобы получить доступ к заранее определенному ресурсу цвета, воспользуйтесь инструкцией @android:color/, а потом укажите название ресурса.

По умолчанию макет занимает весь экран, поскольку свойствам Layout width и Layout height присвоено значение match_parent. Параметру Padding присваивается значение 5dp, в результате чего вдоль границы всего макета создается кайма, шириной в 5 независимых от плотности пикселей. Значение, присвоенное параметру Stretch, определяет растяжение столбцов по горизонтали на всю ширину макета.

Шаг 5. Добавление компонентов TableRow. С помощью окна Outline добавьте в TableLayout пять компонентов TableRow (см. главу 4). Перед добавлением очередного TableRow выделяйте TableLayout, чтобы обеспечить корректное вложение TableRows в TableLayout. Измените свойства Id для пяти компонентов TableRow, выбрав значения tableRow0, tableRow1, tableRow2, tableRow3 и tableRow4 соответственно. Также выделите каждый компонент TableRow и присвойте его свойству Layout width значение match_parent. В результате строки растянутся на всю ширину макета.

Шаг 6. Добавление компонентов в TableRows. Используя рис. 5.6 и 5.7 в качестве руководства, добавьте в макет EditText, Button, TextView и ScrollView. Также поместите элемент TableLayout внутри компонента ScrollView. Присвойте элементам имена (см. рис. 5.7). Просмотрите XML-элементы в файле main.xml (листинг 5.4), чтобы получить представление о значениях, присвоенных атрибутам для каждого GUI-компонента. Новые и ключевые функции, определенные в листинге 5.4, будут рассмотрены позднее.

Листинг 5.4. XML-разметка приложения Favorite Twitter Search

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <TableLayout
  xmlns:android=http://schemas.android.com/apk/res/android
3   android:id="@+id/tableLayout"
  android:layout_width="match_parent"
4   android:layout_height="match_parent" android:padding="5dp"
5   android:stretchColumns="*"
  android:background="@android:color/white">
6
```

```

7  <!-- tableRow0 -->
8  <TableRow android:id="@+id/tableRow0"
9      android:layout_height="wrap_content"
10     android:layout_width="match_parent">
11     <EditText  android:layout_width="match_parent"
12         android:layout_height="wrap_content"
13         android:layout_span="2"
14         android:inputType="text"
15         android:id="@+id/queryEditText"
16         android:hint="@string/queryPrompt"
17         android:imeOptions="actionNext">
18     </EditText>
19 </TableRow>

19 <!-- tableRow1 -->
20 <TableRow android:id="@+id/tableRow1"
21     android:layout_height="wrap_content"
22     android:layout_width="match_parent">
23     <EditText  android:layout_height="wrap_content"
24         android:hint="@string/tagPrompt"
25         android:inputType="text"
26         android:id="@+id/tagEditText"
27         android:imeOptions="actionDone"
28         android:layout_gravity="center_vertical"></EditText>
29     <Button  android:id="@+id/saveButton"
30         android:layout_height="wrap_content"
31         android:layout_width="wrap_content"
32         android:layout_gravity="center_vertical"
33         android:text="@string/save"></Button>
34 </TableRow>

34 <!-- tableRow2 -->
35 <TableRow android:id="@+id/tableRow2"
36     android:layout_height="wrap_content"
37     android:layout_width="match_parent"
38     android:background="@color/light_orange">
39
40     <TextView  android:layout_height="wrap_content"
41         android:id="@+id/taggedSearchesTextView"
42         android:text="@string/taggedSearches"
43         android:layout_width="match_parent"
44         android:layout_gravity="center_horizontal"
45         android:layout_span="2"  android:textSize="18sp"
46         android:textColor="@android:color/black"
47         android:padding="5dp"></TextView>
48 </TableRow>
49
50 <!-- tableRow3 -->
51 <TableRow android:id="@+id/tableRow3"
52     android:background="@color/light_orange"
53     android:layout_height="wrap_content"
54     android:layout_width="match_parent">
55     android:layout_weight="1"
56
57     <ScrollView  android:id="@+id/queryScrollView"

```

```

57         android:layout_width="match_parent"
58         android:layout_span="2"         android:padding="5dp">
59         <TableLayout android:id="@+id/queryTableLayout"
60             android:layout_width="match_parent"
61             android:layout_height="match_parent"
62             android:padding="5dp"
63             android:stretchColumns="*">         </TableLayout>
64         </ScrollView>
65     </TableRow>
66
67     <!-- tableRow4 -->
68     <TableRow android:id="@+id/tableRow4"
69         android:layout_height="wrap_content"
70         android:layout_width="match_parent">
71
72         <Button         android:layout_width="wrap_content"
73             android:layout_height="wrap_content"
74             android:text="@string/clearTags"
75             android:id="@+id/clearTagsButton"
76             android:layout_span="2"
77             android:layout_marginTop="5dp"></Button>
78     </TableRow>
79 </TableLayout>

```

Ключевые компоненты файла main.xml. В главе 4 значение атрибута `android:layout_span` (строки 12, 45, 58 и 75) устанавливалось непосредственно в XML-коде, поскольку этот атрибут не отображается в окне Properties, в котором выбран режим проектирования. Ресурсы, находящиеся в файлах `colors.xml`, `dimen.xml` и `strings.xml`, которые используются для настройки различных свойств компонентов GUI, будут описаны в следующем списке. Чтобы получить доступ к различным значениям ресурсов в XML-коде, выполните следующие действия:

- **Строки.** Укажите инструкции `@string/`, которые следуют за именем ресурса. Например, в строках 14 и 31 заданы значения строкового ресурса для атрибута `android:hint`, соответствующего компонентам `EditText`. Этот атрибут отображает внутри элемента `EditText` подсказку, которая поможет пользователю осознать предназначение компонента `EditText`. Другие строковые ресурсы используются для представления текста в различных GUI-компонентах, например `Button` (строки 31 и 73) и `TextView` (строка 41).
- **Цвета.** Воспользуйтесь инструкцией `@color/`, которая следует после названия ресурса. Например, в строках 38 и 52 определяется ресурс цвета, определяющий фоновый цвет для компонентов `tableRow2` и `ScrollView` соответственно.

В строках 15 и 25 используется атрибут `android:imeOptions` компонента `EditText`, с помощью которого настраиваются параметры для текущего метода ввода. Например, если элемент `queryEditText` получает фокус и отображается виртуальная клавиатура, эта клавиатура содержит кнопку Next (Далее). Эта кнопка определяется путем присваивания атрибуту `android:imeOptions` значения `actionNext` (строка 15). Если пользователь касается этой кнопки, фокус перемещается к следующему компоненту, который принимает пользовательский ввод, — `tagEditText`. Как только компонент `tagEditText` получает фокус, на виртуальной клавиатуре появляется кнопка Done (Готово). Эта кнопка определяется путем присваивания атрибуту `android:imeOptions` значения `actionDone` (строка 25). Как только пользователь касается этой кнопки, система скрывает виртуальную клавиатуру.

В строках 27–31 и 71–75 определяются компоненты `Button`, применяемые для сохранения и очистки всех ранее сохраненных поисковых запросов соответственно. В строках 56–63 определяется компонент `ScrollView`, включающий `TableLayout` (строки 59–62), которые отображают кнопки поиска (причем отображение задается программно). Атрибуту `android:stretchColumns` компонента `TableLayout` присваивается значение `"*"`, в результате чего содержимое каждого компонента `TableRow`, которое программным образом помещается в `TableLayout`, может растягиваться на всю ширину макета. Если имеется несколько кнопок поиска, отображаемых на экране, можно выполнить перетаскивание экрана пальцем вверх или вниз в области компонента `ScrollView`, чтобы выполнить прокрутку объектов `Button` в `TableLayout`. Как показано в разделе 5.5, компонент `TableLayout` будет включать `TableRows`, каждый из которых содержит кнопки поиска и редактирования.

В строке 54 атрибуту `android:layout_weight` компонента `tableRow3` присваивается значение 1. В результате компоненту `tableRow3` присваивается высший приоритет по сравнению с другими строками в процессе изменения размеров основной структуры таблицы, выполняемого при наличии доступного пространства. Поскольку атрибут `android:layout_weight` определяется лишь для компонента `tableRow3`, этот компонент растягивается по вертикали, занимая все свободное место, в котором отсутствуют другие строки.

Создание компонента `TableRow`, отображающего кнопки `Search` и `Edit`. На этом этапе будет определен компонент `TableRow`, который будет «раздут», чтобы программным образом создать каждую кнопку поиска и соответствующую ей кнопку редактирования. В разделе 5.5 рассматриваются методики, используемые для отображения кнопок путем конфигурирования `Button` и добавления элементов `TableRow` в компонент `queryTableLayout` (см. листинг 5.4, строки 59–62). Чтобы создать другой файл XML-разметки, выполните следующие действия:

1. Щелкните правой кнопкой мыши на папке `layout` и в контекстном меню выберите параметры `New > Other...` для отображения диалогового окна `New`.
2. В узле `Android` выберите параметр `Android XML File` и щелкните на кнопке `Next >` для отображения диалогового окна `New Android XML File`.
3. В текстовое поле `File (Файл)` введите имя `new_tag_view.xml`.
4. В разделе `What type of resource would you like to create?` (Тип создаваемого ресурса) установите переключатель `Layout (Макет)`. В результате новый файл `new_tag_view.xml` запишется в папку проекта `res/layout`.
5. В нижней части диалогового окна выберите корневой элемент для нового макета.
6. Выберите компонент `TableRow`.
7. Щелкните на кнопке `Finish` для создания файла, который незамедлительно откроется в представлении XML.
8. Выберите вкладку `Graphical Layout` (Графический макет) в окне редактора `Visual Layout Editor`, в раскрывающемся списке `SDK Selector`, находящемся в правой части вкладки `Graphical Layout`, выберите параметр `Android 2.3.3`. В раскрывающемся списке `Device Configurations` (Конфигурации устройства), находящемся в верхней левой части вкладки `Graphical Layout`, выберите параметр `3.7in WVGA (Nexus One)`.

Добавьте в макет два объекта `Button`. Сконфигурируйте свойства `Button` и `layout` (см. листинг 5.5). Атрибуту `android:text` значение `newTagButton` не присваивается, поскольку это значение присваивается определенному тегу поиска во время программного создания объектов `Button`. Атрибуту `android:background` компонента `TableLayout` присваивается заранее определенный цвет `transparent` (прозрачный), как указано в строке 6. В результате появится возможность «смотреть сквозь» фоновый цвет компонента `ScrollView` при связывании элемента `TableRow` с `ScrollView`. По умолчанию компонент `ScrollView` имеет тот же фоновый цвет, что и родительский элемент (в данном случае

tableRow3). В строках 9 и 12 конструкция @dimen/ следует после имени ресурса размеров, определяя ширину объектов Button.

Листинг 5.5. Компонент newTagTableRow, который «раздувается» программным образом

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <TableRow
  xmlns:android=http://schemas.android.com/apk/res/android
3   android:id="@+id/newTagTableRow"
4   android:layout_width="match_parent"
5   android:layout_height="wrap_content"
6   android:background="@android:color/transparent">
7
8   <Button android:id="@+id/newTagButton"
9     android:layout_width="@dimen/tagButtonWidth"
10    android:layout_height="wrap_content"></Button>
11   <Button android:id="@+id/newEditButton"
12     android:layout_width="@dimen/editButtonWidth"
13     android:layout_height="wrap_content"
14     android:text="@string/edit"></Button>
15 </TableRow>
```

5.6 Создание приложения

В листингах 5.6–5.16 представлен код приложения Favorite Twitter Searches, реализованного в единственном классе FavoriteTwitterSearches, который расширяет класс Activity [1].

Операторы package и import. В листинге 5.6 приведены операторы package и import, используемые в приложении. С помощью оператора package (строка 4) определяется, что класс, находящийся в этом файле, является частью пакета com.deitel.favoritetwittersearches. Эта строка включается автоматически средой разработки (IDE) при создании проекта. Операторы import (строки 6–23) импортируют различные классы и интерфейсы, используемые приложением.

Листинг 5.6. Операторы package и import, используемые в приложении FavoriteTwitterSearches

```
1 // FavoriteTwitterSearches.java
2 // Хранит поисковые запросы Твиттера и теги, облегчающие их
   открытие
3 // в окне браузера.
4 package com.deitel.favoritetwittersearches;
5
6   import java.util.Arrays;
7
8   import android.app.Activity;
9   import android.app.AlertDialog;
10  import android.content.Context;
11  import android.content.DialogInterface;
12  import android.content.Intent;
13  import android.content.SharedPreferences;
14  import android.net.Uri;
15  import android.os.Bundle;
16  import android.view.LayoutInflater;
17  import android.view.View;
```



```

18 import android.view.View.OnClickListener;
19 import android.view.inputmethod.InputMethodManager;
20 import android.widget.Button;
21 import android.widget.EditText;
22 import android.widget.TableLayout;
23 import android.widget.TableRow;
24

```

В строке 6 импортируется класс `Arrays` из пакета `java.util`. С помощью метода `sort` этого класса будут сортироваться теги, представляющие каждый поиск. В результате сохраненные поисковые запросы отображаются в алфавитном порядке. При этом используются операторы `import`, относящиеся к классам, рассматриваемым в этой главе.

- Класс `AlertDialog` из пакета `android.app` (строка 9) применяется для отображения диалоговых окон.
- Класс `Context` из пакета `android.content` (строка 10) обеспечивает доступ к информации о среде, в которой выполняется приложение, а также обеспечивает доступ к различным службам Android. Константа из этого класса вместе с объектом `LayoutInflater` (рассматривается позднее) для упрощения динамической загрузки новых компонентов GUI.
- Класс `DialogInterface` из пакета `android.content` (строка 11) включает вложенный интерфейс `OnClickListener`. С помощью этого интерфейса обрабатываются события, которые имеют место после касания пользователем кнопки `AlertDialog`.
- Класс `Intent` из пакета `android.content` (строка 12) обеспечивает возможность работы с интендами (`Intents`). С помощью класса `Intent` определяется выполняемое действие и обрабатываемые данные — в Android объекты `Intent` применяются для запуска со-ответствующих деятельности.
- Класс `SharedPreferences` из пакета `android.content` (строка 13) применяется для опек-рирования устойчивыми парами «ключ/значение», которые сохранены в файлах, связанных с приложением.
- Класс `Uri` из пакета `android.net` (строка 14) позволяет преобразовывать URL-ссылки из Интернета в формат, требуемый классу `Intent` для запуска веб-браузера устройства. Дополнительные сведения относительно URI- и URL-ссылок приведены в разделе 5.5.
- Класс `LayoutInflater` из пакета `android.view` (строка 16) позволяет динамически «раз-дувать» файл XML-разметки, чтобы создавать GUI-компоненты макета.
- Класс `InputMethodManager` из пакета `android.view.inputmethod` (строка 19) обеспечивает скрывание виртуальной клавиатуры после сохранения поискового запроса пользователем.
- Пакет `android.widget` (строки 20–23) включает виджеты (компоненты GUI) и ма-кетки, которые используются в Android GUI. Класс `Button` из пакета `android.widget` (line 20) представляет простую кнопку, после нажатия которой выполняется опреде-ленное действие. Интерфейс `View.OnClickListener` из пакета `android.view` (строка 18) используется для определения кода, вызываемого после касания пользователем компонента `Button`.

Класс `Activity` для приложения `Favorite Twitter Searches`. Класс `FavoriteTwitterSearches` (см. листинги 5.7–5.16) — это единственный класс `Activity` для приложения `Favorite Twitter Searches`. После создания проекта `FavoriteTwitterSearches` модуль ADT Plugin генерирует этот класс в виде подкласса класса `Activity` (см. листинг 5.7, строка 26) и поддерживает оболочку для переопределенного метода `onCreate`. Этот метод должен быть переопределен для каждого подкласса из класса `Activity`.

Листинг 5.7. Класс FavoriteTwitterSearches является подклассом класса Activity

```
25 // главный (и единственный) класс Activity приложения
    // Favorite Twitter Searches

26 public class FavoriteTwitterSearches extends
Activity 27 {
28     private SharedPreferences savedSearches; // избранные
        // поисковые запросы пользователя

29     private TableLayout queryTableLayout; // кнопки поиска
30     private EditText queryEditText; // ввод запросов
        пользователя
31     private EditText tagEditText; // ввод тега
        поискового запроса
32
```

В строке 28 объявляется экземпляр переменной savedSearches из объектов SharedPreferences. В объектах SharedPreferences хранятся пары «ключ/значение», при чем ключи имеют строковый тип данных, а значения могут иметь примитивный или строковый тип данных. Объект SharedPreferences применяется для хранения сэкономленных пользователем поисковых запросов. В строке 29 объявляется объект TableLayout, используемый для получения доступа к части GUI, в которой программным методом отображаются новые кнопки. В строках 30–31 объявляются два компонента EditText, используемые для получения доступа к запросам и тега, введенным пользователем в верхнюю часть окна приложения.

Переопределенный метод onCreate из класса Activity. Метод onCreate (см. листинг 5.8) вызывается системой в следующих случаях:

- при загрузке приложения;
- если процесс приложения был «убит» операционной системой, пока приложение находилось в фоновом режиме, а затем выполнение приложения было возобновлено;
- при каждом изменении конфигурации, например при вращении устройства пользователем либо при выдвижении/скрытии физической клавиатуры.

Этот метод инициализирует экземпляры переменных из класса Activity и GUI-компоненты. Причем он упрощен до предела в целях ускорения загрузки приложений. В строке 37 выполняется необходимый вызов для метода onCreate суперкласса. Как и в случае с предыдущим приложением, в результате вызова метода setContentView (строка 38) передается константа R.layout.main для «раздувания» GUI на основе файла main.xml. Метод setContentView использует эту константу для загрузки соответствующего XML-документа, затем выполняет «раздувание» GUI.

Листинг 5.8. Переопределение метода onCreate из класса Activity

```
33 // вызывается при первом создании деятельности
34 @Override
35 public void onCreate(Bundle
savedInstanceState) 36 {
37     super.onCreate(savedInstanceState); // вызов версии
        суперкласса
38     setContentView(R.layout.main); // настройка
        макета 39
40     // объект SharedPreferences, который содержит сохраненные
        //поисковые запросы пользователя

41     savedSearches = getSharedPreferences("searches",
        MODE_PRIVATE);
```



```

42

43     // получение ссылки на queryTableLayout
44     queryTableLayout =
45         (TableLayout) findViewById(R.id.queryTableLayout);
46
47     // получение ссылок на два компонента EditTexts и кнопку
Save
48     queryEditText = (EditText)
        findViewById(R.id.queryEditText);
49     tagEditText = (EditText)
        findViewById(R.id.tagEditText); 50
51     // регистрация слушателей для кнопок Save и Clear Tags
52     Button saveButton = (Button)
        findViewById(R.id.saveButton);
53     saveButton.setOnClickListener(saveButtonListener);
54     Button clearTagsButton =
55         (Button) findViewById(R.id.clearTagsButton);
56     clearTagsButton.setOnClickListener(clearTagsButtonListen
        er);
57
58     refreshButtons(null); // добавление ранее сохраненных
        // поисковых запросов в GUI
59 } // конец метода onCreate 60

```

В строке 41 используется метод `getSharedPreferences` (косвенно унаследован из класса `Context`) для получения объекта `SharedPreferences`, который может считывать пары «тег/ запрос», сохраненные ранее в файле «поисков». Первый аргумент определяет имя файла, содержащего данные. Второй аргумент определяет доступность файла и может принимать одно из следующих значений:

- `MODE_PRIVATE`. Файл доступен только для данного приложения. В большинстве случаев эту константу можно использовать в качестве второго аргумента для `getSharedPreferences`.
- `MODE_WORLD_READABLE`. Любое приложение, выполняющееся на устройстве, может выполнять считывание из файла.
- `MODE_WORLD_WRITEABLE`. Любое приложение, выполняющееся на устройстве, может записывать в файл.

Эти константы могут комбинироваться вместе с поразрядным оператором `OR (|)`. Поскольку приложение не выполняет считывание больших объемов данных, метод `onCreate` выполняет достаточно быструю загрузку поисковых запросов. Никогда не реализуйте длинные запросы данных в потоке `UI thread`, если не хотите столкнуться с диалоговым окном `Application Not Responding (ANR)`, которое обычно появляется после пяти секунд «простоя» системы. Дополнительные сведения о диалоговых окнах `ANR` и создании «отзывчивых» приложений можно найти в документе developer.android.com/guide/practices/design/responsiveness.html.

В строках 44–49 находятся ссылки на компоненты `queryTableLayout`, `queryEditText` и `tagEditText` для инициализации соответствующих экземпляров переменных. В строках 52–56 находятся ссылки на компоненты `saveButton` и `clearTagsButton` и выполняется регистрация соответствующих слушателей. И наконец, в строке 58 вызывается объект `refreshButtons` (см. листинг 5.9), который используется для создания объектов `Button`, предназначенных для ранее сохраненных поисков, и кнопок `Edit`, с помощью которых можно изменять поиски.

Метод refreshButtons из класса FavoriteTwitterSearches. Метод refreshButtons из класса FavoriteTwitterSearches (см. листинг 5.9) создает и отображает новый тег запроса и кнопки редактирования либо для только что сохраненного поиска (если аргумент не равен null), либо для всех поисковых запросов (если аргумент равен null).

Объекты Button отображаются в алфавитном порядке, что облегчает поиск необходимых кнопок пользователем. В строках 66–67 получаем строковый массив, представляющий ключи объекта SharedPreferences. Метод getAll объекта SharedPreferences возвращает карту (Map), которая включает все пары «ключ/значение». Затем вызывается метод keySet для данного объекта, с помощью которого создается объект Set для ключей. И напоследок вызывается метод toArray (с пустым строковым массивом в качестве аргумента) объекта Set, с помощью которого выполняется преобразование объекта Set в строковый массив, который затем сортируется в строке 68. Метод Arrays.sort (статический метод класса Arrays из пакета java.util) сортирует массив по первому аргументу. Поскольку при вводе тегов пользователем могут использоваться символы верхнего и нижнего регистров, в данном случае реализована сортировка, независимая от регистра символов. Эта сортировка выполняется путем передачи методу Arrays.sort заранее определенного объекта Comparator<String> String.CASE_INSENSITIVE_ORDER в качестве второго аргумента.

Листинг 5.9. Метод refreshButtons из класса FavoriteTwitterSearches повторно создает и отображает новые теги поиска и кнопки редактирования для всех сохраненных поисковых запросов

```
61 // пересоздание поискового тега и кнопки Edit
    // для всех сохраненных поисковых запросов;

62 // передача null для создания всех тегов и кнопок Edit.
63 private void refreshButtons(String
newTag) 64 {
65     // store saved tags in the tags array
66     String[] tags =
67         savedSearches.getAll().keySet().toArray(new
String[0]);
68     Arrays.sort(tags, String.CASE_INSENSITIVE_ORDER); //
сортировка по тегу
69

70     // после добавления нового тега вставьте компонент GUI
    // в подходящее место

71     if (newTag != null)
72     {
73         makeTagGUI(newTag, Arrays.binarySearch(tags, newTag));
74     } // конец блока if
75     else // отображение GUI для всех тегов
76     {
77         // отображение всех сохраненных поисковых запросов
78         for (int index = 0; index < tags.length; ++index)
79             makeTagGUI(tags[index], index);
80     } // конец блока else
81 } // конец метода refreshButtons
82
```

В строках 71–80 определяется, будет ли метод вызываться для создания GUI, предназначенного для выполнения нового поиска либо для выполнения сохраненных поисков. В строке 73 вызывается метод makeTagGUI (см. листинг 5.11), который включает GUI для нового тега. Вызов Arrays.binarySearch во втором аргументе ищет точку вставки,

чтобы поддерживать отсортированные по алфавиту кнопки тегов. Если метод `refreshButtons` вызывается с аргументом `null`, в строках 78–79 вызывается метод `makeTagGUI` для каждого сохраненного поиска.

Метод `makeTag` из класса `FavoriteTwitterSearches`. Метод `makeTag` из класса `FavoriteTwitterSearches` (см. листинг 5.10) добавляет новый поиск в `savedSearches` либо изменяет существующий поиск. В строке 87 используется метод `getString` из класса `SharedPreferences` для поиска предыдущего значения (при его наличии), связанного с тегом. Если тег отсутствует в файле, возвращается второй аргумент (в данном случае `null`). В рассматриваемом случае метод также вызывает метод `refreshButtons` (строка 96), чтобы добавить GUI для нового поиска.

Листинг 5.10. Метод `makeTag` из класса `FavoriteTwitterSearches` добавляет новый поиск в файл сохранения, потом переустанавливает `Button`

```
83 // добавление нового поиска в файл, переустановка всех Button
84 private void makeTag(String query, String
tag) 85 {
86     // originalQuery может быть null при изменении существующего
запроса
87     String originalQuery =
savedSearches.getString(tag, null); 88
89     // получение SharedPreferences.Editor для хранения новой
// пары "тег/запрос"

90     SharedPreferences.Editor preferencesEditor =
savedSearches.edit();
91     preferencesEditor.putString(tag, query); //сохранение
текущего поиска
92     preferencesEditor.apply(); // сохранение обновленных
настроек
93

94     // Если новый запрос, добавляется соответствующий GUI
95     if (originalQuery == null)
96         refreshButtons(tag); // добавляется новая кнопка для
тега
97     } // конец метода makeTag 98
```

В строках 90–92 добавляется новый тег или изменяется соответствующее значение существующего тега. Чтобы изменить файл, связанный с объектом `SharedPreferences`, сначала нужно вызвать метод `edit`, с помощью которого получить доступ к объекту `SharedPreferences.Editor` (строка 90). Этот объект поддерживает методы, выполняющие добавление пар «ключ/значение», удаление пар «ключ/значение» и изменение значения, связанного с определенным ключом в файле `SharedPreferences`. В строке 91 вызывается метод `putString` этого объекта, с помощью которого сохраняется новый тег поиска (ключ) и запрашивается соответствующее значение. В строке 92 подтверждаются изменения в файле «поисков» путем вызова метода `apply` объекта `SharedPreferences.Editor`, который вносит изменения в файл.

Метод `makeTagGUI` из класса `FavoriteTwitterSearches`. Метод `makeTagGUI` из класса `FavoriteTwitterSearches` (см. листинг 5.11) добавляет в компонент `queryTableLayout` одну новую строку, содержащую тег, и кнопку `Edit`. Чтобы выполнить эту операцию, сначала «раздуйте» макет `new_tag_view.xml`, который был создан в разделе 5.4.5. Этот макет включает компонент `TableRow`, а также `newTagButton` и `newEditButton`.

В Android поддерживается служба, с помощью которой выполняется «раздувание» макета. Чтобы воспользоваться этой службой, следует получить ссылку на нее (строки 103–104) путем вызова метода `getSystemService`, унаследованного из класса `Activity`. При

вызове этого метода используется аргумент `Context.LAYOUT_INFLATER_SERVICE`. Метод `getSystemService` может возвращать ссылки на различные системные службы, но от вас потребуется привести результат к типу `LayoutInflater`. В строке 107 вызывается метод «раздувания» из класса `LayoutInflater` с константой `R.layout.new_tag_view`, которая представляет макет `new_tag_view.xml`. При этом возвращается ссылка представлению (`View`), которое фактически представляет собой компонент `TableRow`, включающий `Button`. В строках 110–113 происходит получение ссылки на элемент `newTagButton`, настраивается его текст в соответствии со значением тега и выполняется регистрация его `OnClickListener`. В строках 116–118 производится получение ссылки на `newEditButton` и выполняется регистрация соответствующих `OnClickListener`. В строке 121 добавляется `newTagView` в `queryTableLayout` в месте, на которое указывает индекс.

Листинг 5.11. Метод `makeTagGUI` из класса `FavoriteTwitterSearches` создает тег и кнопки `Edit Button` для одного поиска, а затем добавляет их в `queryTableLayout`, в позицию

```

99  // добавление в GUI кнопки нового тега и кнопок редактирования
100 private void makeTagGUI(String tag, int index)
101 {
102     // получение ссылки на службу LayoutInflater
103     LayoutInflater inflater = (LayoutInflater)
104         getSystemService(
105             Context.LAYOUT_INFLATER_SERVICE);
106
107     // «раздувание» new_tag_view.xml для создания кнопок
108     // нового тега и редактирования
109
110     View newTagView = inflater.inflate(R.layout.new_tag_view,
111         null);
112
113     // получение newTagButton, настройка текста и регистрация
114     // слушателя
115     Button newTagButton =
116         (Button) newTagView.findViewById(R.id.newTagButton);
117     newTagButton.setText(tag);
118     newTagButton.setOnClickListener(queryButtonListener);
119
120     // получение newEditButton и регистрация его слушателя
121     Button newEditButton =
122         (Button) newTagView.findViewById(R.id.newEditButton);
123     newEditButton.setOnClickListener(editButtonListener);
124
125     // получение кнопок нового тега и редактирования
126     // для queryTableLayout
127
128     queryTableLayout.addView(newTagView, index);
129 } // конец makeTagGUI

```

Метод `clearButtons` из класса `FavoriteTwitterSearches`. Метод `clearButtons` (см. листинг 5.12) удаляет все кнопки (`Button`) сохраненных поисковых запросов из приложения. В строке 128 вызывается метод `removeAllViews` класса `queryTableLayout`, который удаляет все вложенные компоненты `TableRow`, включающие объекты `Button`.

Листинг 5.12. Метод `clearButtons` из класса `FavoriteTwitterSearches` удаляет из приложения все объекты `Button`, представляющие сохраненные поисковые запросы

```

124 // удаляет кнопки всех сохраненных поисков из приложения
125 private void clearButtons()
126 {
127     // удаление всех кнопок сохраненных поисков
128     queryTableLayout.removeAllViews();
129 } // конец метода clearButtons
130

```

Анонимный внутренний класс, реализующий интерфейс `OnClickListener` в ответ на события `saveButton`. В строках 132–170 (см. листинг 5.13) создается анонимный внутренний класс объекта `saveButtonListener`, который реализует интерфейс `OnClickListener`. В строке 53 зарегистрирован слушатель `saveButtonListener` в качестве обработчика событий объекта `saveButtons`. В строках 134–169 реализован метод `OnClickListener`. Если пользователь введет запрос и тег (строки 138–139), метод вызовет метод `makeTag` (листинг 5.10), который сохраняет пару «тег/запрос» (строки 141–142), очищает оба компонента `EditText` (строки 143–144) и скрывает виртуальную клавиатуру (строки 147–149).

Если пользователь одновременно не ввел запрос и тег, метод отображает диалоговое окно `AlertDialog` (строки 151–168), в котором пользователю сообщается о том, что ему нужно ввести одновременно запрос и тег. Воспользуйтесь объектом `AlertDialog.Builder` (создан в строках 154–155) для конфигурирования и создания объекта `AlertDialog`. Аргумент конструктора — объект `Context`, в котором отображается диалоговое окно. В рассматриваемом случае это класс `FavoriteTwitterSearches Activity`, доступ к которому открывается по ссылке. Поскольку доступ открывается из анонимного внутреннего класса, следует полностью квалифицировать его, указав имя класса. В строке 157 определяется заголовок `AlertDialog` с помощью строкового ресурса `R.string.missingTitle`. Заголовок отображается в верхней части диалогового окна.

Листинг 5.13. Анонимный внутренний класс, реализующий интерфейс `OnClickListener` в ответ на события `saveButton`

```

131 // создание нового объекта Button и добавление в ScrollView
132 public OnClickListener saveButtonListener = new
    OnClickListener()
133 {
134     @Override
135     public void onClick(View v)
136     {
137         // создание тега, если queryEditText и tagEditText не
            пусты
138         if (queryEditText.getText().length() > 0 &&
139             tagEditText.getText().length() > 0)
140         {
141             makeTag(queryEditText.getText().toString(),
142                 tagEditText.getText().toString());
143             queryEditText.setText(""); // очистка queryEditText
144             tagEditText.setText(""); // очистка
tagEditText
145
146             // скрывание виртуальной клавиатуры
147             ((InputMethodManager) getSystemService(
148                 Context.INPUT_METHOD_SERVICE)).hideSoftInputFromWindow(
149                 tagEditText.getWindowToken(), 0);
150         } // конец блока if
151     else // отображение сообщения о необходимости ввода
            // пользователем запроса и тега

```

```

152 {
153     // создание нового AlertDialog Builder
154     AlertDialog.Builder builder =
155         new
156         AlertDialog.Builder(FavoriteTwitterSearches.this);

157     builder.setTitle(R.string.missingTitle); //
название строки

158

159     // кнопка ОК, скрывающая диалоговое окно
160     builder.setPositiveButton(R.string.OK, null);

161

162     // отображаемое сообщение
163     builder.setMessage(R.string.missingMessage);
164

165     // создание AlertDialog на основе
AlertDialog.Builder
166     AlertDialog errorDialog = builder.create();
167     errorDialog.show(); // display the Dialog
168 } // конец блока else
169 } // конец метода onClick
170 }; // конец анонимного внутреннего класса
OnClickListener 171

```

Диалоговые окна часто включают несколько кнопок. В рассматриваемом случае требуется лишь одна кнопка, с помощью которой пользователь может ознакомиться с сообщением. В данном случае идет речь о «позитивной» кнопке (строка 160). Метод `setPositiveButton` принимает название кнопки (указано в строковом ресурсе `R.string.OK`) и ссылку на обработчик событий кнопки. В этом диалоговом окне не нужно отвечать на событие, поэтому для обработчика событий указан `null`. Как только пользователь касается кнопки, диалоговое окно исчезает с экрана.

В строке 163 настраивается метод, который отображается в диалоговом окне (указан с помощью строкового ресурса `R.string.missingMessage`). В строке 166 создается диалоговое окно `AlertDialog` путем вызова метода `create` для `AlertDialog.Builder`. В строке 167 вызывается метод `show` объекта `AlertDialog`, отображающий модальное диалоговое окно.

Анонимный внутренний класс, реализующий интерфейс `OnClickListener` в ответ на события `clearTagsButton`. В строках 173–213 из листинга 5.14 создается объект анонимного внутреннего класса `clearTagsButtonListener`, реализующий интерфейс `OnClickListener`. В строке 56 этот объект регистрируется как обработчик событий `clearTagsButtons`. В строках 175–212 реализуется метод `onClick` интерфейса `OnClickListener`. Этот метод отображает диалоговое окно `AlertDialog`, в котором пользователю предлагается подтвердить необходимость удаления всех сохраненных поисков.

Листинг 5.14. Анонимный внутренний класс, реализующий интерфейс `OnClickListener` в ответ на события `clearTagsButton`

```

172 // очистка всех сохраненных поисков
173 public OnClickListener clearTagsButtonListener = new
OnClickListener()
174 {
175     @Override

```



```

176     public void onClick(View v)
177     {
178         // создание нового AlertDialog Builder
179         AlertDialog.Builder builder =
180             new
181                 AlertDialog.Builder(FavoriteTwitterSearches.this);

182         builder.setTitle(R.string.confirmTitle); //
строка заголовка 183

184         // кнопка OK, которая скрывает диалоговое окно
185         builder.setPositiveButton(R.string.erase,
186             new DialogInterface.OnClickListener()
187             {
188                 @Override
189                 public void onClick(DialogInterface dialog, int
button)
190                 {
191                     clearButtons(); //очистка сохраненных поисков из
карты 192
193                     // SharedPreferences.Editor для очистки поисков
194                     SharedPreferences.Editor preferencesEditor =
195                         savedSearches.edit();
196
197                     preferencesEditor.clear(); //удаление пар
тег/запрос
198                     preferencesEditor.apply(); // подтверждение
изменений
199                 } // конец метода onClick
200             } // конец анонимного внутреннего класса
201         ); // конец вызова метода
setPositiveButton 202
203         builder.setCancelable(true);
204         builder.setNegativeButton(R.string.cancel, null);
205
206         // настройка отображаемого сообщения
207         builder.setMessage(R.string.confirmMessage);
208
209         // создание диалогового окна AlertDialog из
AlertDialog.Builder
210         AlertDialog confirmDialog = builder.create();
211         confirmDialog.show(); // отображение диалогового окна
212     } // конец метода onClick
213 }; // конец анонимного внутреннего
класса OnClickListener
214

```

В строках 185–201 определяется «позитивная» кнопка AlertDialog и соответствующий этой кнопке обработчик событий. После щелчка на этой кнопке вызывается этот обработчик событий. В строке 191 вызывается объект clearButtons (см. листинг 5.12), который удаляет все объекты Button, представляющие сохраненные поисковые запросы. Затем вызывается объект SharedPreferences.Editor для savedSearches (строки 194–195), очищаются все пары «ключ/значение» путем вызова метода clear объекта SharedPreferences.Editor (строка 192). Затем подтверждаются изменения в файле (строка 198). Оператор в строке 203 «говорит» о том, что отображение диалогового окна может быть отменено пользователем. А именно, чтобы скрыть диалоговое окно, пользователю

достаточно нажать кнопку Back (Назад). В строке 204 настраивается «отрицательная» кнопка диалогового окна и обработчик событий. Подобно «положительной» кнопке из листинга 5.13, эта кнопка просто скрывает диалоговое окно. В строках 207–211 настраивается сообщение диалогового окна, создается диалоговое окно, в котором отображается сообщение.

Анонимный внутренний класс, реализующий интерфейс OnClickListener в ответ на события, генерируемые newTagButton. В строках 216–234 листинга 5.15 создается объект анонимного внутреннего класса queryButtonListener, который реализует интерфейс OnClickListener. В строке 113 этот объект регистрируется в качестве объекта обработки событий для каждого из созданных объектов newTagButton.

В строках 218–233 реализуется метод onClick интерфейса OnClickListener. В строке 222 производится получение текста кнопки Button, на которой был произведен щелчок мышью, а в строке 223 выбирается соответствующий поисковый запрос из savedSearches. В строке 226 вызывается унаследованный метод getString объекта Activity. В результате обеспечивается получение строкового ресурса searchURL, который включает URL-ссылку страницы поиска Твиттера. Потом запрос присоединяется в конец URL-ссылки.

Листинг 5.15. Анонимный внутренний класс, реализующий интерфейс OnClickListener в ответ на события queryButton

```
215 // загрузка выбранного поиска в окно веб-браузера
216 public OnClickListener queryButtonListener = new
    OnClickListener()
217 {
218     @Override
219     public void onClick(View v)
220     {
221         // получение запроса
222         String buttonText =
            ((Button)v).getText().toString();
223         String query =
            savedSearches.getString(buttonText, null); 224
225         // создание URL-ссылки, соответствующей запросу нажатой
            кнопки
226         String urlString = getString(R.string.searchURL) +
            query;
227
228         // создание интента для запуска веб-браузера
229         Intent getURL = new Intent(Intent.ACTION_VIEW,
230             Uri.parse(urlString));
231
232         startActivity(getURL); // вызов интента
233     } // конец метода onClick
234 }; // конец анонимного внутреннего класса
    OnClickListener
235
```

В строках 229–230 создается новый объект Intent (Интент), который будет использован для запуска веб-браузера устройства и отображения результатов поиска в Твиттере. Объект Intent представляет описание действия, которое будет выполняться по отношению к связанным данным. Первый аргумент, передаваемый конструктору Intent, представляет собой константу, описывающую выполняемое действие. В данный момент используется константа Intent.ACTION_VIEW, поскольку нужно отображать представление данных. Многие константы определены в классе Intent и описывают такие действия, как поиск, выбор, отсылка и воспроизведение. Второй аргумент (строка 230) — это Uri

(uniform resource identifier, унифицированный идентификатор ресурса). Этот аргумент указывает на данные, по отношению к которым выполняется действие. Метод `parse` из класса `Uri` преобразует строку, представляющую URL-ссылку (uniform resource locator, унифицированный локатор ресурса), в `Uri`-ссылку.

В строке 232 передается `Intent` методу `startActivity` (косвенно наследуется из класса `Context`), который запускает корректный класс `Activity`, реализующий выполнение определенного действия по отношению к текущим данным. В рассматриваемом случае указан просмотр `URI`-ссылки, поэтому `Intent` запускает веб-браузер устройства для отображения соответствующей веб-страницы. На этой странице отображаются результаты поиска, выполненного в Твиттере.

А теперь рассмотрим пример неявного объекта `Intent`. При определении подобного объекта не указывается явно компонент, отображающий веб-страницу, а просто разрешается системе запускать наиболее подходящую деятельность, основываясь на типе данных. Если для выполнения этой задачи могут использоваться несколько деятельностей, а данные передаются `startActivity`, система отображает диалоговое окно, в котором пользователь может выбрать используемую деятельность. Если система не может найти подходящую деятельность, метод `startActivity` генерирует исключение `ActivityNotFoundException`. Поэтому рекомендуется принять меры по обработке этого исключения. В данном случае этого не делается, поскольку на устройствах `Android`, на которых выполняется это приложение, установлен браузер, способный отображать веб-страницы.

При разработке приложений в будущем будут использованы явные `Intents`, в которых точно указывается класс `Activity`, выполняемый в том же приложении. Список приложений и поддерживаемых интенгов можно найти на веб-сайтах:

- openintents.org;
- developer.android.com/guide/appendix/g-app-intents.html

Анонимный внутренний класс, реализующий интерфейс `OnClickListener` в ответ на события `editButton`. В строках 237–253 листинга 5.16 создается объект анонимного внутреннего класса `editButtonListener`, который реализует интерфейс `OnClickListener`. В строке 118 этот объект регистрируется в качестве каждого из объектов обработки событий `newEditButton`. В строках 239–252 реализуется метод `onClick` интерфейса `OnClickListener`. Чтобы определить, каким образом будет редактироваться запрос кнопки поиска, сначала получим родительский макет `editButton` (строка 243). Именно здесь находится компонент `editButton`, который используется для получения объекта `Button` с ID, равным `R.id`. `newTagButton` в этом макете (строки 244–245). По сути это и есть соответствующая кнопка поиска. В строке 247 получаем текст `searchButton`, который используется в строке 250 для установки значения `tagEditText`. И наконец, в строке 251 получаем соответствующий запрос из объекта `savedSearches` и отображаем его в `queryEditText`.

Листинг 5.16. Анонимный внутренний класс, реализующий интерфейс `OnClickListener` в ответ на события `editButton`

```
236 // изменение выбранного поискового запроса
237 public OnClickListener editButtonListener = new
    OnClickListener()
238 {
239     @Override
240     public void onClick(View v)
241     {
242         // получение всех необходимых компонентов GUI
243         TableRow buttonTableRow = (TableRow) v.getParent();
244         Button searchButton =
245             (Button) buttonTableRow.findViewById(R.id.newTagButton);
```

246

```
247     String    tag    =
searchButton.getText().toString(); 248

249     // EditTexts должны соответствовать выбранному тегу и
запросу
250     tagEditText.setText(tag);
251     queryEditText.setText(savedSearches.getString(tag,
null));
252 } // конец метода onClick
253 }; // конец анонимного внутреннего класса OnClickListener
254 } // конец класса FavoriteTwitterSearches
```

5.7 Файл AndroidManifest.xml

В процесс создания любого проекта Android в среде Eclipse модуль ADT Plugin создает и конфигурирует файл AndroidManifest.xml (также известный под названием манифест приложения). В этом файле находится информация о приложении. В этом разделе будет рассмотрено содержимое этого файла (листинг 5.17), а также новое, добавленное в этот файл свойство. Другие свойства файла манифеста будут рассматриваться по мере возникновения необходимости в них. Дополнительные сведения о файле манифеста можно найти на следующем веб-сайте: developer.android.com/guide/topics/manifest/manifest-intro.html [1].

Листинг 5.17. Файл AndroidManifest.xml для приложения Favorite Twitter Searches

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest
  xmlns:android=http://schemas.android.com/apk/res/android
3   package="com.deitel.favoritetwittersearches"
4   android:versionCode="1"    android:versionName="1.0">
5   <application    android:icon="@drawable/icon"
6     android:label="@string/app_name">
7     <activity    android:name=".FavoriteTwitterSearches"
8       android:label="@string/app_name"
9       android:windowSoftInputMode="stateAlwaysHidden">
10      <intent-filter>
11        <action android:name="android.intent.action.MAIN" />
12        <category
13          android:name="android.intent.category.LAUNCHER" />
14      </intent-filter>
15    </activity>
16  </application>
17  <uses-sdk android:targetSdkVersion="10"
18    android:minSdkVersion="8"/>
19 </manifest>
```

Элемент manifest (строки 2–17) — это корневой элемент в AndroidManifest.xml. Атрибут элемента package (строка 3) определяет пакет, который используется для управления кодом. Атрибут элемента android:versionCode (строка 4) определяет внутренний целочисленный номер версии, который используется приложением для сравнения версий различных приложений. Атрибут элемента android:versionName (строка 4) определяет номер версии, который видят пользователи при управлении приложением, установленным.

Внутри элемента manifest находятся элементы nested application (строки 5–15) и uses-sdk (строка 16). Элемент application обязателен для указания. Атрибут элемента

android:icon определяет ресурс drawable, используемый в качестве пиктограммы приложения. Если пользователь не определит собственную пиктограмму, приложение будет использовать стандартную пиктограмму ADT Plugin при создании проекта приложения. Версии этой пиктограммы хранятся в папках приложения res/drawable. С помощью атрибута элемента android:label определяется название приложения. Элемент uses-sdk определяет библиотеку SDK, используемую приложением (10 соответствует Android SDK версии 2.3.3), и минимальную версию SDK (8 соответствует версии 2.2). Эти настройки обеспечивают выполнение приложения на устройствах Android 2.2 и выше.

Внутри элемента application находится элемент activity (строки 7–14), который содержит информацию о классе Activity приложения. Если приложение имеет более одного класса Activity, каждому из них будет соответствовать свой собственный элемент activity. Атрибут android:name (строка 7) определяет полностью квалифицированное имя класса Activity. Если имя класса предварить точкой (.), оно автоматически добавится к имени пакета, указанному с помощью элемента manifest. Атрибут android:label (строка 8) определяет строку, которая отображается вместе с Activity. По умолчанию манифест сконфигурирован таким образом, что в качестве значения этого атрибута используется имя приложения. В строку 9 мы добавили атрибут android:windowSoftInputMode. Значение stateAlwaysHidden определяет, будет ли отображаться виртуальная клавиатура после запуска Activity. Чтобы добавить этот атрибут, нужно либо изменить XML-код непосредственно, либо дважды щелкнуть мышью на файле AndroidManifest.xml проекта, чтобы открыть редактор манифеста. На рис. 5.8 показана вкладка Application (Приложение), выбранная в окне редактора манифеста. Названия вкладок отображаются в нижней части окна редактора. Чтобы установить атрибут android:windowSoftInputMode, выберите параметр .FavoriteTwitterSearches в разделе Application Nodes (Узлы приложения) окна (в нижней левой части). В правой нижней части окна редактора отобразятся атрибуты элементов activity. Выберите атрибут Window soft input mode (Режим программного ввода в окне) и щелкните на кнопке Select... (Выбрать...). На экране появится список доступных параметров. Выберите атрибут stateAlwaysHidden и щелкните на кнопке ОК.

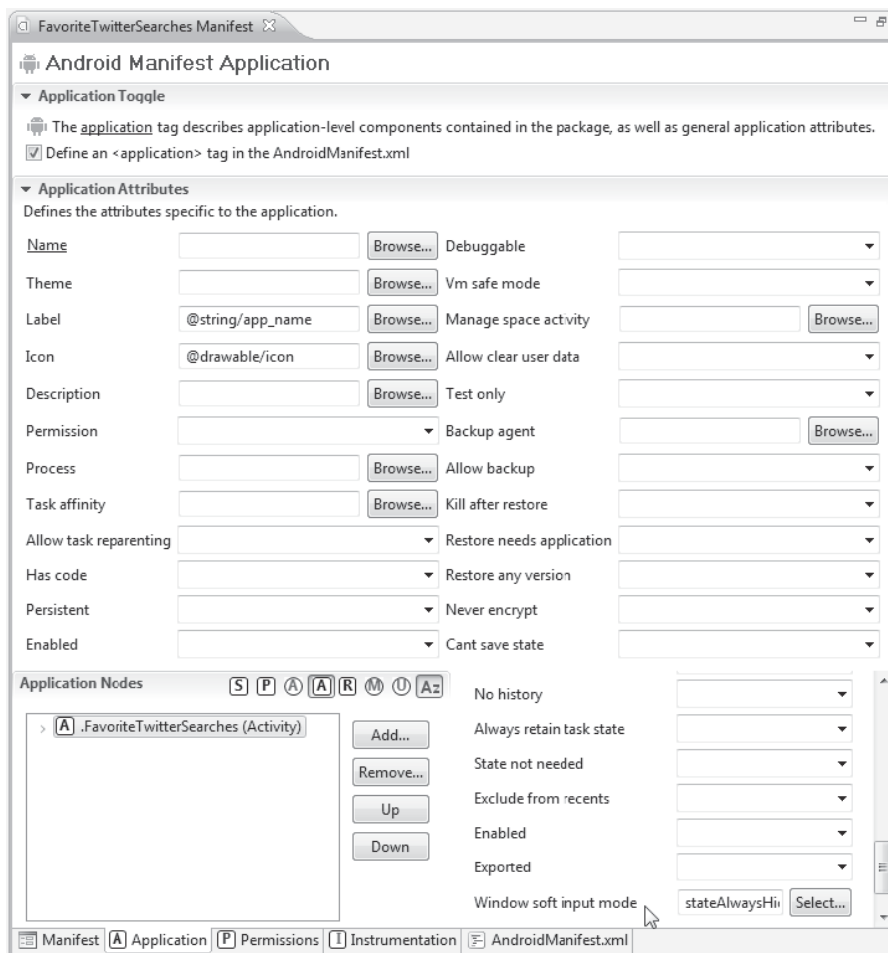


Рис. 5.8. Вкладка Application в окне редактора манифеста

Внутри элемента activity находится элемент intent-filter (строки 10–13), который определяет типы интенгов, на которые может реагировать класс Activity. Этот элемент должен включать один или большее число элементов action. Элемент в строке 11 — основная деятельность приложения, которая отображается после запуска приложения.

Элемент category (строка 12) определяет тип компонента Android, обрабатывающего событие. В данном случае значение "android.intent.category.LAUNCHER" свидетельствует о том, что эта деятельность должна быть внесена в список модулем запуска приложения вместе с другими приложениями устройства.

5.8 Резюме

В этой главе мы рассматривали приложение Favorite Twitter Searches. На первом этапе был создан графический интерфейс пользователя (GUI). Вы познакомились с компонентом ScrollView, который представляет собой группу ViewGroup, позволяющую выполнять про-крутку содержимого, не помещающегося на экране. Этот компонент также позволяет отображать сколь угодно длинные списки сохраненных поисковых запросов. Каждый запрос имеет связанную кнопку (Button), после касания которой выполняется поиск в окне веб-браузера устройства. Вы также узнали о том, каким образом можно создавать файлы ресурсов в диалоговом окне New Android XML File, а именно — файл colors.xml — хранилище ресурсов цвета, файл dimen.xml, предназначенный для хранения размеров, и файл второго макета, на основе которого методом «раздувания» был получен макет. Были рассмотрены методы ссылки на цвета и размеры в XML-разметке и порядок использования заранее определенных цветов из класса Android R.color.

Поисковые пары «тег/запрос» были сохранены в файле `SharedPreferences`, связанном с приложением. Мы рассмотрели скрывание виртуальной клавиатуры, выполненное программным путем. С помощью объекта `SharedPreferences.Editor` было выполнено сохранение значений, изменение значений, а также удаление значений из файла `SharedPreferences`. В ответ на касание пользователем кнопки поиска загружались `Uri`-ссылки в окно веб-браузера устройства путем создания нового интента с дальнейшей передачей метода `startActivity` из класса `Context`.

Объекты `AlertDialog.Builder` были применены для конфигурирования и создания окон `AlertDialogs`, в которых отображаются сообщения для пользователя. Было продемонстрировано программное создание компонентов GUI путем ручного «раздувания» файла XML-разметки. В результате приложение получило возможность динамического изменения GUI на основе взаимодействия с пользователем. Эта методика была использована для создания компонента `TableRow`, включающего два новых объекта `Button` (кнопки), один из которых выполняет поиск, а второй позволяет изменить поиск. Эти компоненты `TableRows` были добавлены в `TableLayout` в `ScrollView`, в результате чего все тегированные поиски могут отображаться в области прокрутки экрана.

И наконец, был рассмотрен файл `AndroidManifest.xml` и показано, как сконфигурировать приложение, чтобы после запуска приложения не отображалась виртуальная клавиатура.

В главе 6 будет создано приложение `Flag Quiz Game`, на экране которого отображается флаг определенной страны, название которой предстоит выбрать пользователю среди доступных 3, 6 или 9 вариантов. С помощью меню и флажков вы сможете настроить приложение, ограничив количество отображаемых флагов и стран с определенными регионами мира.

6 Управление ресурсами приложения

6.1 Ресурсы приложения и системные ресурсы

При генерировании пользовательского интерфейса приложениями Android используются строковые, графические и другие типы ресурсов. Ресурсы могут быть включены в проект Android согласно четко определенной иерархии. В этом главе мы рассмотрим наиболее распространенные типы ресурсов, используемые в приложениях Android, вы узнаете как они хранятся и как к ним получить доступ в программном коде [2].

Ресурсы приложения и системные ресурсы. Ресурсы бывают двух типов: ресурсы приложения и системные ресурсы. Ресурсы приложения определяются разработчиками в файлах проектов Android отдельно для каждого приложения. Системные ресурсы — это общие ресурсы, определенные платформой Android, они доступны для всех приложений через Android SDK.

При выполнении приложения вы можете получать доступ к обоим типам ресурсов. Вы можете также получать доступ к ресурсам из других скомпилированных ресурсов, таких как XML-файлы макетов для определения атрибутов элементов пользовательского интерфейса.

Ресурсы приложения. Ресурсы приложения создаются и хранятся в файлах проектов Android, в папке /res. Согласно строго определенной, но гибкой структуре каталогов ресурсы сортируются, определяются и компилируются с пакетом приложения. В большинстве случаев ресурсы приложения не используются системой Android.

Хранение ресурсов приложения. В процессе программирования данные, используемые приложением, обычно представлены в качестве ресурсов. Объединение ресурсом и компиляция в единый пакет приложения имеет ряд преимуществ:

- Код становится более понятным и легким для чтения, что уменьшает количество
- возможных ошибок.
- Ресурсы сортируются по типу и поэтому не повторяются.
- Ресурсы расположены удобно для обеспечения гибкости настройки телефона.
- Процесс локализации и интернационализации приложения очень прост.

Платформа Android поддерживает множество типов ресурсов, с помощью которых могут быть созданы самые разные приложения. В приложении Android может использоваться множество различных видов ресурсов.

Ниже приведен список некоторых из наиболее распространенных типов:

- строки, цвета и размерности;
- графические файлы рисунков;
- файлы макета;
- подключаемые файлы всех типов.

Все ресурсы распределяются по специальным папкам проекта, а тип ресурса определяется XML-тегами. Некоторые папки, такие как /res/drawable, /res/layout и /res/values, создаются автоматически для каждого нового проекта Android, кроме них разработчик может также добавить свои папки.

К файлам ресурсов, хранящимся в папке /res, применяются следующие требования:

- Символы имени файла ресурса должны быть в нижнем регистре.
- Имя файла ресурса может содержать только латинские буквы, числа, символ подчеркивания и точки.
- Имена файлов ресурса (и XML-атрибуты name) должны быть уникальными.

При компилировании ресурсов, имена соответствующих им переменных берутся из имени файла. Например, графическому файлу mypic.jpg, сохраненному в папке /drawable, соответствует ресурс @drawable/mypic. Поэтому будьте внимательны во время присвоения имен файлам ресурсов.

Более подробную информацию об именовании папок и файлов проекта вы можете найти в документации по Android.

Обращение к ресурсам приложения. Все ресурсы приложения сохраняются в папке /res проекта и компилируются в проект во время сборки. Ресурсы приложения могут использоваться программно, и к ним может происходить обращение из других ресурсов приложения.

Каждый раз, когда вы сохраняете файл ресурса (т.е. копируете файл ресурса, например, мафический файл, в соответствующую папку) в Eclipse, файл класса R.java перекомпилируется для учета изменений. Если имя какого-либо файла не соответствует принятым соглашениям, компилятор сообщит об ошибке на панели Problems (Проблемы) в Eclipse.

К ресурсам приложения можно получить доступ программно через сгенерированный класс R.java. Чтобы обратиться к ресурсу, вы должны получить доступ к объекту приложения Resources с помощью метода getResources () и затем сделать вызов этого метода, основанный на типе ресурса, который вы хотите получить.

Например, чтобы получить строковый ресурс hello, определенный в файле strings.xml, используется следующий вызов метода:

```
String greeting=getResources ().getString (R. string.hello) ;
```

Более подробно о получении доступа к различным типам ресурсов мы поговорим чуть позже.

Для обращения к ресурсу приложения из другого скомпилированного ресурса, такого как файл макета, используется следующий синтаксис:

```
@тип_ресурса/имя_ресурса.
```

Например, к строке, представленной в предыдущем примере, вы можете обратиться следующим образом:

```
@string/hello
```

Более подробно об обращении к ресурсам мы поговорим далее, когда речь пойдет о файлах макета.

Работа с системными ресурсами. Кроме ресурсов приложения, разработчики могут получать доступ к системным ресурсам Android. У вас есть доступ к большинству системных ресурсов, распределяемых между несколькими приложениями, что поможет создать общий стиль и восприятие. Например, класс системных строковых ресурсов Android (android.R.string) содержит стандартные слова, такие как OK, Cancel (Отмена), Yes (Да), No (Нет). Cut (Вырезать), Copy (Копировать), Paste (Вставить).

Для поддержания малого объема вашего приложения и эффективного использования ресурсов перед добавлением новых ресурсов в проект проверяйте системные ресурсы — возможно, нужный вам ресурс, уже существует.

Все системные ресурсы находятся в пакете android. В нем содержатся классы для каждого из основных типов ресурсов.

Например, класс android.R.string содержит строковые системные ресурсы. Чтобы получить системный строковый ресурс со значением ok, например, сначала вы должны воспользоваться статичным методом: getSystemService() класса Resources, обеспечивающим доступ к глобальному системному объекту Resource, а затем методом getString(), указав полное имя строкового ресурса, например:

```
String confirm = Resources.getSystemService().  
getString(android.R.string.ok) ;
```

Для обращения к системному ресурсу из скомпилированного ресурса, такого как файл макета, используется следующий синтаксис:

```
@android:[тип_ресурса]/[имя_ресурса]
```

Например, вы можете обратиться к системному строковому ресурсу ok, установив соответствующий атрибут следующим образом:

```
@android:string/ok
```

Работа с простыми ресурсами. Простые ресурсы, такие как строка, цвет и размерность, должны быть определены в XML-файлах папки /res/values проекта. В этих файлах используются специальные XML-теги, представляющие собой пары имя/значение. Такие ресурсы компилируются в пакет приложения во время сборки.

Управление ресурсами строк, цвета и размерностей производится в редакторе ресурсов Eclipse, но вы можете отредактировать XML-файл в любом текстовом редакторе.

Работа со строками. Строковые ресурсы обычно используются тогда, когда в вашем приложении требуется произвести вывод какого-либо текста на экран. Строковые ресурсы обозначаются тегом <string> и хранятся в файле ресурсов /res/values/ strings.xml.

Ниже приведен пример файла строкового ресурса:

```
<?xml
version="1.0" encoding="utf-
8"?>
<resources>
<string
name="app_
name">Name this App</string>
<string
name="
hello">Hello</string>
</resources>
```

К строковым ресурсам можно применять форматирование. Строковые ресурсы, содержащие апострофы (одинарные прямые кавычки), должны быть заключены в двойные прямые кавычки.

Существует несколько способов доступа к строковому ресурсу на программном уровне. Самый простой заключается в использовании метода getString();

```
String greeting = getResources().getString(R.string.hello);
```

Работа с цветами. Вы можете применять цветовые ресурсы к различным элементам пользовательского интерфейса. Цветовые ресурсы обозначаются тегом <color> в файле /res/values/ colors.xml. Этот XML-файл ресурса не создается автоматически, поэтому при необходимости должен быть создан вручную.

Вы можете добавить новый XML-файл, такой как этот, в специальном диалоговом окне, которое открывается командой File => New => Android XML File (Файл => Новый => XML-файл Android). Таким образом, предполагаемая папка и тип файла ресурса для проекта Android, будут заданы автоматически.

Ниже приведен пример цветового файла ресурса:

```
<?xml
version="1.0" encoding="utf-
8"?>
<resources>
<color
name="background_
color">#006400</color>
<color
name="app_text_
color">#FF00FF</color>
</resources>
```

Система Android поддерживает 12-разрядные и 24-разрядные цвета в цветовой модели RGB. В табл. 6.1 представлен список форматов, поддерживаемых платформой Android.

Таблица 6.1 Цветовые форматы, поддерживаемые платформой Android

Формат	Описание	Пример
#RGB	12-разрядный цвет	#00F (синий)
#ARGB	12-разрядный цвет с поддержкой	#800F (синий, прозрачность 50%)
#RRGGBB	24-разрядный цвет	#FF00FF (пурпурный)

#AARRGGBB	24-разрядный цвет с поддержкой прозрачности	#80FF00FF (пурпурный, прозрачность 50%)

В следующем фрагменте кода происходит получение цветового ресурса `app_text_color` методом `getColor()`:

```
int textColor = getResources().getColor(R.color.
app_text_
color);
```

Существует множество способов подбора цвета. Например, на сайте html-color-codes.info вы найдете простую таблицу цветов, щелкая по ячейкам которой вы можете выбрать подходящий цвет.

Работа с размерностями. Размер элементов пользовательского интерфейса, таких как кнопки или элемент `Textview`, определяется различными видами размерностей. Ресурсы размерности обозначаются тегом `<dimen>` в файле ресурса `/res/values/ dimens.xml`. Этот XML-файл не создается автоматически и должен быть создан вручную.

Пример файла ресурса размерности:

```
<?xml
version="1.0" encoding="utf-
8"?>
<resources>
<dimen
name="
thumbDim">100px</dimen>
</resources>
```

Для каждого значения ресурса размерности должна быть указана единица измерения. В табл. 6.2 перечислены единицы измерения, поддерживаемые платформой Android.

Таблица 6.2 Единицы измерения размерностей, поддерживаемые Android

Единица измерения	Описание	Условное обозначение
Пиксели	Экранные пиксели	px
Дюймы	Физическая величина	in
Миллиметры	Физическая величина	mm
Точки	Применяется в основном для измерения шрифтов	pt
Пиксели (независимо от плотности)		160 dpi
Пиксели относительно плотности		dp
Пиксели (независимо от масштаба)	Применяются для масштабирования шрифтов	sp

В следующем фрагменте кода происходит получение ресурса размерности методом `getDimension()`:

```
float thumbnailDim = getResources().getDimension(R.dimen.thumbnailDim);
```

Работа с ресурсами объектов. Ресурсы рисунков, такие как файлы изображений, хранятся в папке `/res/ drawable` проекта. Во время сборки эти ресурсы будут скомпилированы в пакет приложения, чтобы быть доступными приложению.

Вы можете перетаскивать файлы изображений непосредственно в папку `/res/ drawable` через панель `Project Explorer` (Проводник проектов) в `Eclipse`. Не забывайте, что имена файлов должны быть в нижнем регистре и должны содержать только латинские буквы, числа и символы подчеркивания.

Работа с изображениями. Наиболее распространенные графические ресурсы, используемые в приложениях, это растровые изображения, такие как PNG и JPG-файлы. Эти файлы часто используются в качестве значков приложений и кнопок, а также элементов пользовательского интерфейса.

Как вы можете видеть в табл. 6.3, Android поддерживает несколько распространенных форматов изображения.

Таблица 6.3 Графические форматы изображения, поддерживаемые платформой Android

Формат изображения	Описание	Расширение файла
PNG	Рекомендуемый формат (без потерь)	.png
PNG с поддержкой растягивания NinePatch	Рекомендуемый формат (без потерь)	.9.png
JPEG	Допустимый формат (с потерями)	.jpg
GIF	Не рекомендуется к использованию	.gif

Использование ресурсов изображения в программном коде. Ресурсы изображений содержатся в классе `BitmapDrawable`. Чтобы получить доступ к графическому файлу ресурса, например `/res/drawable/logo.png`, используйте метод `getDrawable()`:

```
BitmapDrawable logoBitmap =
(BitmapDrawable) getResources().getDrawable(R.drawable.logo);
```

Однако в большинстве случаев нет необходимости загружать графику напрямую. Вместо этого вы можете использовать идентификатор ресурса в качестве атрибута элемента пользовательского интерфейса, например `ImageView`. В следующем фрагменте кода файл `logo.png` устанавливается и загружается в элемент интерфейса `ImageView` с именем `LogoImageView`, который должен быть определен в макете:

```
ImageView logoView = (ImageView) findViewById(R.id.LogoImageView);
logoView.setImageResource(R.drawable.logo);
```

Работа с другими типами рисунков. В дополнение к графическим файлам вы можете также создать специальные XML-файлы для описания других подклассов `Drawable`, таких как `ShapeDrawable`. Вы можете использовать класс `ShapeDrawable` для описания различных геометрических фигур, таких как прямоугольники и овалы. Более подробную информацию вы можете найти в документации Android, пакет `android.graphics.drawable`.

Работа с макетами. Большинство элементов пользовательского интерфейса приложений Android описывается специальными XML-файлами, называемыми файлами макета. Файлы ресурсов макетов находятся в папке `/res/layout`. Также, как и другие ресурсы, файлы макета компилируются вместе с приложением.

Очень часто файлы макета описывают весь экран и ассоциируются с определенной деятельностью, но это не единственный верный подход. Ресурсы макетов могут также описывать часть экрана и быть составной частью другого макета.

Далее приведен пример файла ресурса макета:

```
<?xml
version="1.0" encoding="utf-
8"?>
<LinearLayout
xmlns:
android="http://schemas.android.com/apk/res/android"
android:
orientation="vertical"
android:layout_
width="fill_parent"
android:layout_
height="fill_
parent">
<TextView
android:layout_
```

```
width="fill_parent"  
android:layout_  
height="wrap_content"  
android:  
text="@string/hello"  
</LinearLayout>
```

Возможно, вы уже узнали код этого файла — это файл макета `main.xml` по умолчанию, который автоматически создается для каждого нового приложения Android. Этот файл макета описывает пользовательский интерфейс деятельности в пределах приложения. Он содержит элемент `LinearLayout`, используемый в качестве контейнера для всех элементов пользовательского интерфейса — в нашем случае элемента `TextView`.

В этом примере файл макета `main.xml` содержит ссылку на другой ресурс — строковый — `@string/hello`, который определен в файле ресурсов `strings.xml`.

Макеты могут также создаваться, редактироваться и использоваться во время работы приложения. Однако в большинстве случаев XML-файлы макетов значительно упрощают код и его использование.

Существует два способа форматирования ресурсов макета. Самый простой способ заключается в использовании редактора ресурсов Eclipse для разработки и предварительного просмотра файлов макета. Вы также можете редактировать XML-файлы макета в любом текстовом редакторе.

Проектирование макетов в редакторе ресурсов Вы можете проектировать и просматривать макет в Eclipse с помощью редактора ресурсов макета. Если выбрать файл проекта `/res/layout/main.xml` в Eclipse, вы увидите панель `Layout (Макет)`, на которой в режиме предварительного просмотра отображается интерфейс, сформированный этим макетом. Добавлять и удалять элементы макета вы можете на панели `Outline (Структура)`. Свойства и атрибуты для каждого элемента задаются на панели `Properties (Свойства)`.

Как большинство других инструментов проектировки интерфейса, редактор ресурсов макета подходит для редактирования простого макета. Однако редактор ресурсов макета поддерживает элементы пользовательского интерфейса не полностью. Для создания более сложных элементов придется редактировать XML-файл вручную. После добавления новых элементов интерфейса в макет вы можете потерять возможность предварительного просмотра. Однако вы по-прежнему можете видеть макет интерфейса в эмуляторе или на мобильном телефоне.

Помните, что одна из главных задач разработчика — корректное отображение приложения на мобильном телефоне, а не в редакторе макета Eclipse.

Проектировка макетов в коде XML. Вы можете редактировать код необработанного XML-файла макета вручную. Если вы выберете файл `/res/layout/main.xml` и перейдете на вкладку `main.xml`, то увидите XML-код файла макета.

По мере приобретения опыта в разработке макетов, вы должны лучше ознакомиться с принципами составления XML-файлов. Время от времени переходите на вкладку редактирования кода XML и обращайте внимание на код, сгенерированный для каждого типа элемента пользовательского интерфейса. Не полагайтесь на один только редактор ресурсов макета Eclipse.

Ознакомьтесь с редактором ресурсов макета Eclipse:

1. Откройте проект, созданный вами ранее.
2. Найдите файл макета `/res/layout/main.xml` и откройте его в редакторе ресурсов макета Eclipse двойным нажатием.
3. Переключитесь на вкладку `Layout (Макет)`, на ней вы увидите макет главного экрана в режиме предварительного просмотра.
4. Перейдите на панель `Outline (Структура)`. Здесь представлена иерархия XML-файла макета. Сейчас вы можете видеть элемент интерфейса `LinearLayout`, который содержит в себе элемент интерфейса `TextView`.

5. Выберите элемент интерфейса `TextView` на панели `Outline` (Структура). Теперь этот элемент интерфейса в предварительном просмотре макета выделен цветом.
6. Перейдите на вкладку `Properties` (Свойства). Здесь приведены все свойства и атрибуты элемента пользовательского интерфейса `TextView`, которые вы можете конфигурировать. Найдите в этом списке свойство `Text` и обратите внимание, что его значением является строковый ресурс `@string/hello`.
7. Щелкните по значению `@string/hello` этого свойства на панели `Properties` (Свойства), чтобы изменить его. Каждый раз, когда вы меняете значение, обращайтесь также внимание на изменения в предварительном просмотре макета. Вы можете ввести строку непосредственно, указать другой строковый ресурс (например, `@string/app_name`) или, щелкнув по кнопке с многоточием, выбрать подходящий ресурс из списка доступных для вашего приложения.
8. Перейдите на вкладку `main.xml` и обратите внимание на XML-код. Если вы сохраните и запустите свой проект в эмуляторе, то все изменения, которые были видны в предварительном просмотре, будут применены в эмуляторе.

Не бойтесь экспериментировать с `Eclipse`. Вы можете добавить несколько новых элементов интерфейса, таких как `ImageView` или еще один элемент `TextView` в ваш макет.

Наиболее важные панели в редакторе ресурсов макета — `Outline` (Структура) и `Properties` (Свойства). Если разместить панель `Properties` (Свойства) в верхнем правом углу перспективы `Java`, то вы можете одновременно видеть основной редактор ресурсов макета в середине экрана, панель `Project Explorer` (Проводник проекта) слева, панель `Properties` (Свойства) справа (чтобы полностью использовать вертикальное пространство), а панель `Outline` (Структура) — внизу (на ней обычно меньше информации, чем на панели `Properties` (Свойства)).

Использование ресурсов макета в программном коде. Для доступа к макету программно вы можете воспользоваться классом `LayoutInflater`, чтобы расширить файл макета к объекту `View`, как в следующем примере:

```
LayoutInflater inflater = getLayoutInflater();  
View layout = inflater.inflate(R.layout.main, null);
```

В большинстве случаев нет необходимости загружать весь макет, так как нам нужны лишь отдельные элементы, такие как `TextView` в файле макета `main.xml`. Содержание макета, независимо от того, какой это элемент интерфейса (`Button`, `ImageView` или `TextView`), происходит из класса `View`.

Файл макета по умолчанию, созданный для проекта `Droid1`, содержит элемент интерфейса `TextView`. Однако элемент интерфейса `TextView` по умолчанию не имеет атрибута `name`. Самый простой способ получить доступ к необходимому элементу интерфейса `View` — по его имени, поэтому вам необходимо задать атрибут `id` элементу интерфейса `TextView` в редакторе ресурсов макета. Назовите его `@+id/TextView01`.

Ниже приведен пример доступа к объекту `TextView` с именем `TextView01`, который был определен в файле ресурса макета:

```
TextView txt = (TextView)findViewById(R.id.TextView01);
```

Работа с файлами. Кроме ресурсов строк, графических ресурсов и ресурсов макетов, в проектах `Android` могут содержаться файловые ресурсы. Такие файлы могут быть в любом формате. Тем не менее некоторые форматы более удобны и используются чаще.

Работа с XML-файлами. Формат `XML` полностью поддерживается платформой `Android`. Любые `XML`-файлы могут быть включены в качестве ресурсов, именно их рекомендуется использовать для хранения упорядоченных данных.

Структура `XML`-файла может быть любой — на ваше усмотрение. Множество утилит `XML` (табл. 6.4) входят в состав платформы `Android`.

Таблица 6.4 Пакеты утилит XML

Пакет	Описание
android.sax.*	Средство для написания стандартных обработчиков SAX
android.util.Xml.*	Утилиты XML, включая XMLPullParser
org.xml.sax.*	Основные возможности SAX (см. www.saxproject.org)
javax.xml.*	SAX и ограничения DOM с поддержкой ядра 2-го уровня
org.w3c.dom	Интерфейсы для DOM, ядро 2-го уровня
org.xmlpull.*	Интерфейсы XmlPullParser и XMLSerializer (см. www.xmlpull.org)

Для доступа к XML-файлу `/res/xml/default_values.xml` программно вы можете воспользоваться методом `getXml()`:

```
XmlResourceParser defaultDataConfig = getResources().getXml(R.xml.
default_values);
```

Работа с подключаемыми файлами. В приложениях Android в качестве ресурсов могут использоваться подключаемые необработанные файлы (raw files). Подключаемыми могут быть аудио- и видеофайлы, а также множество других форматов. Все подключаемые файлы ресурсов должны содержаться в папке `/res/raw`.

Не существует никаких правил или ограничений на содержание подключаемых файлов (кроме имен файлов ресурсов, о которых говорилось ранее). Но перед интегрированием каких-либо медиафайлов в проект обратитесь к документации по платформе Android, чтобы узнать какие именно форматы и кодировки поддерживаются на тех мобильных телефонах, где вы планируете распространять ваше приложение. То же самое касается любого другого формата файла, который вы будете использовать в качестве ресурса приложения. Если формат файла, который вы планируете использовать, не поддерживается системой Android, то ваше приложение должно производить всю работу по обработке файлов собственными силами.

Для доступа к подключаемым файлам ресурсов программно можете воспользоваться методом `openRawResource()`. Например, в следующем фрагменте кода создается объект `InputStream` для доступа к файлу ресурса `/res/raw/file1`.

```
txt:InputStream iFile = getResources().openRawResource(R.raw.file1);
```

Каждому подключаемому файлу ресурса должно быть присвоено уникальное имя, исключая суффикс имени файла, например одновременное использование имен файлов `file1.txt` и `file1.dat` недопустимо.

Бывают случаи, когда вам может быть необходимо использовать файлы в приложении, но при этом не компилировать их вместе с остальными ресурсами. Для этой цели в Android предусмотрена специальная папка проекта — `/assets`. Эта папка находится в той же самой папке, что и `/res`. Все файлы, хранящиеся в этой папке, представлены в бинарном виде, с пакетом установки приложения, и они не компилируются.

Некомпилированные файлы, называемые активами приложения, не доступны методом `getResources()`. В этом случае для доступа к файлам а папке `/assets` используется `AssetManager`.

Работа с другими типами ресурсов. Мы рассмотрели все наиболее часто используемые типы ресурсов, которые нам могут понадобиться в приложении. Но, кроме рассмотренных, существует еще несколько типов ресурсов. Эти типы используются редко, и их применение на практике может вызвать затруднения. Они рассчитаны на очень мощные приложения. Ниже перечислены некоторые из таких типов ресурсов:

- строковые массивы;
- меню;
- анимации;
- геометрические фигуры;

- стили и темы;
- собственные элементы пользовательского интерфейса.

Если вы все же планируете использовать эти типы ресурсов, ознакомьтесь с информацией по данной теме в документации Android.

Таким образом, в приложениях Android может использоваться множество различных типов ресурсов, включая узкоспециальные типы ресурсов и системные ресурсы. Редактор ресурсов Eclipse упрощает управление ресурсами, но код XML- файлов ресурсов должен редактироваться вручную. После определения к ресурсам можно получить доступ непосредственно или по имени из других ресурсов. Строковые ресурсы, ресурсы цвета и размерности хранятся в специальных XML-файлах, а графические изображения хранятся в виде отдельных файлов. Пользовательские интерфейсы определяются XML- файлами макетов.

Подключаемые файлы, содержащие данные в различных форматах, также могут использоваться в качестве ресурсов. Кроме этого, приложения могут включать некомпилированные бинарные файлы, называемые активами приложения.

6.2 Конфигурирование файла манифеста ANDROID

Каждый проект Android должен содержать специальный файл, называемый файлом манифеста. Из этого файла система получает настройки конфигурации приложения, включая идентификационную информацию, а также права, требуемые приложению для нормальной работы. В данном разделе мы рассмотрим файл манифеста Android более подробно, и вы узнаете, как он используется в различных приложениях [2].

Файл манифеста Android — AndroidManifest.xml — один из важнейших файлов любого проекта Android. В системе Android этот файл служит для нескольких целей:

- установки и обновления пакетов приложений;
- предоставления информации о приложении пользователю;
- запуска деятельности приложения;
- управления правами приложения;
- обработки множества других дополнительных конфигураций приложения, включая работу в качестве провайдера услуг или контент-провайдера.

Если вы пользуетесь Eclipse с плагином ADT, то мастер проектов Android автоматически создаёт готовый файл AndroidManifest.xml со значениями по умолчанию для наиболее важных настроек конфигурации.

Вы можете редактировать файл манифеста Android и редакторе ресурсов Eclipse или в любом другом текстовом редакторе. Редактор ресурсов Eclipse распределяет данные файла манифеста по нескольким категориям, представленным на пяти вкладках:

- Manifest (Манифест);
- Application (Приложение);
- Permissions (Права);
- Instrumentation (Инструментарий);
- AndroidManifest.xml.

Вкладка Manifest (Манифест). На вкладке Manifest (Манифест) содержатся настройки всего пакета, включая имя пакета, информацию о версии приложения и минимально допустимой версии Android SDK. Здесь вы можете также задать аппаратные требования.

Вкладка Application (Приложение). На вкладках Application (Приложение) содержатся настройки всего приложения, включая метку приложения и значок, а также информацию о компонентах приложения, таких как деятельности, интент-фильтры, и другой функциональности приложения, включая конфигурацию провайдера услуг и контент-провайдера.

Вкладка Permissions (Права). На вкладке Permissions (Права) описаны права, предоставляемые приложению. Здесь могут быть также указаны специальные права, которые требуются приложению для работы.

Не путайте раскрывающийся список Permission (Права) вкладки Application (Приложение) с возможностями вкладки Permissions (Права). На вкладке Permissions (Права) указываются права, требуемые приложению для доступа к необходимым ресурсам или API. В раскрывающемся списке Permission (Права) определяются права, требуемые другими приложениями для доступа к открытым ресурсам и API вашего приложения.

Вкладка Instrumentation (Инструментарий). На вкладке Instrumentation (Инструментарий) указываются классы инструментария для мониторинга за приложением. В поле Name (Имя) вводится полное имя класса подкласса instrumentation вашего приложения, а в поле Target Package (Целевой пакет) вы указываете имя пакета, файл которого содержит <application> для контролируемого приложения. Более подробно об инструментарии и тестировании мы поговорим далее.

Вкладка AndroidManifest.xml. Файл манифеста Android представляет собой XML-файл. Вы можете редактировать код XML вручную на вкладке AndroidManifest.xml редактора ресурсов.

Конфигурирование основных настроек приложения. Большинство самых важных настроек приложения задаются атрибутами и дочерними тегами блоков <manifest> и <application>.

Пакет определяется в теге <manifest> файла манифеста Android с помощью атрибута package, например:

```
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
package="com.androidbook.droid1"
android:
versionCode="1"
android:
versionName="
1.0">
```

Если для создания проектов вы пользуетесь мастером проектов Android, то вы должны указывать имя пакета. Это имя будет использоваться в файле манифеста Android по умолчанию.

Подключение вспомогательных библиотек. С помощью тега <uses-library> приложения могут взаимодействовать с другими библиотеками в дополнение к стандартным пакетам Android. Эта возможность доступна на вкладке Application (Приложение) редактора ресурсов. Например:

```
<uses-library
android:
name="com.company.sharedutilities"
/>
```

Управление версиями приложений. Информация о версии манифеста используется в двух случаях:

- для организации и отслеживания возможностей приложения;
- управления обновлениями приложения.

По этой причине у тега <manifest> есть два отдельных атрибута версии: имя версии и код версии.

Имя версии. Имя версии представляет собой информацию о версии, используемой для отслеживания сборок приложения. Продуманное управление версиями очень важно для публикации приложения и последующей поддержки. Атрибут android:versionName тега <manifest> хранит строковое значение для отслеживания номера сборки приложения.

Формат имени версии определяется разработчиком. Учтите, что это поле могут видеть пользователи.

Установка кода версии. Код версии позволяет платформе Android программно обновлять приложения. Значение атрибута `android:versionCode` тега `<manifest>` может быть целое число, используемое платформой Android и рынками распространения для управления обновлением приложения. Значение атрибута `android:versionCode` обычно начинается с 1. Это значение должно постепенно увеличиваться с каждой новой версией приложения, готовой для опубликования. Поле кода версии невидимо для пользователей и не должно синхронизироваться с именем версии.

Код версии должен увеличиваться только для опубликованных приложений или в целях тестирования. Не изменяйте его каждый раз, когда вы развертываете приложение на устройстве для отладки.

Установка минимальной версии Android SDK. Приложения Android могут быть скомпилированы для совместимости с несколькими различными версиями SDK. В теге `<uses-sdk>` указывается минимально допустимая версия Android SDK для обеспечения нормальной работы приложения на мобильном телефоне. Атрибут `android:minSdkVersion` этого тега представляет минимально допустимую версию SDK в виде целого числа. В табл. 6.5 представлен список доступных версий.

Таблица 6.5 Доступные версии Android SDK

Версия Android sdk	Значение
Android 1.0	SDK 1 1
Android 1.1	SDK 2 2
Android 1.5	SDK 3 3
Android 1.6	SDK 4 4
Android 2.0	SDK 5 5
Android 2.0.1	SDK 6 6
Android 2.1	SDK 7 7

Например, можно указать минимальную версию для Android SDK 2.1:

```
<uses-sdk
android:
minSdkVersion="7"
/>
```

Номер версии для новых релизов Android SDK вы можете найти в сопроводительных комментариях. Во многих инструментах разработки этот номер часто упоминается как API Level (Уровень API), особенно в Android SDK and AVD Manager.

Именованное приложение. Атрибут `android:label` тега `<application>` хранит строку, представляющую имя приложения. Вы можете определить это имя в виде фиксированной строки, как в следующем примере:

```
<application
android:
label="My application
name">
```

Как вариант, в атрибуте `android: label` вы можете указать ссылку на строковый ресурс. В проекте Droid 1 имени приложения соответствует строковый ресурс:

```
<application
android:
label="@string/app_
name">
```

В этом случае, имя приложения будет хранить строковый ресурс `app_name` в файле `strings.xml`.

Присвоение значка приложению. Атрибут `android:icon` тега `<application>` хранит ссылку на строковый ресурс, представляющий собой значок приложения. В проекте Droid1 вы можете задать значку приложения графический ресурс следующим образом:

```
<application
android:
```



```
icon="@drawable/  
icon">
```

Выполните самостоятельно.

Несмотря на то, что вы поместили единственную ссылку на значок в атрибуте android: Icon тега <application>, вам необходимо создать три различных значка. Каждый из них предназначен для экрана с определенной плотностью распределения пикселей относительно физических размеров экрана: для экранов с низкой плотностью (ldpi). Для экранов с высокой плотностью (hdpi) и экранов со средней плотностью (mdpi) (в версии Android 2.2 введено новое значение плотности — xhdpi, т.е. сверхвысокая плотность. Система Android будет автоматически выбирать наиболее подходящее изображение исходя из характеристик устройства, на котором запущено приложение.

Сейчас вы можете просто создать одно графическое изображение и изменить его размеры для трех различных значений плотности распределения пикселей.

Чтобы создать собственный значок приложения, выполните следующие шаги:

1. Создайте изображение размером 48 x 48 точек в любом графическом редакторе, который вам нравится. Это изображение будет значком для экранов со средней плотностью.
2. Сохраните изображение в формате PNG с именем myicon.png.
3. Добавьте этот файл в папку /res/drawable-mdpi в качестве графического ресурса вашего приложения.
4. Повторите шаги 1-3, но теперь для размера 72x72 точки и поместите получившийся файл в папку /res/drawable-hdpi. Это значок для экранов с высокой плотностью.
5. Повторите шаги 1-3 для размеров 36x36 точек и поместите графический файл в папку /res/drawable-ldpi. Это значок для экранов с низкой плотностью точек.
6. Установите значение атрибута android: icon, чтобы оно соответствовало имени ресурса вашего значка – @drawable/myicon. Система сама выберет одно из трех графических изображений, исходя из плотности размещения точек на экране мобильного телефона (или эмулятора), на котором запущено приложение.

В результате будет выводиться один и тот же значок, но в трех различных разрешениях из трех различных ресурсных папок. Как правило, впоследствии изображение автоматически оптимизируется для каждого разрешения.

Описание приложения. Значение атрибута android:description тега <application> строка, содержащая краткое описание приложения. Вы можете также указать в этом атрибуте ссылку на строковый ресурс:

```
<application  
android:  
label="My application name"  
android:  
description="@string/app_  
desc">
```

Система Android и электронные системы дистрибуции используют это описание для отображения информации о приложении.

Установка отладочной информации для приложения. Атрибуту android:debuggable тега <application> присваивается булево значение, которое разрешает или запрещает отладку приложения отладчиком, например Eclipse. Вы не сможете проводить отладку приложения, если значение этого атрибута не равно true. Не забудьте перед публикацией приложения присвоить этому атрибуту значение false.

Установка других атрибутов приложения. На вкладке Application (Приложение) вы можете найти еще несколько дополнительных настроек, но они применяются довольно редко. Например, если вы хотите применить к приложению вместо оформления по умолчанию какое-либо другое. Здесь есть также настройки для управления тем, как приложение будет взаимодействовать с операционной системой Android- Для большинства приложений вы можете оставить эти настройки без изменений.

На вкладке Application (Приложение) вам предстоит выполнить множество действий в разделе Application Nodes (узлы приложения), где вы можете зарегистрировать компоненты приложения каждый раз, когда вы регистрируете новую деятельность.

Определение деятельности. Как вы уже знаете, приложения Android включают множество различных деятельности.

Каждая деятельность перед использованием должна быть зарегистрирована в файле манифеста Android. Вы должны обновлять файл манифеста каждый раз при добавлении новой деятельности к приложению.

Каждая деятельность представляет собой определенную задачу для выполнения, часто связанную с определенным экраном. Деятельность может быть запущена различными способами с помощью интенгов. Каждая деятельность может иметь свою собственную метку (имя) и значок, но по умолчанию метка и значок приложению уже назначены.

Регистрация деятельности. Вы можете регистрировать новые деятельности в разделе Application Nodes (Узлы приложения) вкладки Application (Приложение). Каждая новая деятельность обладает собственным тегом <activity> в получившемся XML- файле. Например, в следующем фрагменте кода XML происходит определение класса деятельности DroidActivity:

```
<activity
  android:
  name=".DroidActivity"
/>
```

Эта деятельность должна быть определена как класс пакета приложения.

Если вы добавляете новый элемент при выделенной активности, то он будет добавлен в эту деятельность. Поэтому следите, чтобы не было выделенных деятельности, или устанавливайте переключатель в положение Create a new element at the top level, in Application (Создать новый элемент на верхнем уровне, в приложении) вверху диалогового окна.

Определение активности запуска. С помощью интент-фильтра вы можете определить деятельность как основную точку входа приложения. Интент-фильтр запуска деятельности по умолчанию должен быть сконфигурирован в теге <intent-filter> с типом действия MAIN и категорией LAUNCHER. В вашем проекте Name1 мастер проектов Android установил деятельность Name1Activity как основную точку запуска приложения:

```
<activity
  android:
  name=". Name1Activity"
  android:
  label="@string/app_
  name">
  <intent-filter>
  <action
  android:
  name="android.intent.action.MAIN"
  />
  <category
  android:
  name="android.intent.category.LAUNCHER"
  />
</intent-filter>
</activity>
```

Таким образом, тег <intent-filter> посылает системе Android что все запросы запуска приложения должны направляться деятельности Name1Activity.

Управление правами приложения. Платформа Android основывается на ядре операционной системы Linux, известной своей встроенной системой безопасности, ставшей частью модели безопасности Android.

Каждое приложение Android существует в собственной виртуальной машине и работает в собственной учетной записи пользователя Linux.

Для приложений, требующих доступа к совместно используемым или привилегированным ресурсам на мобильном телефоне, должны быть объявлены определенные права в файле манифеста Android. Такой принцип обеспечения безопасности гарантирует, что никакое приложение не может изменить свое поведение во время выполнения или выполнить какие-либо операции без разрешения пользователем.

Поскольку каждое приложение работает под индивидуальной учетной записью, у каждого приложения есть свои файлы и папки так же, как это реализовано для пользователей в операционной системе Linux.

Приложения Android могут получить доступ к собственным файлам и базам данных без наличия специальных прав. Однако, если приложению необходимо получить доступ к совместно используемым или особым ресурсам, права, разрешающие такие действия, должны быть указаны в теге `<uses-permission>` файла манифеста Android.

Управлять правами вы можете на вкладке Permissions (Права) в редакторе ресурсов файла манифеста Android.

Выполните самостоятельно.

Чтобы предоставить вашему приложению право доступа ко встроенной камере, выполните следующие шаги:

1. Откройте проект в Eclipse.
2. Откройте файл манифеста Android и перейдите на вкладку Permissions (Права) ресурса редактор.
3. Нажмите кнопку Add (Добавить) и выберите пункт Uses Permission (Права пользования). Поле ввода атрибута Name (Имя) отображается в правой стороне экрана.
4. Введите имя `android.permission`.
5. CAMERA.
6. Сохраните файл манифеста. Переключитесь на вкладку AndroidManifest.xml, чтобы увидеть изменения в коде XML.
7. Таким образом, вы зарегистрировали разрешение на пользование камерой. Теперь ваше приложение может получать доступ с соблюдением прав безопасности.

В табл. 6.6 перечислены некоторые из наиболее часто используемых прав приложениями Android.

Таблица 6.6 Права, требуемые приложениями Android наиболее часто

Категория прав	Права
Геолокационные сервисы	<code>android.permission.ACCESS_COARSE_LOCATION</code> <code>android.permission.ACCESS_FINE_LOCATION</code>
Доступ к базе данных контактов	<code>android.permission.READ_CONTACTS</code> <code>android.permission.WRITE_CONTACTS</code>
Доступ к органайзеру	<code>android.permission.READ_CALENDAR</code> <code>android.permission.WRITE_CALENDAR</code>
Изменение общих настроек мобильного телефона	<code>android.permission.SET_ORIENTATION</code> <code>android.permission.SET_TIME_ZONE</code> <code>android.permission.SET_WALLPAPER</code>
Голосовой вызов	<code>android.permission.CALL_PHONE</code> <code>android.permission.CALL_PRIVILEGED</code>
Отправка и прием сообщений	<code>android.permission.READ_SMS</code> <code>android.permission.RECEIVE_MMS</code> <code>android.permission.RECEIVE_SMS</code> <code>android.permission.RECEIVE_WAP_PUSH</code> <code>android.permission.SEND_SMS</code>

	android.permission.WRITE_SMS
Использование сетевых соединений	android.permission.INTERNET
Доступ к аудионастройкам	android.permission.RECORD_AUDIO android.permission.MODIFY_AUDIO_SETTINGS
Доступ к настройкам сети	android.permission.ACCESS_NETWORK_STATE android.permission.CHANGE_NETWORK_STATE
Доступ к настройкам WiFi	android.permission.ACCESS_WIFI_STATE android.permission.CHANGE_WIFI_STATE
Доступ ко встроенным устройствам Мобильного телефона	android.permission.BLUETOOTH android.permission.CAMERA android.permission.FLASHLIGHT android.permission.VIBRATE android.permission.BATTERY_STATS
Службы учетных записей	android.permission.GET_ACCOUNTS android.permission.MANAGE_ACCOUNTS
Синхронизация	android.permission.READ_SYNC_SETTINGS android.permission.READ_SYNC_STATS android.permission.WRITE_SYNC_SETTINGS

Полный список прав, используемых приложениями Android, вы можете найти в документации по классу `android.Manifest.permission`.

Некоторые права еще не были реализованы в системе Android. Тем не менее приложения всегда должны запрашивать права на выполнение действий для совместимости с последующими версиями системы.

Приложения могут определять и осуществлять свои собственные права. Это может быть очень важно для определенных типов приложений, таких как банковская деятельность и коммерческие приложения.

Управление другими настройками приложения. Помимо возможностей, о которых рассказано в этом часе, в файле манифест Android можно сконфигурировать многое другое. Например, если вашему приложению требуется аппаратная клавиатура или сенсорный экран, вы можете указать эти требования в файле манифеста.

Вы также должны объявлять здесь все используемые приложением компоненты — например, если ваша программа предоставляет доступ к каким-либо услугам, контенту или функциям приемника.

Итоги. Файл манифеста Android (`AndroidManifest.xml`) присутствует в корневой папке каждого проекта. Это незаменимый компонент любого проекта. В файле манифеста Android используется простая XML-схема для описания самого приложения, его компонентов и прав. Эта информация используется платформой Android для управления приложением. В Eclipse есть удобный редактор ресурсов для управления файлами манифеста Android.

7 Разработка игрового приложения

7.1 Проектирование каркаса игрового приложения ANDROID

Поскольку в последнее время большую популярность приобрели простые социальные игры, в качестве примера приложения мы разработаем одну из таких игр. В процессе нашей игры пользователю будут задаваться случайные вопросы о том, в каких путешествиях он побывал и какой получил опыт, например:

- Бывали ли вы когда-нибудь у египетских пирамид?
- Вы когда-нибудь доили корову?
- Занимались ли вы когда-нибудь дайвингом с белыми акулами?
- Вы когда-нибудь поднимались на высокую скалу?

Игрок, прошедший большую часть игры и достигший наилучших результатов, должен обладать самым высоким счетом. Игра будет называться «Been There, Done That!».

Определение высокоуровневых игровых функций. Попробуйте представить, какими возможностями должно обладать хорошее игровое приложение. Кроме игрового экрана с вопросами об игре вам понадобятся:

- экранная заставка;
- вывод на экран заработанных игроком очков;
- вывод текста с разъяснением игровых правил;
- возможность сохранения игровых настроек.

Вы также должны придумать способ перехода между различными экранами игрового интерфейса. Один из способов решения этой проблемы — создать экран главную меню, с помощью которого пользователь может перемещаться между экранами приложения.

Исходя из перечисленных требований, в игровом приложении «Been There, Done That!»* нам может понадобиться шесть основных экранов:

- экран-заставка;
- экран главного меню;
- основной игровой экран;
- экран настроек;
- экран таблицы рекордов;
- экран справки.

Эти экраны составят основу пользовательского интерфейса приложения.

Требования, предъявляемые к деятельности. У каждого экрана игрового приложения «Been There, Done That!» будет собственный класс Activity.

Практика показывает, что удобнее реализовать базовый класс Activity с общедоступными компонентами. Он будет называться QuizActivity. Таким образом, будут определяться все деятельности, необходимые в приложении «Been There, Done That!»:

- Класс QuizActivity, полученный из android, app. Activity, будет основным классом. С его помощью вы определите параметры приложения и другие настройки и возможности приложения.
- Класс QuizSplashActivity, полученный из класса QuizActivity, представляет собой экран-заставку.
- Класс QuizMenuActivity, полученный из класса QuizActivity, представляет собой экран главную меню.
- Класс QuizHelpActivity, полученный из класса QuizActivity, представляет собой экран справки.
- Класс QuizScoresActivity, полученный из класса QuizActivity, представляет собой экран таблицы рекордов.
- Класс QuizSettingsActivity, полученный из класса QuizActivity, представляет собой экран настроек.

- Класс QuizGameActivity, полученный из класса QuizActivity, представляет собой основной игровой экран.

Определение игровых возможностей для каждого экрана Теперь пришло время определить основные возможности каждой деятельности в игровом приложении «Been There, Done That!».

Определение возможностей экрана-заставки. Экран-заставка в игре «Been There, Done That!» служит отправной точкой для последующих действий игрока. Вся его функциональность должна заключаться в классе QuizSplashActivity. На этом экране будут выполняться следующие действия:

- отображение имени и версии приложения;
- отображение графического изображения или логотипа игры;

Определение возможностей экрана главного меню. Экран главного меню служит для навигации по возможностям приложения. Этот экран появляется сразу после заставки и предоставляет пользователю выбор дальнейших действий. Его функциональность заключается в классе QuizMenuActivity. Этот экран должен:

- автоматически отображаться после заставки;
- предоставлять пользователю возможность выбора пунктов меню Play Game, Settings, Scores или Help.

Определение возможностей экрана справки. На экране справки пользователь может ознакомиться с правилами игры. Эта функциональность заключается в классе QuizHelpActivity. На этом экране должны выполняться следующие операции:

- отображение текста справки и предоставление возможности выполнять прокрутку текста;
- предоставление навигации по содержанию справочной информации.

Определение возможностей экрана таблицы рекордов. В таблице рекордов пользователи могут видеть текущий игровой счет с перечнем игроков, заработавших наибольшее количество очков. Эта функциональность должна быть заключена в классе QuizScoresActivity. На этом экране должны выполняться следующие операции:

- отображение статистики лучших игровых достижений;
- отображение только текущего счета, если пользователь перешел на этот экран в процессе игры.

Определение возможностей экрана настроек. На экране настроек пользователи могут редактировать и сохранять параметры игры, включая имя пользователя и другие. Эта функциональность заключается в классе QuizSettingsActivity. Здесь должны выполняться следующие операции:

- изменение игровых настроек пользователем;
- приглашение других игроков для совместной игры.

Определение возможностей основного игрового экрана. На игровом экране будут отображаться вопросы. Эта функциональность должна быть заключена в классе QuizGameActivity. На этом экране должны выполняться следующие действия:

- вывод серии вопросов да/нет;
- обработка вводимых данных, сохранение текущего счета и состояния игры;
- переход к экрану счета после завершения игры.

Реализация прототипа приложения. Теперь, когда у вас есть общее представление о том, как будет выглядеть программа «Been There, Done That!» и как она будет работать, настало время для написания кода.

Этот процесс состоит из нескольких этапов:

1. Создание ноуя проекта Android в Eclipse/
2. Добавление новых ресурсов проекта, включая текст и графику.
3. Создание разметки для каждого экрана.
4. Реализация деятельности для каждого экрана

5. Настройка параметров для всего приложения.

Создание нового проекта Android. Создать новый проект приложения вы можете с помощью мастера проектов Android в Eclipse.

При создании проекта вам необходимо ввести следующие данные:

- Project name (Название проекта): TriviaQuiz.
- Build target (Целевая платформа): Android 2.1.
- Application name (Название приложения): Been There. Done That!.
- Package name (Название пакета): com.androidbook.triviaquiz.
- Create activity (Создать деятельность): QuizSplashActivity.

Это лишь основные параметры нового проекта Android. Кроме того, вам необходимо сделать еще несколько корректировок.

Добавление ресурсов в проект. В проекте «Been There, Done That!» необходимо добавить несколько дополнительных ресурсов. В частности, вам нужно добавить файл макета для каждой деятельности и текст для каждого имени деятельности. Еще неплохо бы изменить значок приложения на более подходящий.

Добавление строковых ресурсов. Сначала необходимо отредактировать файл ресурса strings.xml. Удалите строку с именем hello и создайте шесть новых строковых ресурсов — по одному для каждого экрана.

Например, вы можете создать строковый ресурс help со значением Help Screen. После выполнения этих действий файл strings.xml должен выглядеть так:

```
<?xml
version="1.0" encoding="utf-
8"?>
<resources>
<string
name="app_
name">Been There, Done That!</string>
<string
name="
help">Help Screen</string>
<string
name="
menu">Main Menu Screen</string>
<string
name="
splash">Splash Screen</string>
<string
name="
settings">Settings Screen</string>
<string
name="
game">Game Screen</string>
<string
name="
scores">Scores Screen</string>
</resources>
```

Добавление ресурсов макета. Далее вам нужно создать файл макета для каждой деятельности. Переименуйте файл макета main.xml в splash.xml. Затем сделайте пять копий файла splash.xml и присвойте этим копиям следующие имена: game.xml, help.xml, menu.xml, scores.xml и settings.xml.

Вы можете встретить предупреждение ошибке в каждом файле макета. Ошибка заключается в том, что элемент управления TextView в макете ссылается на несуществующую строку @string/hello, которую вы удалили. Поэтому в каждом файле макета вы должны изменить ссылку на ресурсы, загружаемые элементом управления TextView. Например, в файле game.xml вам необходимо заменить ссылку на ресурс

@string/hello новой строкой — @string/game. Теперь после загрузки файлов разметки они должны отображать соответствующий экран.

Добавление графических ресурсов. Нам нужно заменить значок приложения на более подходящий. Например, вы можете создать значок quizicon.png размером 48x48 из исходного снимка Земли со спутника и добавить в папку ресурсов /drawable. После этого удалите файл icon.png, используемый по умолчанию.

Реализация деятельности приложения. Чтобы реализовать основной класс Activity, вы можете просто скопировать исходный файл QuizSplashActivity.java. Затем создайте новый файл класса QuizActivity. Этот класс должен выглядеть очень просто:

```
package com.androidbook.triviaquiz6;
import android.app.Activity;
public class QuizActivity extends Activity {
    public static final String
    GAME_PREFERENCES
    = "GamePrefs";
}
```

Остальное вы добавите в этот класс позже, Обновите класс QuizSplashActivity, чтобы расширить его из класса QuizActivity вместо класса Activity напрямую.

Создание остальных деятельности приложения. Скопируйте деятельность QuizSplashActivity пять раз — по одному разу для каждой активности: QuizMenuActivity, QuizHelpActivity, QuizScoresActivity, QuizSettingsActivity и QuizGameActivity.

Обратите внимание, как удобно Eclipse обновляет имя класса при копировании файла класса. Создавать файлы классов вы можете также, щелкнув правой кнопкой по имени пакета com.androidbook.triviaquiz и выбрав команду New Class (Новый Класс).

В открывшемся диалоговом окне вы можете выбрать параметры создаваемого класса.

Обратите внимание, что сейчас в каждом Java-файле присутствует ошибка. Она возникает из-за того, что каждая деятельность пытается загрузить файл макета main.xml, которого больше нет. Вы должны изменить каждый класс, чтобы иметь возможность загружать соответствующие макеты, связанные с деятельностью. Например, в классе QuizHelpActivity вы должны изменить метод setContentView(), чтобы загружать файл макета, созданного вами для экрана справки, следующим образом:

```
setContentView(R.layout.help);
```

Сделайте аналогичные изменения в других файлах деятельности, чтобы при каждом вызове метода setContentView() загружался соответствующий ему файл макета.

Обновление файла манифеста ANDROID. Теперь вам необходимо сделать некоторые изменения в файле манифеста Android. Впервых, вам надо изменить ресурс значка приложения, чтобы ссылка @drawable/quizicon указывала на созданный вами значок. Во-вторых, чтобы новые деятельности выполнялись правильно, их нужно зарегистрировать в файле манифеста.

Наконец, вы должны установить значение true в атрибуте приложения Debuggable и убедиться, что деятельность QuizSplashActivity является деятельностью запуска по умолчанию.

В нашем случае мы создали только один значок. Тем не менее, если вы уже создали три значка разных размеров и поместили их в папки по умолчанию (/drawable-ldpi, /drawable-mdpi и /drawable-hdpi), использоваться будет только один значок. Обязательно убедитесь, что все значки называются одинаково. Таким образом, система Android может автоматически выбрать подходящий значок с учетом аппаратных особенностей устройства, на котором запущено приложение.

Установка параметров приложения. В игре «Been There, Done That!» вам нужен простой механизм сохранения информации о состоянии и пользовательских данных. Эту возможность вы можете реализовать с помощью общедоступных параметров системы Android (android.content.SharedPreferences).

Получить доступ к общедоступным параметрам вы можете по его имени из любой деятельности приложения. Для этого объявите имя набора параметров в базовом классе QuizActivity так, чтобы они стали доступны всех подклассов:

```
public static final String  
GAME_PREFERENCES = "GamePrefs";
```

Количество наборов параметров, которые вы можете создать, не ограничено. Разделить параметры по категориям вы можете строкой имени параметра, например игровые параметры и параметры пользователя. То, как будут организованы общедоступные параметры, зависит от вас.

Чтобы добавить общедоступные параметры в приложение, выполните следующие шаги:

1. Методом `getSharedPreferences()` получите экземпляр объекта `SharedPreferences`.
2. Создайте объект `SharedPreferences.Editor`, чтобы изменить параметры.
3. Сделайте изменения в параметрах с помощью редактора.
4. Примените изменения методом `commit()` в редакторе.

Сохранение общедоступных параметров. Каждый параметр представляет собой пару ключ/значение. Значения параметров могут быть следующих типов:

- булевый;
- число с плавающей точкой;
- целое число;
- длинное целое число;
- строка.

После того, как вы решили, какие параметры хотите сохранять, вы должны получить экземпляр объекта `SharedPreferences`, затем с помощью объекта `Editor` сделать изменения и сохранить их. В следующем примере сохраняются два параметра — имя пользователя и его возраст:

```
import android.content.SharedPreferences;  
// ...  
SharedPreferences settings =  
getSharedPreferences(  
    GAME_  
    PREFERENCES,  
    MODE_  
    PRIVATE);  
SharedPreferences.Editor prefEditor = settings.edit();  
prefEditor.putString("UserName", "JaneDoe");  
prefEditor.putInt("UserAge", 22);  
prefEditor.commit();
```

Вы можете также использовать редактор общедоступных параметров для их сброса методом `clear()` и удаления определенных параметров по имени методом `remove()`.

Получение общедоступных параметров. Получение значений общедоступных параметров намного проще, чем их создание, потому что для этого вам не нужен редактор. В следующем примере показано, как можно получить доступ к значениям общедоступных параметров:

```
SharedPreferences settings =  
getSharedPreferences(  
    GAME_  
    PREFERENCES,  
    MODE_  
    PRIVATE);  
if (settings.contains("UserName") == true) {  
    // We have a user name  
    String user = Settings.getString("UserName", "Default");  
}
```

Вы можете использовать объект `SharedPreferences` для проверки параметров по имени, получить строго введенные параметры или получить все параметры и сохранять их на карте.

Несмотря на то, что в игре «Been Three, Done That!» общедоступные параметры пока не используются, полученные знания о них помогут вам при разработке активностей приложения.

Подготовка прототипа игры. Уже сейчас вы почти готовы к выполнению и тестированию вашего приложения. Но сначала вы должны создать конфигурацию отладки для вашего нового проекта в Eclipse.

Создание конфигурации отладки. Вы должны создать в Eclipse конфигурацию отладки для каждого проекта. Убедитесь, что выбранное для проекта AVD совместимо с Google API (API Level 7). Если у вас нет такого AVD (или вы не уверены в этом), нажмите на кнопку вызова Android SDK and AVD Manager в Eclipse. Здесь вы можете определить, какие AVD лучше всего подходят приложению, и создать новые, если необходимо.

Запуск прототипа в эмуляторе. Теперь все готово для запуска приложения «Been There, Done That!» в эмуляторе Android.

Вы можете сделать это, щелкнув по кнопке отладки в Eclipse или на кнопке Run (Выполнение) в созданной только что конфигурации отладки.

В приложении пока еще нет никакого пользовательского интерфейса, поэтому вы должны переладить между различными деятельностью вручную. Однако у вас уже есть готовый каркас приложения для последующей разработки.

Установка прототипа приложения. Приложение «Been There, Done That!» в текущем виде выполняет довольно мало задач, но с помощью эмулятора Android и других инструментов вы можете оценить результат своей работы.

- Application Manager (Менеджер приложений) — полезен для поиска интересной информации о приложении. В эмуляторе переместитесь на экран по умолчанию, нажмите на кнопку Menu (Меню), выберите команду `Settings=>Applications=>Manage applications` (Настройки=>Приложение=>Управление приложениями) и выберите приложение «Been There, Done That!» из списка. Здесь вы можете видеть основную информацию о приложении, включая занимаемый объем и используемые права, а также информацию о кэше и т.д.
- Dev Tools (Средства разработки) - с их помощью вы можете получить более подробную информацию о приложении. Зайдите в эмулятор, перечитайте в меню приложений, запустите приложение Dev Tools (Средства разработки) и выберите Package Browser (Обзор пакетов). Переместитесь к пакету `com.androidbook.triviaquiz`.

С помощью этого средства вы можете увидеть информацию о манифесте и проверить настройки каждой зарегистрированной активности. Разумеется, вы можете исследовать приложение в перспективе DDMS. Например, вы можете проверить папку приложения пакета `com.androidbook.triviaquiz` в файловой системе Android. Вы можете также рассмотреть код деятельности `QuizSplashActivity`.

Итоги. В этом разделе вы создавали прототип приложения, на основе которого в последующем вы можете собрать полноценное приложение. Вы спроектировали прототип и подробно определили его требования. Потом вы создали новый проект Android, сконфигурировали его и создали деятельность для каждого экрана. Вы также добавили собственные макеты и применили общедоступные параметры в приложении.

7.2 Реализация анимированного экрана-заставки

В этом разделе мы сосредоточимся на реализации экрана-заставки для приложения «Been There, Done That!». Создав эскиз дизайна экрана-заставки, вы сможете точно определить, какие элементы пользовательского интерфейса View платформы Android вам понадобится включить в файл макета `splash.xml`. Когда вы будете удовлетворены

внешним видом макета экрана- заставки. вы сможете добавить несколько анимаций движения, чтобы придать экрану-заставке некую пикантность. Наконец, вам нужно создать плавный переход между экраном-заставкой и экраном с основным меню, когда воспроизведение ваших анимаций будет завершено.

Разработка дизайна экрана-заставки. Вы будете реализовывать приложение «Been There, Done That!», начав с экрана-заставки, который будет виден пользователям сразу после запуска приложения. Как было отмечено в разделе 7.1, к этому экрану есть несколько требований. В частности, на нем должна отображаться определенная информация о приложении (его название и версия), и затем, по истечении небольшого промежутка времени, должен происходить автоматический переход к экрану основного меню.

Для начала разработайте экран-заставку в портретной ориентации, но сразу подумайте о том, как максимально упростить последующий переход на альбомную или квадратную ориентацию. Разработав простой дизайн экрана-заставки, вы можете быть уверены, что в других ориентациях элементы экрана будут довольно хорошо масштабироваться при условии, что вы настроите размеры шрифтов и изображений. Проблемы, связанные с поддержкой различных устройств, рассматриваются в последующих часах.

Реализация макета заставки. Теперь, когда вы знаете, как должен выглядеть экран-заставка вашего приложения, нужно преобразовать эскиз в соответствующий дизайн-макет. Напомним, что файл макета `/res/layout/splash.xml` используется деятельностью `QuizSplashActivity`. Вы должны обновить стандартный макет, на котором отображается один элемент `TextView` (сообщая нам о том, что это экран-заставка), чтобы добавить элементы пользовательского интерфейса для каждого элемента, представленного на эскизе дизайна.

Макеты экранов формируются при помощи разнообразных элементов пользовательского интерфейса. Каждый элемент — прямоугольник, контролирующий определенную часть экрана. На экране-заставке используются два распространенных элемента пользовательского интерфейса:

- Элемент `TextView` используется для отображения на экране текстовой строки.
- Элемент `ImageView` используется для отображения на экране изображения.

Вам также нужен способ для организации различных элементов пользовательского интерфейса `View` в определенном порядке. Для этого применяются элементы-контейнеры `Layout`. Например, элемент-контейнер `LinearLayout` позволяет расположить дочерние элементы-представления друг за другом в вертикальном или горизонтальном направлении.

Помимо элемента-контейнера `LinearLayout` существует ряд других элементов контейнеров `Layout`. Элементы-контейнеры могут быть вложенными и контролировать либо некоторую часть экрана, либо весь экран. Довольно часто элементы пользовательского интерфейса размещаются в одном большом родительском элементе контейнера — например, `LinearLayout`. В табл. 7.1 перечислены доступные элементы контейнеров `Layout`.

Распространенные элементы-контейнеры `Layout`:

- Ключевые атрибуты/элементы `LinearLayout`. Каждый дочерний элемент-представление размещается за предыдущим элементом-представлением в одной строке или столбце
- Ориентация (вертикальная или горизонтальная) `RelativeLayout`. Каждый дочерний элемент-представление размещается относительно других элементов-представлений в макете или относительно границ родительского элемента-контейнера. Множество атрибутов выравнивания, позволяющих управлять размещением дочернего элемента-представления относительно других дочерних элементов-представлений.

- **View FrameLayout.** Каждый дочерний элемент-представление размещается внутри фрейма, относительно его левого верхнего угла. Элементы-представления View могут перекрываться. Порядок размещения дочерних элементов-представлений View имеет значение, когда используются соответствующие значения атрибута.
- **Gravity TableLayout.** Каждый дочерний элемент-представление размещается в одной из ячеек сетки, образуемой строками и столбцами. Каждая строка представляется элементом TableRow.

В инструментарии Android 1.5 SDK элемент-контейнер **AbsoluteLayout**, позволяющий указывать определенные значения координат X/Y для элементов, был объявлен устаревшим, однако он до сих пор доступен для использования. И хотя использовать этот элемент-контейнер не рекомендуется, иногда полезно иметь элемент-контейнер с возможностью указания точных координат X/Y для определенных типов макетов экранов, которые требуют точного размещения элементов по пикселям. Этот элемент контейнер используется в элементе пользовательского интерфейса **WebView** инструментария SDK.

Элементы-контейнеры и их дочерние элементы-представления. View имеют определенные атрибуты, которые помогают управлять их поведением. Например, все элементы-контейнеры имеют атрибуты `android:layout_width` и `android:layout_height`, позволяющие управлять шириной и высотой элемента-контейнера. В качестве значений этих атрибутов могут использоваться конкретные измерения, например число пикселей, или может применяться более гибкий подход с использованием значений `fill_parent` или `wrap_content`. Когда используется значение `fill_parent`, элемент-контейнер масштабируется до размеров родительского элемента-контейнера, а использование значения `wrap_content` позволяет «растянуть» родительский элемент-контейнер вокруг дочернего элемента-представления View, подогнав размеры родительского элемента-контейнера до размеров дочернего элемента представления View. Существует ряд других интересных свойств, которые могут быть использованы для управления поведением отдельных элементов-контейнеров, включая настройки отступов и атрибуты, характерные для конкретных типов элементов контейнеров.

Давайте воспользуемся элементом-контейнером **TableLayout**, чтобы отобразить несколько элементов-представлений **ImageView**, являющихся частью нашего экрана заставки.

В дизайне экрана-заставки вы можете использовать элемент-контейнер **LinearLayout** с вертикальной ориентацией, чтобы разместить элементы пользовательского интерфейса данного экрана в следующем порядке: элемент **TextView**, элемент-контейнер **TableLayout** с несколькими элементами **TableRow**, содержащими элементы **ImageView**, и два дополнительных элемента **TextView**.

Добавление новых ресурсов в проект. Теперь, когда у вас есть дизайн-макет экрана-заставки для приложения, настало время создать строковые ресурсы, ресурсы цветов и размеров, которые будут использованы в макете.

Сначала вы добавите четыре новых графических ресурса в каталог `/res/drawable` (создавая при необходимости этот каталог): `splash1.png`, `splash2.png`, `splash3.png` и `splash4.png`. Эти изображения будут отображаться внутри элемента-контейнера **TableLayout** в центре экрана-заставки.

Затем вы можете добавить три новых строковых ресурса в файл ресурсов `res/values/strings.xml`: один для верхнего заголовка (**Been There**), один для нижнего заголовка (**Done That!**) и один для информации о версии приложения (эта информация будет многострочной). Строковый ресурс `splash` теперь можно удалить, поскольку он больше не будет использоваться.

После этого вы создадите новый файл ресурсов `/res/values/colors.xml`, который будет содержать три необходимых ресурса цвета: один для текста заголовка (золотисто-

желтый), один для текста версии приложения (серовато-белый) и один для цвета фона, поверх которого будет отображаться текст версии (насыщенно-синий).

Поскольку общий цвет фона заставки будет черным, вам не нужно создавать отдельный ресурс цвета. Вместо этого вы просто воспользуетесь встроенным ресурсом платформы Android `@android:color/black`.

Наконец, вам нужно создать несколько ресурсов размеров в новом файле ресурсов `/res/values/dimens.xml`. Вам потребуется три новых значения размеров: один для управления размером шрифта заголовка (24pt), один для управления размером шрифта версии приложения (5pt) и один для задания промежутка между строками текста с информацией о версии приложения (3pt).

После того, как вы сохраните файлы ресурсов, можно приступить к использованию новых ресурсов в файле макета `splash.xml`.

Обновление макета экрана-заставки. Вы могли бы изменить существующий макет `splash.xml`, но иногда бывает проще начать с нуля, используя редактор ресурсов в среде разработки Eclipse, чтобы удалить все существующие элементы пользовательского интерфейса. Затем нужно выполнить следующие шаги, чтобы создать желаемый макет экрана в соответствии с разработанным дизайнмакетом:

1. Сначала добавьте новый элемент-контейнер `LinearLayout` и присвойте его Атрибуту `background` значение `@android:color/black`, а атрибуту `orientation` — значение `vertical`. Все остальные элементы пользовательского интерфейса будут добавляться в элемент-контейнер `LinearLayout`. Для добавления элементов-представлений `View` внутрь других элементов или для их перемещения удобно использовать панель `Outline` (Обзор).
2. Добавьте элемент `TextView` под названием `TextViewTopTitle`. Присвойте его атрибуту `layout_width` значение `fill_parent`, а атрибуту `layout_height` — значение `wrap_content`. Присвойте атрибуту `text` соответствующий строковый ресурс, атрибуту `textColor` — ресурс желтого цвета, а атрибуту `textSize` — соответствующий ресурс размера.
3. Добавьте элемент-контейнер `TableLayout` под названием `TableLayout01`. Присвойте атрибуту `layout_width` этого элемента значение `fill_parent`, а атрибуту `layout_height` — значение `wrap_content`. Также присвойте атрибуту `stretchColumns` значение `*`, которое позволяет при необходимости растягивать любой столбец, чтобы заполнить свободное пространство экрана.
4. Добавьте элемент `TableRow` внутрь добавленного элемента-контейнера `TableLayout`. Добавьте два элемента `ImageView` внутрь элемента `TableRow`. Для первого элемента `ImageView` присвойте атрибуту `src` значение `@drawable/splash1`, которое соответствует графическому ресурсу `splash1.png`. Добавьте второй элемент `ImageView` и присвойте его атрибуту `src` значение, соответствующее графическому ресурсу `splash2.png`.
5. Повторите шаг 4, создав второй элемент `TableRow`. Опять же добавьте два элемента `ImageView` для графических ресурсов `splash3.png` и `splash4.png`.
6. По аналогии с шагом 2 добавьте еще один элемент `TextView` под названием `TextViewBottomTitle` внутрь родительского элемента-контейнера `LinearLayout`. Присвойте его атрибуту `layout_width` значение `fill_parent`, а атрибуту `layout_height` — значение `wrap_content`. Присвойте атрибуту `text` соответствующий строковый ресурс, атрибуту `textColor` — ресурс желтого цвета, а атрибуту `textSize` — соответствующий ресурс размера.
7. Для отображения информации о версии последний элемент `TextView` под названием `TextViewBottomVersion`. Присвойте его атрибутам `layout_width` и `layout_height` значение `fill_parent`. Присвойте атрибуту `text` соответствующий строковый ресурс, атрибуту `textColor` — ресурс сероватого цвета, а атрибуту `textSize` — соответствующий ресурс размера. Также присвойте атрибуту

background соответствующий ресурс цвета (темно-синий) и атрибуту lineSpacingExtra – соответствующий ресурс размера, определяющий расстояние между строками.

8. Настройте значения атрибутов layout_gravity и gravity добавленных элементов пользовательского интерфейса, чтобы макет выглядел приемлемо в редакторе ресурсов среды разработки Eclipse. Теперь сохраните файл макета splash.xml и запустите приложение «Been There, DoneThat!» на эмуляторе Android.

На этом можно было бы остановиться, но в вашей заставке явно не хватает какой-то живинки. Кроме того, нужно добавить переход между экраном-заставкой и основным навигационным экраном.

Использование анимации. Отличный способ придать экспрессии вашему экрану-заставке — добавить движение. Платформа Android поддерживает четыре типа анимации: Анимированные GIF-изображения. Это графические файлы, состоящие из нескольких кадров.

- Покадровая анимация. Инструментарий Android SDK предоставляет механизм покадровой анимации, схожий с механизмом создания анимированных GIF-изображений, когда разработчик добавляет отдельные графические кадры и настраивает переходы между ними (дополнительную информацию можно найти в описании класса AnimationDrawable).
- Анимация движения. Анимация движения — это простой и универсальный способ определения анимационных действий, которые в дальнейшем могут быть применены к любому элементу-представлению или элементу-контейнеру.
- Интерфейс OpenGL ES. Интерфейс OpenGL ES платформы Android предоставляет широкие возможности для трехмерного рисования, создания анимаций, настройки освещения и наложения текстур.

Для вашего приложения лучше всего подходит анимация движения. Платформа Android поддерживает анимацию движения с применением альфа-канала, вращения, масштабирования и перемещения. Вы можете создавать наборы анимационных действий, которые могут выполняться одновременно, последовательно или в заданное время. Таким образом, анимация движения — отличный выбор для экрана-заставки.

Создать анимацию движения можно либо программным путем, либо добавив анимационный ресурс в каталог /res/anim. Для каждой анимационной последовательности необходимо создавать отдельный XML-файл, но при этом созданная анимация может применяться к любому числу элементов-представлений View.

Вы можете использовать анимационные клипы, доступные в классе android.R.anim.

Добавление анимационных ресурсов. Для заставки вам потребуется создать три пользовательских анимационных последовательности в виде XML-файлов и сохранить их в каталоге ресурсов /res/anim: fade_in.xml, fade_in2.xml и custom_anim.xml.

Первая анимация, представленная файлом fade_in.xml, плавно увеличивает непрозрачность (альфа-канал) целевого элемента от значения 0 (прозрачный) до значения 1 (непрозрачный) в течение 2500 миллисекунд, или 2,5 секунд. В среде Eclipse нет редактора анимации. Содержимое XML-файла fade_in.xml выглядит следующим образом:

```
<?xml
version="1.0" encoding="utf-8"
?>
<set
xmlns:
android="http://schemas.android.com/apk/res/android"
android:
shareInterpolator="
false">
<alpha
android:
fromAlpha="0.0"
```

```

android:
toAlpha="1.0"
android:
duration="
2500">
</alpha>
</set>

```

Вы можете применить эту анимацию к верхнему элементу TextView с названием вашего приложения.

Теперь создайте анимацию fade_in2.xml. Она делает то же самое, что и fade_in.xml, за исключением того, что атрибуту startOffset будет присвоено значение 2500 миллисекунд.

Это значит, что ее продолжительность будет составлять 5 секунд: после задержки, равной 2,5 секунды, начинается плавное отображение элемента-представления в течение 2,5 секунды. Поскольку для заставки пяти секунд вполне достаточно, но завершении анимации fade_in2.xml начинается переход к экрану с основным меню.

Наконец, вам потребуется некая забавная анимация для изображений внутри элементаконтейнера TableLayout. В данном случае ваш набор анимационных действий включает несколько операций, выполняющихся одновременно: вращение, масштабирование и изменение альфа-прозрачности. В результате целевой элемент представление View будет совершать полный оборот вокруг своего центра. Содержимое файла custom_anim.xml выглядит следующим образом:

```

<?xml version="1.0" encoding="utf-8" ?>
<set
xmlns:
android="http://schemas.android.com/apk/res/android"
android:
shareInterpolator="
false">
<rotate
android:
fromDegrees="0"
android:
toDegrees="360"
android:
pivotX="50%"
android:
pivotY="50%"
android:
duration="2000"
/>
<alpha
android:
fromAlpha="0.0"
android:
toAlpha="1.0"
android:
duration="
2000">
</alpha>
<scale
android:
pivotX="50%"
android:
pivotY="50%"
android:
fromXScale=".1"
android:
fromYScale=".1"
android:

```

```

toXScale="1.0"
android:
toYScale="1.0"
android:
duration="2000"
/>
</set>

```

Как видим, поворот на 360 градусов занимает 2 секунды, при этом вращение осуществляется вокруг центра элемента-представления. Операция по изменению альфа-прозрачности должна быть вам знакома; плавное изменение прозрачности происходит в течение тех же 2 секунд. Наконец, операция по изменению масштаба от 10% до 100% также происходит в течение двухсекундного интервала. Воспроизведение всей анимации занимает 2 секунды.

Сохранив все три файла анимации, вы можете приступить к применению анимаций к конкретным элементам-представлениям.

Анимация конкретных элементов-представлений. Применение и управление анимациями должно производиться программным путем. Не забывайте, что ресурсоемкие операции должны прекращаться, если выполнение приложения приостанавливается по какой-либо причине. Воспроизведение анимации может быть продолжено после того, как приложение снова перейдет в активное состояние.

Давайте начнем с простейшего случая: применим анимацию `fade_in.xml` к элементу-представлению `TextView` под названием `TextviewTopTitle`, используемого для отображения названия приложения. Все, что вам нужно, - это получить экземпляр вашего элемента `TextView` в методе `onCreate()` класса `QuizSplashActivity`, загрузить анимационный ресурс в объект `Animation` и вызвать метод `startAnimation()` элемента-представления `TextView`:

```

TextView logo1 = (TextView) findViewById(R.id.TextViewTopTitle);
Animation fade1 = AnimationUtils.loadAnimation(this, R.anim.
fade_
in);
logo1.startAnimation(fade1);

```

Если воспроизведение анимации должно быть остановлено — например, в методе `onPause()` соответствующей деятельности, — вы просто вызываете метод `clearAnimation()`. Например, следующий метод `onPause()` демонстрирует этот подход для угловых логотипов:

```

@Override
protected void onPause() {
super.onPause();
// Stop the animation
TextView logo1 = (TextView) findViewById(R.id.TextViewTopTitle);
logo1.clearAnimation();
TextView logo2 = (TextView)
findViewById(R.id.TextViewBottomTitle);
logo2.clearAnimation();
// ... stop other animations
}

```

Анимация всех элементов в элементе-контейнере Помимо возможности применения анимаций к отдельным элементам `View`, вы также можете применить анимации к каждому элементу `View`, содержащемуся в элементе контейнера (например, в элементе-контейнере `TableLayout` и к каждому элементу `TableRow`), используя класс `LayoutAnimationController`.

Чтобы анимировать элементы-представления `View` с использованием данного подхода, вы должны загрузить соответствующую анимацию, создать экземпляр класса `LayoutAnimationController`, произвести необходимые настройки этого экземпляра и затем вызвать метод `setLayoutAnimation()` элемента-контейнера. Например, следующий код загружает анимацию `custom_anim`, создает экземпляр класса `LayoutAnimationController` и

затем применяет эту анимацию к каждому элементу TableRow в элемент-контейнере TableLayout:

```
Animation spinin =
AnimationUtils.loadAnimation(this, R.anim.custom_anim);
LayoutAnimationController controller =
new LayoutAnimationController(spinin);
TableLayout table = (TableLayout) findViewById(R.id.TableLayout01);
for (int i = 0; i < table.getChildCount(); i++) {
    TableRow row = (TableRow) table.getChildAt(i);
    row.setLayoutAnimation(controller);
}
```

В данном случае нет необходимости вызывать метод startAnimation(), поскольку для вас это сделает экземпляр класса LayoutAnimationController. При использовании данного подхода анимация применяется к каждому дочернему элементу представлению, однако воспроизведение каждой анимации начинается в разное время. (По умолчанию задержка составляет 50% от длительности анимации — в данном случае эта задержка равна 1 секунде.) Это создает красивый эффект, когда каждый из элементов представлений ImageView поочередно делает полный оборот вокруг своего центра.

Остановка анимаций, реализуемых при помощи экземпляра класса LayoutAnimationController, ничем не отличается от остановки отдельных анимаций: для этого применяется метод clearAnimation(). Дополнительные строки кода, которые потребуются добавить в существующий метод onPause(), представлены ниже:

```
TableLayout table = (TableLayout) findViewById(R.id.TableLayout01);
for (int i = 0; i < table.getChildCount(); i++) {
    TableRow row = (TableRow) table.getChildAt(i);
    row.clearAnimation();
}
```

Обработка событий жизненного цикла анимаций. Теперь, когда вы не можете нарадоваться на ваши анимации, осталось создать переход между деятельностью QuizSplashActivity и QuizMenuActivity по завершении воспроизведения анимаций. Для этого вы создадите новый объект типа Intent, который будет создавать экземпляр класса QuizMenuActivity и вызывать метод startActivity(). Вы также должны вызывать метод finish() класса QuizMenuActivity, поскольку хранить эту деятельность в стеке нет необходимости (если вы не хотите, чтобы при нажатии на кнопку Back пользователь возвращался к экрану-заставке).

Из всех ваших анимаций самая длинная — fade_in2, продолжительностью 5 секунд. Таким образом, момент завершения воспроизведения этой анимации можно использовать in2 запуска процесса перехода к экрану с основным меню. Для этого нужно создать экземпляр класса AnimationListener, который имеет методы обработки событий для жизненного цикла анимации: начало, конец и повторение. В данном случае нас интересует только метод onAnimationEnd(). Ниже представлен код, который создает экземпляр класса AnimationListener и реализует метод обработки событий onAnimationEnd():

```
Animation fade2 = AnimationUtils.loadAnimation(this, R.anim.fade_in2);
fade2.setAnimationListener(new AnimationListener() {
    public void onAnimationEnd(Animation animation) {
        startActivity(new Intent(QuizSplashActivity.this,
            QuizMenuActivity.class));
        QuizSplashActivity.this.finish();
    }
});
```

Теперь, если снова запустить приложение «Been There, Done That!» на эмуляторе или на мобильном телефоне, вы увидите привлекательную анимацию на экране-заставке. После завершения воспроизведения анимации происходит плавный переход к экрану с основным меню, реализацией которого вы займетесь в следующем часе.

Итоги. Примите наши поздравления! Вы завершили работу над вашим первым экраном опроса викторины «Been There, Done That!». В этом часе вы разработали дизайн

экрана-заставки, а также определили подходящий дизайн- макет и элементы представления View, необходимые для реализации данного дизайна. После этого вы добавили необходимые ресурсы и внесли изменения в файл макета splash.xml. Наконец, вы добавили несколько анимаций движения на заставку и затем реализовали переход между деятельностью QuizSplashActivity и QuizManuActivity.

7.3 Реализация экрана с основным меню

В этом разделе вы узнаете о различных типах меню, доступных в операционной системе Android. Сначала вы создадите экран с основным меню для приложения «Been There, Done That!», используя новые элементы ListView и RelativeLayout. Затем вы познакомитесь с другими экранами, для которых могут подойти особые типы меню, например меню опций [2].

Разработка дизайна экрана с основным меню. Для создания экрана с основным меню вам понадобится сделать набросок того, как должен выглядеть этот экран. Если взглянуть на требования к этому экрану, вы увидите, что он должен обеспечивать основную навигацию по остальной части приложения.

Пользователям доступны четыре различных варианта: они могут сыграть в игру, ознакомиться с ее правилами, настроить параметры или просмотреть таблицу результатов.

Существует ряд различных подходов для реализации экрана с основным меню. Например, можно создать кнопку для каждого пункта меню, прослушивать события нажатий и переправлять пользователя на соответствующий экран. Тем не менее по мере увеличения количества пунктов меню масштабируемость данного подхода будет ухудшаться. Таким образом, более приемлемо использование меню в виде списка, реализованного при помощи элемента ListView. В этом случае, если количество элементов в списке выйдет за границы экрана, в нашем распоряжении окажется встроенная возможность прокрутки содержимого списка.

Навигационный экран нужно снабдить какими-то «бантиками», делающими его привлекательным. Сначала вы будете использовать стандартное поведение для каждого элемента макета, а затем добавите специальный стиль к этим элементам. Например, можно добавить красивое фоновое изображение для всего экрана и отдельное для активного элемента списка.

Наконец, вы добавите бизнес-логику для элемента ListView, чтобы гарантировать, что при нажатии на элемент списка будет открыт соответствующий экран. Это позволит пользователям легко переходить на другие экраны, которые вам потребуется реализовать для приложения «Been There, Done That!».

Определение требований к экрану с основным меню. Теперь, когда вы знаете, как будет выглядеть ваш экран с основным меню, мы необходимо преобразовать эскиз дизайна в соответствующий дизайн- макет. Вам потребуется обновить файл макета /res/layout/menu.xml, который используется в деятельности QuizMenuActivity. Для макета экрана с основным меню вам потребуется заголовок, за которым расположится элемент ListView и затем элемент ImageView.

Создание заголовка экрана при помощи элемента управления RelativeLayout.

Очевидно, что для отображения заголовка экрана можно использовать элемент TextView. Но для красоты можно добавить изображения по обе стороны от элемента TextView, не так ли? Самое время воспользоваться элементом-контейнером RelativeLayout, который позволяет расположить каждый дочерний элемент представления в определенной позиции относительно родительского элемента-контейнера или других дочерних элементов-представлений. Таким образом, вы можете легко создать заголовок экрана при помощи элемента-контейнера RelativeLayout и трех дочерних элементов-представлений:

- элемента ImageView, выровненного по левому верхнему углу родительского элемента-контейнера;

- элемента `TextView`, выровненного по центру верхнего края родительского элемента-контейнера;
- элемента `ImageView`, выровненного по правому верхнему углу родительского элемента-контейнера.

Добавление элемента `ListView`. Далее вы должны добавить в макет элемент `ListView`. Элемент `ListView` просто исполняет роль контейнера, который будет содержать список элементов `View`. По умолчанию `ListView` используется для размещения элементов `TextView`, однако элементы `ListView` могут содержать множество разнообразных элементов `View`.

Элемент `ListView`, содержащий элементы `TextView`, отлично подойдет для данного примера. Чтобы переопределить стандартное поведение для каждого элемента `TextView`, содержащегося в элементе `ListView`, вас потребуется определить ресурс макета, который будет служить шаблоном для каждого элемента `TextView` внутри элемента `ListView`. Также вы можете сделать ваше меню более интересным, определив собственный разделитель и селектор для элемента `ListView`.

Завершающие штрихи для макета экрана с основным меню. Чтобы полностью завершить разработку макета экрана с основным меню, вы добавите элемент управления `ImageView`, расположив его за элементом управления `ListView`.

Как и раньше, все элементы экрана необходимо поместить в элемент-контейнер `LinearLayout` с вертикальной ориентацией при этом элементы `RelativeLayout`, `ListView` и `ImageView` должны следовать друг за другом по вертикали.

Реализация макета экрана с основным меню.

Реализацию экрана с основным меню вы начнете с добавления новых ресурсов в проект. После этого вам потребуется обновить файл макета `menu.xml`, чтобы реализовать разработанный дизайн-макет экрана с основным меню.

Добавление новых ресурсов в проект. Теперь, когда у вас есть дизайн-макет экрана, вам нужно создать графические ресурсы, строковые ресурсы, ресурсы цветов и размеров, которые будут использоваться в макете экрана с основным меню.

Сначала вы добавите четыре новых графических ресурса в каталог `/res/drawable`:

- `bkgrnd.jpg`,
- `divider.png`,
- `half.png`,
- `textured.png`.

В качестве фонового изображения для элемента-контейнера `LinearLayout` будет использован файл `bkgrnd.jpg`. В качестве разделителя и селектора для элемента `ListView` будут использованы файлы `divider.png` и `textured.png` соответственно. Для элемента `ImageView`, расположенного в нижней части экрана, будет использован файл `half.png`.

После этого вы добавите несколько новых и измените несколько существующих строковых ресурсов в файле `/res/values/strings.xml`, которые будут использованы для каждого пункта меню и для элемента `TextView`, представляющего заголовок экрана.

Наконец, вы обновите ресурсы цветов в файле `/res/menu/colors.xml`, чтобы определить цвета, которые будут использованы для заголовка экрана, отображаемого при помощи элемента `TextView`, а также для элементов `TextView`, расположенных внутри элемента `ListView`. Кроме того, вам также понадобится обновить ресурсы в файле `/res/values/dimens.xml`, чтобы определить размеры для текста заголовка и для текста, отображаемого при помощи элемента `ListView`.

В качестве значений конкретных ресурсов вы можете либо использовать значения, представленные в этой книге, либо указать свои собственные.

Сохранив файлы ресурсов, вы можете приступить к их использованию в макетах экрана с основным меню.

Обновление макетов экрана с основным меню. Возможно, вы уже заметили, что для экрана с основным меню используется множество файлов с ресурсами макета. Основной файл макета — `menu.xml` — определяет макет для всего экрана. Кроме того, вы также должны создать новый файл макета шаблона для каждого пункта в вашем элементе `ListView`.

Обновление основного макета.

Как и раньше, откройте редактор ресурсов в среде разработки Eclipse и удалите все существующие элементы из файла макета `menu.xml`. Затем выполните следующие шаги, чтобы создать желаемый макет экрана на основе заранее подготовленного дизайн-макета.

1. Добавьте новый элемент-контейнер `LinearLayout` и присвойте его атрибуту `background` значение `@drawable/bkgrnd`. Все остальные элементы будут размещаться внутри данного элемента-контейнера `LinearLayout`.
2. Добавьте элемент-контейнер `RelativeLayout`. Присвойте его атрибуту `layout_width` значение `wrap_content`, а атрибуту `layout_height` — значение `wrap_content`.
3. Добавьте элемент `ImageView` внутрь элемента-контейнера `RelativeLayout`. Атрибутам `layout_alignParentLeft` и `layout_alignParentTop` этого элемента `ImageView` присвойте значение `true`. Атрибуту `src` присвойте значение `@drawable/quizicon`.
4. Добавьте элемент `TextView`, который будет использоваться для отображения заголовка экрана, внутри элемента-контейнера `RelativeLayout`. Атрибутам `text`, `textSize` и `textColor` этого элемента `TextView` присвойте названия соответствующих ресурсов, которые вы создали чуть ранее. Затем присвойте атрибутам `layout_centerHorizontal` и `layout_alignParentTop` значение `true`. Чтобы «оживить» текст, отображаемый при помощи элемента `TextView`, вы можете использовать атрибуты для создания эффекта тени, включая атрибуты `shadowColor`, `shadowDx`, `shadowDy` и `shadowRadius`.
5. Завершите работу с элементом-контейнером `RelativeLayout`, добавив внутрь него еще один элемент `ImageView`. Атрибутам `layout_alignParentRight` и `layout_alignParentTop` этого элемента `ImageView` присвойте значение `true`. Атрибуту `src` присвойте значение `@drawable/quizicon`.
6. Теперь мы должны добавить второй элемент-контейнер `RelativeLayout`, который будет использоваться для размещения элементов `ListView` и `ImageView`. Начните с добавления нового элемента-контейнера `RelativeLayout` за пределами элемента-контейнера `RelativeLayout`, с которым мы работали до недавнего времени, но при этом новый элемент-контейнер `RelativeLayout` должен размещаться внутри существующего элемента контейнера `LinearLayout`.
7. Теперь добавьте последний элемент `ImageView` внутрь только что добавленного элемента-контейнера `RelativeLayout`. Присвойте его атрибуту `src` значение `@drawable/half`, а атрибутам `layout_width` и `layout_height` — значение `fill_parent`, чтобы гарантировать, что этот элемент займет все свободное пространство в нижней части экрана. Также присвойте его атрибуту `layout_alignParentBottom` значение `true`, а атрибуту `scaleType` — значение `fitEnd`, чтобы после масштабирования этот элемент управления по-прежнему оставался внизу экрана.
8. Теперь добавьте элемент `ListView` с именем `ListView_Menu` непосредственно под элементом `ImageView`. Присвойте его атрибуту `layout_width` значение `fill_parent`, а атрибуту `layout_height` — значение `wrap_content`. Также присвойте атрибуту `layout_alignTop` значение `true`. Этот атрибут позволяет сделать так, чтобы элемент отображался над изображением. После этого сохраните файл макета `menu.xml`.

Редактор ресурсов среды разработки Eclipse не отображает элементы ListView в режиме дизайна. Чтобы увидеть элемент ListView, используйте эмулятор Android. В данном случае редактор ресурсов не отражает реальный вид приложения.

Добавление макета шаблона элемента ListView. Теперь вам нужно создать новый файл макета `/res/layout/menu_item.xml`, который будет служить шаблоном для пунктов вашего элемента ListView. В данном случае файл макета `menu_item.xml` будет содержать элемент `TextView`.

Для элемента `TextView` настраиваются все типовые атрибуты, за исключением одного атрибута — атрибута, который представляет текстовое значение этого элемента. Атрибут `text` будет автоматически заполняться элементом ListView. Пока вы можете настроить значения атрибутов `textColor` и `textsize` элемента `TextView`, используя ресурсы цветов и размеров, созданных ранее.

Содержимое файла `menu_item.xml` выглядит следующим образом:

```
<TextView
xmlns:
android="http://schemas.android.com/apk/res/android"
android:layout_
width="fill_parent"
android:
textSize="@dimen/menu_item_size"
android:
text="test string"
android:layout_
gravity="center_horizontal"
android:layout_
height="wrap_content"
android:
shadowRadius="5"
android:
gravity="center"
android:
textColor="@color/menu_color"
android:
shadowColor="@color/menu_glow"
android:
shadowDy="3"
android:
shadowDx="3"
/>
```

Чтобы увидеть, как выглядят атрибуты элемента `TextView` в редакторе ресурсов среды разработки Eclipse, можно присвоить атрибуту `text` какую-либо тестовую строку. В дальнейшем эта строка будет перезаписываться элементом ListView программным путем. Теперь сохраните файл макета `menu_item.xml`.

Работа с элементом LISTVIEW.

Теперь вы должны открыть файл `QuizMenuActivity.java`. Вам нужно доработать элемент ListView. Во-первых, вам потребуется заполнить элемент ListView содержимым, а затем вам потребуется прослушивать события нажатий для отдельных пунктов элемента ListView и переправлять пользователя на соответствующий экран.

Заполнение элемента ListView. Элементу ListView требуется контент. Элементы ListView могут быть заполнены содержимым из множества источников данных, включая массивы и базы данных, путем использования адаптеров данных. В данном случае у вас есть список, состоящий из четырех элементов, поэтому простой массив значений типа `String` подходит в качестве источника данных для вашего элемента ListView.

Сначала вы должны получить экземпляр элемента ListView сразу после вызова метода `setContentView()` в методе `onCreate()` вашей деятельности. Чтобы заполнить элемент

ListView, вы должны получить экземпляр этого элемента, используя метод `findViewById()`, как показано в следующем коде:

```
ListView menuList = (ListView) findViewById(R.id.  
    ListView_  
    Menu);
```

Далее вам нужно определить значения типа `string` — они будут использованы для создания элементов `TextView`, которые будут добавлены в элемент `ListView`. В данном случае вы загрузите четыре строковых ресурса, представляющих пункты меню:

```
String[] items = { getResources().getString(R.string.  
    menu_item_  
    play),  
    getResources().getString(R.string.  
    menu_item_  
    scores),  
    getResources().getString(R.string.  
    menu_item_  
    settings),  
    getResources().getString(R.string.  
    menu_item_  
    help) };
```

Теперь, когда вы получили экземпляр элемента `ListView` и определили данные, которые должны быть помещены в этот элемент, вам потребуется адаптер данных, чтобы сопоставить данные с созданным макетом шаблона (`menu_item.xml`). Выбор адаптера зависит от типа используемых данных. В данном случае вы будете использовать класс `ArrayAdapter`:

```
ArrayAdapter<String> adapt = new ArrayAdapter<String>(this,  
    R.layout.  
    menu_  
    item, items);
```

Теперь вы должны сделать так, чтобы элемент `ListView` использовал этот адаптер:

```
menuList.setAdapter(adapt);
```

После этого вы можете сохранить файл `QuizMenuActivity.java` и запустить приложение «Been There, Done That!» на эмуляторе Android. После экрана-заставки появится экран с основным меню.

Если вы устали просматривать заставку при каждом запуске приложения, просто измените файл `AndroidManifest.xml`, чтобы по умолчанию пускалась деятельность `QuizMenuActivity`, пока вы не завершите отладку.

Как видите, экран с основным меню начинает приобретать форму. Тем не менее, нажатие по пунктам меню пока не приводят к желаемым результатам.

Прослушивание событий элемента `ListView`. Вы должны прослушивать и реагировать на определенные события, генерируемые элементом `ListView`. И хотя этот элемент может генерировать несколько различных событий, нас больше всего интересует событие, которое возникает в тот момент, когда пользователь нажимает на один из пунктов элемента `ListView`.

Чтобы прослушивать события, возникающие при нажатии по одному из пунктов элемента `ListView`, вам нужно использовать метод `setOnClickListener()` элемента `ListView`. В частности, необходимо реализовать метод `onItemClick()`, метод класса `AdapterView.OnItemClickListener`, как показано ниже:

```
menuList.setOnItemClickListener(new AdapterView.OnItemClickListener()  
{  
    public void onItemClick(AdapterView<?> parent, View itemClicked,  
        int position, long id) {  
        TextView textView = (TextView) itemClicked;  
        String strText = textView.getText().toString();  
        if (strText.equalsIgnoreCase(getResources().getString(  
            R.string.  
            menu_item_  
            play)) {
```

```

play))) {
// Launch the Game Activity
startActivity(new Intent(QuizMenuActivity.this,
QuizGameActivity.class));
} else if (strText.equalsIgnoreCase(getResources().getString(
R.string.
menu_item_
help))) {
// Launch the Help Activity
startActivity(new Intent(QuizMenuActivity.this,
QuizHelpActivity.class));
} else if (strText.equalsIgnoreCase(getResources().getString(
R.string.
menu_item_
settings))) {
// Launch the Settings Activity
startActivity(new Intent(QuizMenuActivity.this,
QuizSettingsActivity.class));
} else if (strText.equalsIgnoreCase(getResources().getString(
R.string.
menu_item_
scores))) {
// Launch the Scores Activity
startActivity(new Intent(QuizMenuActivity.this,
QuizScoresActivity.class));
}
}
});

```

В метод `onItemClick()` передается вся информация, которая необходима для определения того, на какой пункт нажал пользователь. В данном случае один из наиболее простых способов — привести выбранный пользователем объект представления к элементу `TextView` (поскольку известно, что все пункты являются элементами `TextView`, хотя вы можете дополнительно проверить это, используя оператор `instanceof`) и просто извлечь соответствующий элемент `Text` и сопоставить его с соответствующим экраном. Другой способ определить, на каком пункте нажал пользователь, — проверить атрибут `id` элемента `View`.

Теперь, реализовав метод `OnItemClickListener()` и перезапустив приложение на эмуляторе, вы можете использовать основное меню для перехода между экранами в приложении «Been There, Done That!».

Настройка внешнего вида элемента `ListView`. Теперь вы готовы приступить к настройке достаточно безвкусного (учитывая тот факт, что мы создаем меню для игры) элемента `ListView`, указав собственное изображение-разделитель и изображение-селектор.

`ListView` состоит из нескольких частей — верхней части, списка элементов и нижней части. По умолчанию элемент `ListView` отображается без верхней и нижней частей.

Если у вас есть экран с единственным элементом `ListView`, подумайте над возможностью использования класса `ListActivity`, который упрощает управление элементом `ListView`.

Добавление пользовательского разделителя.

Разделитель элемента `ListView` отображается между отдельными пунктами данного элемента. В качестве значения атрибута `divider` можно указать либо ресурс цвета, либо графический ресурс. Если указан ресурс цвета, между элементами в списке будет отображаться горизонтальная линия (толщину этой линии можно настраивать). Если указан графический ресурс, между элементами и списке будет отображаться соответствующее изображение. По умолчанию перед первым элементом списка и после последнего разделитель не отображается.

Добавление пользовательского селектора.

Селектор элемента `ListView` обозначает, какой элемент списка выбран в настоящий момент. Селектор элемента `ListView` устанавливается при помощи атрибута `listselector`. По умолчанию в качестве селектора элемента `ListView` используется полоса светло-оранжевого цвета.

Выполните самостоятельно.

Чтобы установить разделитель для элемента `ListView`, просто откройте файл макета `menu.xml` и присвойте атрибуту `divider` элемента `ListView` значение `@drawable/divider`, представляющее графический ресурс (волнообразная линия желтого цвета), который вы добавили ранее.

Теперь установите пользовательский селектор для элемента `ListView`. Для этого просто откройте файл макета `menu.xml` и присвойте атрибуту `listSelector` элемента `ListView` значение `@drawable/textured`, представляющее графический ресурс (текстурированный ореол оранжевого цвета), который вы добавили ранее.

Работа с другими типами меню.

Платформа Android предоставляет два других полезных типа меню:

- **Контекстное меню.** Отображается на экране, когда пользователь совершает продолжительное нажатие любого объекта типа `View`. Этот тип меню зачастую используется вместе с элементами `ListView`, содержащими похожие элементы, например названия музыкальных композиций в списке воспроизведения. Пользователь может сделать продолжительное нажатие по названию композиции, чтобы получить доступ к контекстному меню с такими командами, как, например, `Play` (Воспроизвести), `Delete` (Удалить) и `Add to Playlist` (Добавить в список воспроизведения), для этой конкретной композиции.
- **Меню опций.** Отображается на экране всякий раз, когда пользователь нажимает кнопку `Menu` на мобильном телефоне. Этот тип меню зачастую используется для предоставления пользователю возможности настройки параметров приложения и выполнения похожих операций. Поскольку в этом случае мы работали над экраном навигации по приложению, давайте посмотрим, на каких экранах можно было бы использовать перечисленные типы меню в приложении «Been There, Done That!». Текущая архитектура приложения позволяет легко интегрировать меню опций в игровой экран, позволяя пользователю приостановить процесс игры, быстро получить доступ к экранам с настройками приложения или помощи и затем вернуться обратно к игровому экрану.

Добавление меню опций на игровой экран. Чтобы добавить меню опций на игровой экран, вам потребуется добавить специальный тип ресурса, называемый ресурсом меню. После этого вы можете изменить класс `QuizGameActivity`, чтобы активизировать меню опций и добавить обработку выбранных пунктов меню.

Добавление ресурсов меню.

Для вашего меню опций необходимо создать ресурс с определением меню в формате XML и сохранить его в каталоге ресурсов `/res/menu` в файле с именем `gameoptions.xml`.

Ресурс меню содержит тег `<menu>`, за которым следует несколько дочерних элементов `<item>`. Каждый элемент `<item>` представляет отдельный пункт меню и имеет ряд атрибутов. Ниже перечислены некоторые наиболее часто используемые атрибуты.

- `id` — этот атрибут позволяет легко идентифицировать отдельные пункты меню.
- `title` — этот атрибут представляет строку, которая отображается для соответствующего пункта в меню опций.
- `icon` — этот атрибут используется для указания графического ресурса, представляющего значок для соответствующего пункта меню.

Ваше меню опций будет содержать всего два пункта; `Settings` (Настройки) и `Help` (Помощь). Таким образом, в содержимом файла вашего ресурса меню `gameoptions.xml` нет ничего сложного:


```

<menu
xmlns:
android="http://schemas.android.com/apk/res/
android">
<item
android:
id="@+id/settings_menu_item"
android:
title="@string/menu_item_settings"
android:
icon="@android:drawable/ic_menu_
preferences"></item>
<item
android:
id="@+id/help_menu_item"
android:
title="@string/menu_item_help"
android:
icon="@android:drawable/ic_menu_
help"></item>
</menu>

```

Для атрибута title каждого пункта меню опций вы указали те же строковые ресурсы, которые используются на экране с основным меню. Обратите внимание, что вместо добавления новых графических ресурсов для значков пунктов меню опций используются встроенные графические ресурсы из инструментария Android SDK, чтобы обеспечить согласованный внешний вид между различными приложениями.

Вы можете использовать встроенные графические ресурсы, доступные в классе android.R.drawable, точно так же, как вы используете ресурсы, включенные в пакет вашего приложения. Если вы хотите увидеть, как выглядит каждый из этих встроенных ресурсов, обратитесь к содержимому инструментария Android SDK, установленного на вашем компьютере. В частности, найдите каталог /platforms, выберите соответствующую целевую платформу и откройте каталог /data/res/drawable.

Добавление меню опций деятельности.

Чтобы меню опций отображалось в тот момент, когда пользователь нажмет кнопку Menu, находясь на игровом экране, вы должны реализовать метод onCreateOptionsMenu() в классе QuizGameActivity. В частности, вы должны создать (загрузить) меню опции из ресурса меню и установить соответствующий объект типа Intent для каждого меню. Вот пример реализации метода onCreateOptionsMenu() для каждого класса QuizGameActivity:

```

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    super.onCreateOptionsMenu(menu);
    getMenuInflater().inflate(R.menu.gameoptions, menu);
    menu.findItem(R.id.
    help_menu_
    item).setIntent(
    new Intent(this, QuizHelpActivity.class));
    menu.findItem(R.id.
    settings_menu_
    item).setIntent(
    new Intent(this, QuizSettingsActivity.class));
    return true;
}

```

Обработка события выбора пункта меню.

Чтобы обрабатывать ситуацию, когда пользователь открывает меню опций и выбирает некоторый пункт меню, необходимо реализовать метод onOptionsItemSelected() нужной деятельности. Например, вы можете запустить соответствующую деятельность, получив интент из пункта меню, как показано ниже:

```

@Override

```

```

public boolean onOptionsItemSelected(MenuItem item) {
    super.onOptionsItemSelected(item);
    startActivity(item.getIntent());
    return true;
}

```

Представленная реализация метода `onOptionsItemSelected()` работает как положено. С технической точки зрения, если ваше меню будет использоваться исключительно для запуска определенного интента, указанного посредством вызова метода `setIntent()`, реализовывать данный метод нет никакой необходимости. Тем не менее, если нужно добавить любую другую функциональность для каждого элемента `MenuItem`, вам придется реализовать данный метод.

Итак, все готово: вы создали меню опций на игровом экране. Если вы сохраните класс и запустите приложение еще раз, вы увидите, что можно перейти на игровой экран, нажать кнопку `Menu` и использовать полнофункциональное меню опций.

Итоги. Вы проделали большую работу. Экран с основным меню теперь полностью функционален, и вы освоили ключевые навыки разработки Android-приложений, включая использование новых элементов-контейнеров, например `RelativeLayout`, а также использование элемента с широкими возможностями `ListView`. Вы также познакомились с другими типами навигации, доступными в операционной системе Android, и реализовали меню опций на игровом экране.

7.4 Разработка экрана с инструкциями и экрана с результатами

В этом разделе вы разработаете еще два экрана приложения «Been There, Done That!»: экран с инструкциями и экран с результатами. Сначала вы займетесь разработкой экрана с инструкциями, взяв за основу элемент `TextView` и текст, загружаемый из текстового файла, что позволит вам познакомиться с некоторыми классами для работы с файлами, доступными в инструментарии Android SDK. После этого вы подготовите дизайн и реализуете экран с результатами. Поскольку к экрану с результатами предъявляются более сложные требования, он идеальный кандидат для использования элемента `TabHost`, предоставляющего набор вкладок. Наконец, вы протестируете экран, осуществив разбор тестовых XML-данных с результатами и отобразив соответствующую информацию на каждой вкладке [2].

Подготовка дизайна экрана с инструкциями.

Требования, предъявляемые к экрану с инструкциями, достаточно просты: этот экран должен отображать большой объем текста и иметь возможность прокрутки содержимого.

Вы хотите сделать так, чтобы экраны приложения обладали некоторым сходством. Таким образом, на экран с инструкциями можно добавить черты экрана меню навигации, например заголовок. Чтобы воплотить эскиз дизайна в соответствующем дизайн-макете, вам потребуется обновить файл макета `/res/layout/help.xml` и класс `QuizHelpActivity`.

Вы можете использовать такой же заголовок экрана, который применяется на экране с основным меню (реализованный при помощи элемента-контейнера `RelativeLayout`) и за которым будет следовать элемент `TextView` с возможностью прокрутки содержимого.

Реализация макета экрана с инструкциями.

Реализацию экрана с инструкциями вы начнете с добавлением новых ресурсов в проект. Затем вы обновите файл макета `help.xml`, чтобы отразить подготовленный дизайн-макет экрана с инструкциями.

Добавление новых ресурсов в проект. Помимо всех новых строковых ресурсов, ресурсов цветов и размеров, которые будут использованы в макете экрана с инструкциями, вам также потребуется добавить новый тип ресурса — ресурс файла с данными. В данном примере вы добавите текстовый файл `/res/raw/quizhelp.txt`, который

содержит несколько абзацев текста с инструкциями для отображения в основном элементе TextView экрана с инструкциями.

Вы также можете добавлять большие блоки текста в качестве строковых ресурсов. Такой подход может оказаться полезным при локализации приложения. Использование строковых ресурсов также позволяет вам воспользоваться встроенной поддержкой тегов в стиле языка HTML. В данном примере мы остановили свой выбор на текстовом файле, чтобы продемонстрировать использование ресурсов файлов с данными.

Обновление макета экрана с инструкциями. Файл макета help.xml определяет пользовательский интерфейс экрана с инструкциями. Откройте редактор ресурсов среды разработки Eclipse и удалите все существующие элементы пользовательского интерфейса из макета. Далее выполните следующие шаги, чтобы создать желаемый макет на основе ранее подготовленного дизайн-макета экрана.

Добавьте новый элемент-контейнер LinearLayout и присвойте его атрибуту background значение @drawable/bkgrnd. Все остальные элементы будут добавляться внутри этого элемента-контейнера LinearLayout.

Добавьте тот же заголовок, который вы использовали в файле макета menu.xml. Он должен включать элемент-контейнер RelativeLayout, содержащий два элемента ImageView и один элемент TextView. Атрибуту text элемента TextView присвойте строковый ресурс с именем @string/help, чтобы отобразить соответствующий заголовок экрана.

Добавьте новый элемент TextView с именем TextView_HelpText внутри элемента контейнера LinearLayout, но за пределами элемента-контейнера RelativeLayout.

Присвойте его атрибутам layout_width и layout_height значение fill_parent.

Вы можете автоматически связать номера телефона, веб-сайты, адреса электронной почты и почтовые адреса, отображаемые в элементе TextView, с приложениями Телефон, Браузер, Email и Карта операционной системы Android, присвоив атрибуту linksClickable элемента TextView значение true, а атрибуту autoLink – значение all.

Внеся необходимые изменения, сохраните файл макета help.xml.

Чтобы установить полужирное или курсивное начертание для текста, отражаемого в элементе TextView, используйте атрибут textStyle.

Работа с файлами.

Теперь, когда файл макета help.xml готов, вам нужно обновить класс QuizHelpActivity так, чтобы он считывал содержимое файла quizhelp.txt и сохранял полученный текст в элементе TextView с именем TextView_HelpText.

Добавление ресурсов файлов с данными. Ресурсы файлов с данными, как, например, текстовый файл quizhelp.txt, могут быть добавлены в проект путем помещения в каталог /raw, находящийся внутри каталога с ресурсами проекта. Вы можете либо создать новые файлы внутри этой папки, либо перетащить существующие файлы в файловом менеджере, либо использовать любой другой способ, который вы привыкли использовать для добавления файлов в проекты Android в среде разработки Eclipse.

Каждое Android-приложение имеет свой собственный закрытый каталог в файловой системе Android для хранения файлов приложения. Помимо широко известных классов File и Stream, вы можете обращаться к закрытым файлам и каталогам приложения при помощи следующих методов класса Context:

```
getFilesDir(), getDir(), openFileInput(), openFileOutput(),  
deleteFile()  
и getFileStreamPath().
```

Обращение к ресурсам файлов с данными

Платформа Android предоставляет множество классов, присущих языку Java, для выполнения операции ввода/вывода, включая классы для выполнения потоковых операций. Чтобы прочитать строковые данные из файла, используйте метод openRawResource(), как показано в следующем коде:

```
InputStream iFile = getResources().openRawResource(R.raw.quizhelp);
```

Обратите внимание, что, имея экземпляр класса `InputStream`, вы можете построчно или посимвольно прочесть данные из файла и получить результирующую строку. Сделать это можно несколькими способами. Создан соответствующую строку с текстом инструкций по игре, вы просто получаете ссылку на экземпляр элемента `TextView`, используя метод `findViewById()`, и присваиваете строку этому элементу, используя его метод `setText()`, как показано в следующем коде:

```
TextView helpText = (TextView) findViewById(R.id.  
    TextView_  
    HelpText);  
String strFile = inputStreamToString(iFile);  
helpText.setText(strFile);
```

В предыдущем коде используется вызов вспомогательного метода `inputStreamToString()`, реализованного в классе `QuizHelpActivity`, который доступен на диске, прилагаемом к данной книге. Этот метод просто считывает данные из экземпляра класса `InputStream` и возвращает объект типа `string`, представляющий содержимое экземпляра класса `InputStream`.

Теперь сохраните файл `QuizHelpActivity.java` и запустите приложение «Been There, Done That!» на эмуляторе Android. Когда заставка приложения исчезнет с экрана, выберите в основном меню пункт, соответствующий экрану с инструкциями.

Выполните самостоятельно.

Теперь, когда на экране с инструкциями отображается нужная информация, вы можете захотеть улучшить его внешний вид. Начните с изменения размера шрифта текста с инструкциями в элементе `TextView`. Затем попробуйте изменить другие атрибуты, например, отступы, гарнитуру, цвета текста и т.д.

Подготовка дизайна экрана с результатами.

Теперь, когда вы создали экран с инструкциями, можно приступить к работе над другим экраном с похожей функциональностью — над экраном с результатами. В соответствии с требованиями, предъявляемыми к данному экрану, пользователь должен видеть несколько различных результатов. Существует два типа результатов: лучшие результаты игры и результаты друзей пользователя. Результаты обоих типов должны отображаться на одном экране. Каждый экран будет содержать имя пользователя, его результат и общий рейтинг.

Существует несколько вариантов реализации экрана с результатами. Например, вы могли бы использовать элементы `TextView` или `ListView` для отображения информации о результатах. Тем не менее вы работаете с экраном небольшого размера, и вы не хотите перегружать пользователя слишком большим количеством информации. Поскольку у вас есть два различных набора данных для отображения, идеальным решением для данного экрана было бы использование двух вкладок.

Определение требований к макету экрана с результатами. Теперь, когда вы знаете, как должен выглядеть ваш экран с результатами игры, необходимо преобразовать эскиз дизайна в соответствующий дизайн-макет. В данном случае вы должны обновить файл макета `/res/layout/scores.xml`, который используется классом `QuizScoresActivity`. Как и раньше, вы можете воспользоваться преимуществами элемента-контейнера `RelativeLayout`, чтобы добавить уже знакомый вам заголовок в верхнюю часть экрана. В данном случае сразу после заголовка экрана будет расположен элемент `TabHost` с двумя вкладками, каждая из которых будет содержать элемент-контейнер `TableLayout` с результатами — одна вкладка будет представлять общие результаты игры, а другая — результаты друзей пользователя.

Добавление элемента `TabHost`. Чтобы добавить поддержку вкладок на экран с результатами игры, необходимо включить в макет элемент `TabHost`, содержащий дочерние вкладки, каждая из которых может использоваться для отображения некоего макета. Элемент `TabHost` довольно сложен в настройке. Чтобы правильно сконфигурировать его в XML-файле макета, нужно выполнить следующие действия:

- добавить элемент TabHost;
- внутрь элемента TabHost добавить элемент контейнер внутрь элемента-контейнера LinearLayout, добавить элемент TabWidget со специальным именем и элемент-контейнер FrameLayout;
- определить содержимое каждой вкладки внутри элемента-контейнера FrameLayout.

Реализация макета экрана с результатами игры.

Реализацию экрана с результатами игры вы начнете с добавления новых ресурсов в проект. Затем вы обновите файл макета scores.xml, чтобы отразить ранее подготовленный дизайн-макет экрана с результатами.

Добавление новых ресурсов в проект. Помимо новых строковых ресурсов, ресурсов пиктограмм и размеров, которые были использованы в макете экрана с результатами, вам также потребуется добавить новый тип ресурса — ресурс файла с XML-данными. Со временем приложение «Been There, Done That!» начнет получать данные о результатах игры с удаленного сервера, однако пока вы можете разработать экран и использовать тестовые данные, представляющие результаты игры.

Эти тестовые данные будут представлены в XML-формате, поэтому вы можете использовать структуру, которая в дальнейшем будет применяться для представления реальных результатов.

В этом примере вы добавите в каталог с ресурсами /res/xml два XML-файла — allscores.xml и friendscores.xml, которые представляют тестовые данные с результатами игры.

```
<?xml
version="1.0" encoding="utf-
8"?>
<!-- Это фрагмент текстовых XML-данных с результатами игры -->
<scores>
<score
username="LED"
score="12345"
rank="1"
/>
<score
username="SAC"
score="12344"
rank="2"
/>
<score
username="NAD"
score="12339"
rank="3"
/>
</scores>
```

Для представления результатов игры используется очень простая схема. Единственный элемент <scores> имеет несколько дочерних элементов <score>. Каждый элемент <score> имеет три атрибута: username, score и rank. В нашем случае вы можете предположить, что данные результатов будут заранее отсортированы и будут содержать максимум 20 (или любое другое число) записей.

Обновление макета экрана с результатами. Пользовательский интерфейс экрана с результатами определяется в файле макета scores.xml. Чтобы обновить этот макет в соответствии с разработанным дизайн-макетом, выполните следующие шаги.

В режиме дизайна редактор ресурсов среды разработки Eclipse неправильно отображает элементы TabHost. Для создания макетов, использующих данные элементы, вы должны переключиться в режим XML-разметки. Для просмотра вкладок нужно

использовать эмулятор Android или реальное устройство, работающее над управлением операционной среды Android.

1. Удалите все существующие элементы пользовательского интерфейса из макета, как вы делали это для других макетов в этой книге.
2. Добавьте новый элемент-контейнер `LinearLayout`, присвоив его атрибуту `android:background` значение `@drawable/bkgrnd`. Все остальные элементы будут добавляться внутрь данного элемента-контейнера `LinearLayout`.
3. Добавьте тот же заголовок, который вы использовали в других макетах. Он должен включать элемент-контейнер `RelativeLayout` с двумя элементами `ImageView` и одним элементом `TextView`. Атрибуту `text` элемента `TextView` присвойте строковый ресурс `@string/scores`, чтобы отобразить соответствующий заголовок экрана.
4. Внутри элемента-контейнера `LinearLayout`, но вне элемента-контейнера `RelativeLayout`, добавьте элемент `TabHost` с именем `TabHost1`. Присвойте его атрибутам `layout_width` и `layout_height` значение `fill_parent`.
5. Внутри элемента `TabHost` добавьте другой элемент-контейнер `LinearLayout`, присвоив его атрибуту `orientation` значение `vertical`. Атрибутам `layout_width` и `layout_height` этого элемента-контейнера присвойте значение `fill_parent`.
6. Внутри элемента-контейнера `LinearLayout`, добавленного на предыдущем шаге, добавьте элемент `TabWidget`. Присвойте его атрибуту `id` значение `@android:id/tabs`.
7. Внутри элемента-контейнера `LinearLayout`, добавленного на шаге 5, на том же уровне вложенности, на котором находится элемент `TabWidget`, добавьте элемент-контейнер `FrameLayout`. Присвойте атрибуту `id` этого элемента контейнера `FrameLayout` значение `@android:id/tabcontent`, а атрибутам `layout_width` и `layout_height` — значение `fill_parent`.
8. Определите содержимое ваших вкладок. Внутри элемента-контейнера `FrameLayout` добавьте два элемента-контейнера `TableLayout` — по одному для каждой вкладки. Вы будете использовать эти элементы-контейнеры `TableLayout` для отображения результатов игры. Назовите первый элемент-контейнер `TableLayout_FriendScores`. Присвойте атрибутам `layout_width` и `layout_height` этих элементов-контейнеров значение `fill_parent`. Присвойте атрибуту `stretchColumns` значение `*`, чтобы размеры столбцов изменялось в зависимости от содержимого.

При создании подобного представления с вкладками вы должны использовать идентификатор, который был указан выше, — `@android:id/tabcontent`; в противном случае, в процессе выполнения приложения возникнут исключения.

Это значение ссылается на специальный ресурс в пакете платформы Android.

Это не то же самое, что значение `@+id/tabcontent`. В последнем случае будет создан новый идентификатор для объекта макета в пакете вашего приложения.

Вы можете обеспечить прокрутку содержимого для многих элементов, поместив их внутрь элемента `ScrollView`. Например, чтобы обеспечить вертикальную прокрутку содержимого для элемента-контейнера `TableLayout`, добавьте этот элемент-контейнер внутрь элемента `ScrollView` и присвойте атрибуту `scrollbars` элемента `ScrollView` значение `vertical`. Также необходимо присвоить значения атрибутам `layout_width` и `layout_height` элемента `ScrollView`.

Часть файла макета экрана с результатами, относящаяся к элементу `TabHost` (с возможностью прокрутки содержимого вкладок, реализованных при помощи элементов-контейнеров `TableLayout`), должна выглядеть следующим образом:

```
<TabHost
android:
id="@+id/TabHost1"
android:layout_
```

```

width="fill_parent"
android:layout_
height="fill_
parent">
<LinearLayout
android:
orientation="vertical"
android:layout_
width="fill_parent"
android:layout_
height="fill_
parent">
<TabWidget
android:
id="@android:id/tabs"
android:layout_
width="fill_parent"
android:layout_
height="wrap_content"
/>
<FrameLayout
android:
id="@android:id/tabcontent"
android:layout_
width="fill_parent"
android:layout_
height="fill_
parent">
<ScrollView
android:
id="@+id/ScrollViewAllScores"
android:layout_
width="fill_parent"
android:layout_
height="fill_parent"
android:
scrollbars="
vertical">
<TableLayout
android:
id="@+id/TableLayout_AllScores"
android:layout_
width="fill_parent"
android:layout_
height="fill_parent"
android:stretchColumns="*">
</TableLayout>
</ScrollView>
<ScrollView
android:
id="@+id/ScrollViewFriendScores"
android:layout_
width="fill_parent"
android:layout_
height="fill_parent"
android:
scrollbars="
vertical">
<TableLayout
android:
id="@+id/TableLayout_FriendScores"
android:layout_
width="fill_parent"

```

```

android:layout_
height="fill_parent"
android:stretchColumns="*"></TableLayout>
</ScrollView>
</FrameLayout>
</LinearLayout>
</TabHost>

```

Сохраните файл макета scores.xml.

Если сейчас вы переключитесь в режим визуального представления макета, среда разработки Eclipse может сгенерировать ошибку «NullPointerException: null».

Редактор ресурсов не поддерживает элементы TabHost платформы Android 2.1. При этом макет будет без проблем отображаться на эмуляторе и на реальных устройствах.

Разработка дизайна экрана с вкладками.

Теперь вы должны полностью сосредоточиться на файле QuizScoresActivity.java и добавить код для элементе, используемых элементом TabHost. Сначала вы должны проинициализировать элемент TabHost и затем добавить в него две вкладки, одна из которых — вкладка All Scores (Общие результаты) — будет использоваться в качестве вкладки по умолчанию. Наконец, вы воспользуетесь тестовыми данными с результатами игры для заполнения элемента-контейнера TableLayout в каждой вкладке.

Настройка элемента TabHost. Для правильного функционирования элемент TabHost сначала должен быть проинициализирован. Таким образом, вы должны получить доступ к экземпляру этого элемента, используя метод findViewById(). Затем вы должны вызвать метод setup() элемента TabHost, который выполняет инициализацию элемента TabHost и «склеивает» вместе элементы TabWidget и FrameLayout со специальными именами, чтобы образовать набор вкладок, как показано в следующем коде:

```

TabHost host = (TabHost) findViewById(R.id.TabHost1);
host.setup();

```

Добавление вкладок в элемент TabHost. Теперь, когда вы получили доступ к вашему элементу TabHost и проинициализировали его, вам нужно настроить каждую вкладку, используя метод addTab().

Этот метод принимает параметр типа TabSpec, который определяет содержимое вкладки. Например, следующий код создает вкладку All Scores (Общие результаты):

```

TabSpec allScoresTab = host.newTabSpec("allTab");
allScoresTab.setIndicator(getResources().getString(R.string.
all_scores),
getResources().getDrawable(android.R.drawable.
star_on));
allScoresTab.setContent(R.id.ScrollViewAllScores);
host.addTab(allScoresTab);

```

Объект типа TabSpec с именем allScoresTab имеет метку для ссылок allTab. На экране название вкладки представляется элементом TextView и значком (звездочкой).

Наконец, содержимое вкладки определяется элементом с идентификатором ScrollViewAllScores, который содержит элемент-контейнер TableLayout с именем TableLayout_AllScores, как было указано в файле макета scores.xml.

Затем вы добавляете вторую вкладку с меткой для ссылок friendTab в элемент TabHost. Вторая вкладка реализована почти так же, как и первая вкладка, за исключением содержимого (вместо лучших результатов игры на этой вкладке отображаются только результаты игры друзей пользователя).

Назначение вкладки, используемой по умолчанию. Теперь вы должны указать, какая вкладка будет отображаться по умолчанию. Для этого вам нужно вызвать метод setCurrentTabById() и передать в этот метод метку соответствующей вкладки:

```

host.setCurrentTabByTag("allTab");

```

Сохраните файл QuizScoresActivity.java и запустите приложение на эмуляторе Android. Когда вы откроете экран с результатами игры, вы увидите две вкладки, однако у вас по-прежнему нет данных с результатами игры для отображения.

Если на экране приложения должен отображаться только элемент TabHost, рассмотрите возможность использования класса TabActivity, который упрощает работу с элементом TabHost.

Работа с XML-данными.

Платформа Android имеет ряд механизмов для работы с XML-данными, включая поддержку следующих технологий:

- SAX (Simple API for XML);
- XML Pull Parser;
- Limited DOM Level 2 core support.

Выбор используемой технологии для работы с XML-данными зависит от конкретного проекта. В данном примере вы просто хотите прочитать содержимое простого XML-файла и извлечь тестовые данные с результатами игры.

Получение XML-ресурсов. Сначала вы должны получить доступ к тестовым XML-данным, которые вы сохранили в ресурсах проекта. В частности, вам нужно разобрать файл /res/xml/allscores.xml.

Вы можете проинициализировать экземпляр класса XmlResourceParser, используя метод getXML(), как показано в следующем коде:

```
XmlResourceParser mockAllScores =  
getResources().getXml(R.xml.allscores);
```

Разбор XML-файлов при помощи класса XmlResourceParser. Тестовые файлы с результатами игры используют очень простую схему, содержащую всего два тега: <scores> и <score>. Вам нужно найти все теги <score> и получить значения их атрибутов username, rank и score. Поскольку подразумевается, что вы будете работать с небольшим количеством данных, процедуру разбора XML-данных можно реализовать при помощи простого цикла while(), используя для перехода между событиями метод next(), как показано в следующем коде:

```
int eventType = -1;  
boolean bFoundScores = false;  
// Find Score records from XML  
while (eventType != XmlResourceParser.  
END_ DOCUMENT) {  
    if (eventType == XmlResourceParser.  
START_ TAG) {  
        // Get the name of the tag (eg scores or score)  
        String strName = scores.getName();  
        if (strName.equals("score")) {  
            bFoundScores = true;  
            String scoreValue = scores.getAttributeValue(null,  
"score");  
            String scoreRank = scores.getAttributeValue(null, "rank");  
            String scoreUserName =  
scores.getAttributeValue(null, "username");  
            insertScoreRow(scoreTable, scoreValue, scoreRank,  
scoreUserName);  
        }  
    }  
    eventType = scores.next();  
}
```

В цикле вы ожидаете появления события START_TAG. Когда имя тега совпадет с именем тега <score>, вы будете знать, что обнаружена часть данных с результатами игры. После этого вы можете получить обнаруженные данные, используя метод getAttributeValue(). Для каждого обнаруженного результата вы добавляете новый элемент TableRow в соответствующий элемент-контейнер TableLayout (на соответствующей вкладке).

Внесение завершающих штрихов на экран с результатами. Написав код для разбора двух тестовых XML-файлов и заполнения двух элементов-контейнеров `TableLayout` внутри элемента `TabHost`, внести несколько незначительных изменений в класс `QuizScoresActivity`. Вы должны добавить заголовок в виде элемента `TableRow` в каждый элемент-контейнер `TableLayout`, придать заголовкам столбцов привлекательный вид, а также обработать ситуацию, когда данные о результатах игры недоступны.

Внеся эти завершающие штрихи, сохраните исходный код класса и запустите приложение на эмуляторе или на реальном устройстве. Когда откроете экран с результатами, то увидите, чтобы обе вкладки содержат данные.

Итоги. В этом разделе вы добавили два новых экрана в опрос-викторину «Beep There, Done That!». В процессе реализации экрана с инструкциями вы узнали, как можно отображать большие объемы данных при помощи элемента `TextView` с возможностью прокрутки содержимого. Вы также узнали, как обращаться к ресурсам файлов и изменять настройки макета программным путем. В процессе реализации экрана с результатами вы познакомились с элементом `TabHost`, а также узнали, как можно разобрать XML данные для отображения тестовых результатов.

8 Разработка многопоточного игрового приложения

8.1 Тестирование приложения

В игре Cannon Game требуется разрушить состоящую из семи частей мишень в течение 10 секунд, отведенных на игру (рис. 8.1). Игра включает четыре визуальных компонента — *пушка*, управляемая пользователем, *пушечное ядро*, *мишень* и *блок*, защищающий мишень. Чтобы нацелить пушку, коснитесь пальцем экрана. В результате пушка нацелится в точку, в которой вы коснулись экрана пальцем. После двойного касания пальцем экрана пушка выстрелит пушечным ядром. В конце игры приложение отобразит диалоговое окно AlertDialog, с помощью которого можно узнать, выиграли вы или проиграли, количество сделанных выстрелов и время игры (рис. 8.2) [1].

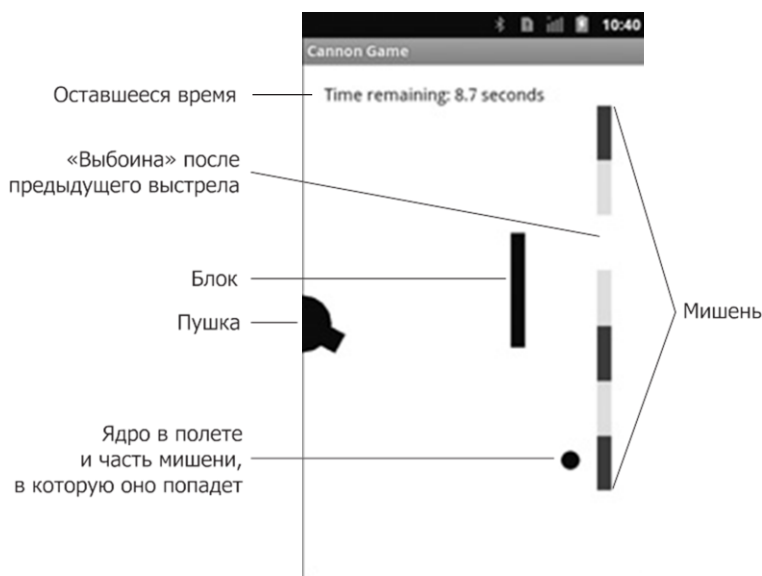


Рис. 8.1. Завершенное приложение Cannon Game

В начале игры каждому пользователю выделяется лимит времени, равный 10 секундам. При каждом попадании в секцию мишени к лимиту времени добавляются три секунды, а при каждом попадании в блок вычитаются две секунды. Игра считается выигранной, если были разрушены все секции мишени до истечения заданного лимита времени. Если значение таймера становится равным нулю до разрушения мишени, вы проиграете.

После выстрела из пушки приложение воспроизводит звук выстрела. Мишень состоит из семи секций. После попадания пушечного ядра в секцию мишени раздается звук разбивающегося стекла, а сама секция исчезает с экрана. Если ядро попадает в блок, раздается звук удара, а ядро отскакивает назад. При этом блок не разрушается. Мишень и блок перемещаются по вертикали с различной скоростью. Причем направление перемещения изменяется после касания верхней или нижней части экрана.

Откройте среду Eclipse и импортируйте проект Cannon Game app. Выполните следующие действия:

1. *Откройте диалоговое окно Import.* Для этого выполните команды File¢Import...(Файл¢Импорт...).

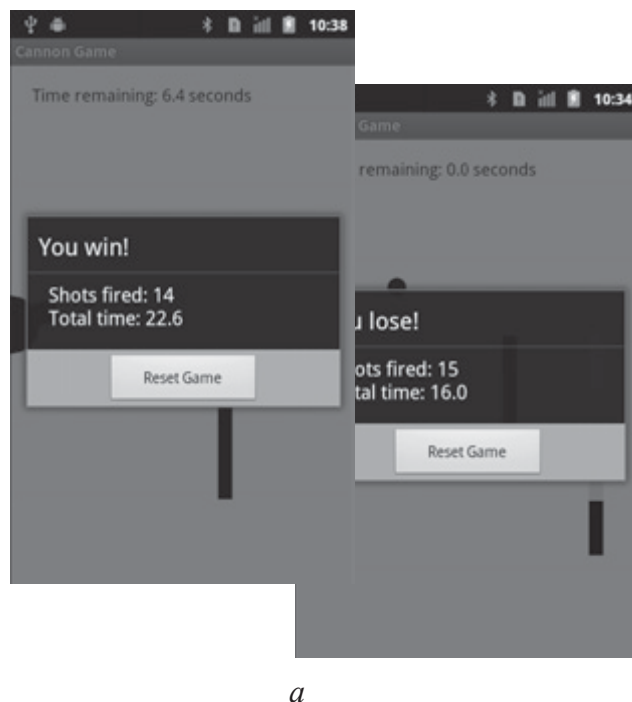


Рис. 8.2. С помощью диалогового окна AlertDialogs приложения Cannon Game можно узнать, выиграли вы или проиграли: *а* — после разрушения всех семи секций мишени соответствующее сообщение появляется в окне AlertDialog; *б* — если время истечет до разрушения всех секций мишени, в окне AlertDialog отобразится соответствующее сообщение

2. *Импортируйте проект приложения Canon Game.* В диалоговом окне Import раскройте узел General (Общий) и выберите параметр Existing Projects into Workspace (Существующие проекты в рабочую область). Затем щелкните на кнопке Next > (Далее >) для выполнения шага Import Projects (Импорт проектов). Выберите корневой каталог и щелкните на кнопке Browse... (Просмотр...). В диалоговом окне Browse For Folder (Просмотр папки) выберите папку CanonGame в папке примеров книги и щелкните на кнопке OK. Щелкните на кнопке Finish (Готово) для импорта проекта в среду Eclipse. Проект отобразится в окне Package Explorer, находящемся в левой части окна Eclipse.
3. *Запустите приложение Canon Game.* В среде Eclipse щелкните правой кнопкой мыши на проекте CanonGame, находящемся в окне Package Explorer, затем в появившемся меню выберите команды Run As Android Application (Выполнить как Приложение Android).

Перетащите палец в области экрана или коснитесь им экрана, чтобы нацелить пушку. Дважды коснитесь экрана, чтобы выстрелить из пушки. Вы сможете выстрелить пушечным ядром только в том случае, если на экране не отображается другое ядро. Если приложение выполняется на экране AVD, в качестве «пальца» будет использоваться мышь. Попробуйте уничтожить мишень как можно быстрее. Если показания таймера обнулятся, игра завершится.

8.2 Обзор применяемых технологий

В этом разделе вашему вниманию представлены несколько новых технологий, использованных при создании приложения Cannon Game. Эти технологии будут рассмотрены в том порядке, в котором они встречаются в главе.

Определение ресурсов форматированных строк в файле strings.xml.

В этом приложении определяются ресурсы String, используемые для

представления форматированных строк. Эти строки применяются при вызовах метода `getString` (или статического метода `format` класса `String`) класса `Resource`. Если форматные строки включают несколько спецификаторов формата, потребуется пронумеровать их (начиная с 1) для определения порядка подстановки соответствующих значений в форматированную строку. Фразы на различных языках могут представлять собой результат подстановки значений в различные места локализованных строковых ресурсов. В подобных случаях локализованные версии файла `strings.xml` используют оригинальные номера спецификаторов формата, которые помещаются в соответствующие места локализованных строк.

Добавление в макет пользовательского представления. Можно создать пользовательское представление путем расширения класса `View` или одного из его подклассов. Пример подобного представления — класс `CannonView` (раздел 7.5.3), который расширяет класс `SurfaceView` (будет рассмотрен в ближайшее время). Чтобы добавить пользовательский компонент в файл XML-разметки, следует полностью определить имя соответствующего ему класса в XML-элементе, представляющем компонент.

Использование папки ресурсов `raw`. Медиафайлы (например, звуки), используемые в приложении `Cannon Game`, находятся в папке ресурсов приложения `res/raw`. В эту папку помещаются звуковые файлы.

Методы «жизненного цикла» `onPause` и `onDestroy` класса `Activity`. Это приложение использует дополнительные методы «жизненного цикла» `Activity`. Метод `onPause` вызывается для текущего класса `Activity` в случае, если другая деятельность получает фокус. При этом текущая деятельность будет выполняться в фоновом режиме. С помощью метода `onPause` текущая игра приостанавливается. То есть если пользователь не взаимодействует с игрой, выполнение игры прекращается. Если класс `Activity` завершается, вызывается его метод `onDestroy`. Этот метод применяется для «освобождения» звуковых ресурсов приложения.

Переопределение метода `onTouchEvent` класса `Activity`. Как уже упоминалось ранее, пользователи взаимодействуют с приложением путем касания экрана устройства. Чтобы нацелить пушку на определенную точку экрана, *коснитесь (или один раз нажмите) пальцем*. Для выполнения обработки событий класса `Activity`, вызываемых касанием экрана пальцем, переопределите `onTouchEvent` класса `Activity`, а затем воспользуйтесь константами класса `MotionEvent` (пакет `android.view`) для определения типа происходящих событий и их последующей обработки.

Классы `GestureDetector` и `SimpleOnGestureListener`. Для обработки более сложных жестов, например двойных нажатий, используемых для выполнения стрельбы из пушки, можно воспользоваться классом `GestureDetector` (пакет `android.view`). С помощью этого класса распознаются действия пользователя, представленные наборами событий `MotionEvent`. С помощью класса `GestureDetector` приложение может реагировать на более сложные пользовательские жесты, такие как щелчки, двойные нажатия, длительные нажатия и прокрутки. Приложения могут «отвечать» на подобные события с помощью реализации методов интерфейсов `GestureDetector.OnGestureListener` и `GestureDetector.OnDoubleTapListener`. Класс `GestureDetector.SimpleOnGestureListener` представляет собой класс адаптера, который реализует все методы для двух вышеперечисленных интерфейсов. В результате появляется возможность расширения класса и переопределения требуемого метода(ов), относящегося к этим интерфейсам.

Добавление звука с помощью `SoundPool` и `AudioManager`. Звуковыми эффектами приложения можно управлять с помощью класса `SoundPool` (пакет `android.media`). Доступны загрузка, воспроизведение и выгрузка звуков. Для воспроизведения звуков используется один из нескольких аудиопотоков `Android`, включающих потоки для оповещений, тонов DTMF, музыки, уведомлений,

телефонных звонков, системных звуков и рингтонов. В документации Android рекомендуется использовать для воспроизведения звука в играх музыкальный аудиопоток. С помощью метода `setVolumeControlStream` класса `Activity` определяется возможность управления громкостью звука в игре кнопками регулировки громкости устройства, которые также используются для изменения громкости воспроизведения музыки. Метод получает константу из класса `AudioManager` (пакет `android.media`).

Покадровая анимация с помощью потоков, `SurfaceView` и `SurfaceHolder`.

Это приложение реализует выполнение анимации вручную путем обновления элементов игры, находящихся в отдельном потоке выполнения. Для этого используется подкласс класса `Thread` с методом `run`, который с помощью пользовательского класса `CannonView` обновляет позиции всех элементов игры, а затем обновляет сами элементы. Как правило, любые обновления пользовательского интерфейса пользователя должны выполняться с помощью потока выполнения GUI. Но в Android важно минимизировать объем работы, выполняемой в потоке GUI, чтобы убедиться в том, что GUI остается «отзывчивым» и не отображает диалоговые окна ANR (Application Not Responding, Приложение не отвечает).

Зачастую в играх задействована сложная логика, которая должна работать в отдельных потоках выполнения, иногда эти потоки требуется отображать на экране. Для подобных случаев Android поддерживает класс `SurfaceView` — подкласс класса `View`, в котором может быть «нарисован» произвольный поток. Классом `SurfaceView` можно манипулировать посредством объекта класса `SurfaceHolder`, позволяющего получить объект `Canvas`, на котором рисуется графика. Класс `SurfaceHolder` также поддерживает методы, обеспечивающие потоку исключительный доступ к объекту `Canvas` для рисования, поскольку рисовать в `SurfaceView` может лишь один поток одновременно. Каждый подкласс `SurfaceView` должен реализовывать интерфейс `SurfaceHolder.Callback`, включающий методы, которые вызываются при создании, изменять (размеры или ориентацию) либо уничтожать компонент `SurfaceView`.

Простое обнаружение конфликтов. С помощью класса `CannonView` выполняется простое обнаружение столкновений пушечного ядра с каким-либо краем компонента `CannonView`, с блоком или с секцией мишени.

ПРИМЕЧАНИЕ.

Многие фреймворки, предназначенные для разработки игр, предлагают более сложные механизмы обнаружения столкновений.

Рисование графики с помощью `Paint` и `Canvas`. Методы класса `Canvas` (пакет `android.graphics`) применяются для рисования текста, линий и окружностей. Метод `Canvas` рисует в области объекта `Bitmap` класса `View`. Каждый метод рисования класса `Canvas` использует объект класса `Paint` (пакет `android.graphics`) для определения характеристик рисования, включая цвет, толщину линии, размер шрифта и другие параметры. Эти характеристики реализованы с помощью метода `drawGameElements`. Чтобы получить дополнительные сведения о характеристиках, используемых в процессе рисования и указываемых с помощью объекта `Paint`, посетите веб-сайт developer.android.com/reference/android/graphics/ Paint.html.

8.3 Создание графического интерфейса пользователя приложения и файлов ресурсов

В этом разделе будут созданы файлы ресурсов приложения и файл разметки `main.xml`.

Начните создание нового проекта Android под названием `Cannon Game`. В диалоговом окне `New Android Project` (Новый проект Android) введите следующие значения, а затем нажмите кнопку `Finish` (Готово):

- `Build Target` (Операционная система): `Android 2.3.3`.
- `Application name` (Имя приложения): `Cannon Game`.

- Package name (Название пакета): com.deitel.cannongame.
- Create Activity (Создать деятельность): CannonGame.
- Min SDK Version (Минимальная версия SDK): 8.

Файл AndroidManifest.xml. В листинге 8.1 приведен XML-код из файла приложения AndroidManifest.xml. Атрибуту элемента деятельности android:screenOrientation было присвоено значение "portrait" (строка 9), в результате чего приложение всегда отображается в портретном режиме.

Листинг 8.1. Файл AndroidManifest.xml

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest
  xmlns:android=http://schemas.android.com/apk/res/android
3   package="com.deitel.cannongame" android:versionCode="1"
4   android:versionName="1.0">
5   <application android:icon="@drawable/icon"
6     android:label="@string/app_name"
7     android:debuggable="true">
8     <activity android:name=".CannonGame"
9       android:label="@string/app_name"
10      android:screenOrientation="portrait">
11     <intent-filter>
12       <action android:name="android.intent.action.MAIN" />
13       <category
14         android:name="android.intent.category.LAUNCHER" />
15     </intent-filter>
16   </activity>
17 </application>
18 </manifest>

```

Файл strings.xml. В файле strings.xml, относящемся к приложению, определяются форматные строки (листинг 7.2, строки 4–5 и 9–10). Как упоминалось в разделе 7.3, в форматных строках, включающих несколько спецификаторов формата, эти спецификаторы нумеруются (например, для задач локализации приложения). Запись 1\$ в спецификаторе %1\$.1f (строка 5) указывает на то, что первый аргумент после форматной строки должен заменять спецификатор формата %1\$d. Аналогично, запись %2\$.1f указывает на то, что второй аргумент после форматной строки должен заменять спецификатор формата %2\$.1f. Буква d в первом спецификаторе формата указывает на то, что выполняется форматирование в виде десятичного целого числа, а буква f во втором спецификаторе формата определяет форматирование в виде значения с плавающей точкой. В локализованных версиях файла strings.xml спецификаторы формата %1\$d и %2\$.1f могут переупорядочиваться в случае необходимости. То есть первый аргумент после форматной строки в вызове метода format класса Resourcesmethod getString или String может заменять %1\$d, а второй аргумент может заменять %2\$.1f (независимо от того, где они появляются в форматной строке).

Листинг 8.2. Строки, определенные в файле strings.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <resources>
3   <string name="app_name">Cannon Game</string>
4   <string name="results_format">
5     Shots fired: %1$d\nTotal time: %2$.1f</string>

```



```

6    <string name="reset_game">Reset Game</string>
7    <string name="win">You win!</string>
8    <string name="lose">You lose!</string>
9    <string name="time_remaining_format">
10       Time remaining: %.1f seconds</string>
11 </resources>

```

Файл main.xml. При создании приложения из этой главы удален файл main.xml, вместо которого подставлен файл, включающий FrameLayout. Единственный компонент в макете приложения — экземпляр пользовательского подкласса View под названием CannonView добавлен в проект. В листинге 8.3 приведен завершенный файл main.xml, в который вручную добавлен XML-элемент, показанный в строках 2–7. Этот элемент определяет, что компонент CannonView занимает всю ширину и высоту родительского макета, а также имеет белый фон.

Листинг 8.3. XML-разметка (main.xml) приложения Cannon Game

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <com.deitel.cannongame.CannonView
3      xmlns:android=http://schemas.android.com/apk/res/android
4      android:id="@+id/cannonView"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      android:background="@android:color/white"/>

```

Добавление звуков. Как уже упоминалось ранее, звуковые файлы хранятся в папке приложения res/raw. В рассматриваемом приложении используются три звуковых файла: blocker_hit.wav, target_hit.wav и cannon_fire.wav. Эти файлы находятся в папке sounds примеров к книге. Чтобы добавить эти файлы в проект, выполните следующие действия:

1. Щелкните правой кнопкой мыши на папке res приложения и выполните команды New Folder (Создать Папка).
2. Укажите имя папки raw и щелкните на кнопке Finish (Готово), чтобы создать папку.
3. Перетащите файлы звуков в папку res/raw.

8.4 Создание приложения

Это приложение включает три класса — Line (листинг 8.4), CannonGame (подкласс Activity; листинги 8.5–8.8) и CannonView (листинги 8.9–8.21) [1].

Определение концов линии с помощью класса Line. Класс Line (см. листинг 8.4) просто группирует две точки Points, которые представляют начальную и конечную точки линии (Point). Объекты из этого класса будут использованы для определения блока и мишени. Чтобы добавить класс Line в проект, выполните следующие действия:

1. Раскройте узел проекта src в окне обозревателя Package Explorer.
2. Щелкните правой кнопкой мыши на пакете (com.deitel.cannongame) и в контекстном меню выберите команды New Class (Создать Класс) для отображения диалогового окна New Java Class (Новый класс Java).
3. В поле Name (Имя), которое отображается в диалоговом окне, введите Line, а затем щелкните на кнопке Finish (Готово).
4. Введите код, представленный в листинге 7.4, в файл Line.java.

Листинг 8.4. Класс Line представляет линию двумя концами

```

1 // Line.java
2 // Класс Line, представляющий

```



```

    // линию двумя концами.

3 package com.deitel.cannongame;
4
5     import    android.graphics.Point;
6
7     public class Line
8     {
9         public Point start; // начальная точка
10        public Point end; // конечная точка
11
12        // заданный по умолчанию конструктор, который присваивает
13        // объектам Point (точки) начальные
14
15        // значения координат (0, 0)
16
17    public Line()
18    {
19        start = new Point(0, 0); // начальная точка
20        end = new Point(0, 0); // конечная точка
21    } // конец метода Line
22 } // конец класса Line

```

Подкласс CannonGame класса Activity. Класс CannonGame (см. листинги 8.5–8.8) является главным классом Activity приложения Cannon Game.

Операторы package, import и экземпляры переменных. В разделе 8.3 были рассмотрены новые ключевые классы и интерфейсы, которые использует класс CannonGame. Эти классы и интерфейсы выделены в листинге 8.5. В строке 15 определяется переменная cannonView, с помощью которой осуществляется взаимодействие между классами CannonGame и CannonView.

Листинг 8.5. Операторы package, import и экземпляры переменных класса CannonGame

```

1 // CannonGame.java
2 // Главный класс Activity для приложения Cannon Game.
3 package com.deitel.cannongame;
4
5 import android.app.Activity;
6 import android.os.Bundle;
7 import android.media.AudioManager;
8 import android.view.GestureDetector;
9 import android.view.MotionEvent;
10 import android.view.GestureDetector.SimpleOnGestureListener
11
12 public class CannonGame extends
13 Activity
14 {
15     private GestureDetector gestureDetector; // прослушивание
16                                             // двойных тапов
17
18     private CannonView cannonView; // пользовательское
19                                     // представление
20                                     //экрана игры
21
22 }

```

Переопределение методов onCreate, onPause и onDestroy класса Activity.

В листинге 8.6 представлены перегруженные методы onCreate (строки 18–32), onPause (строки 35–40) и onDestroy (строки 43–48) класса Activity. Метод onCreate создает структуру main.xml деятельности, затем получает ссылку на объект CannonView (строка 25). В строке 28 создается переменная GestureDetector, которая идентифицирует двойные тапы для текущей деятельности с помощью метода gestureListener, определенного в листинге 8.8. Оператор, находящийся в строке 31, обеспечивает регулирование громкости звукового сопровождения игры с помощью кнопок настройки громкости устройства Android.

Листинг 8.6. Переопределение методов onCreate, onPause и onDestroy класса Activity

```
17  /  вызывается при первом запуске приложения
18  @Override
19  public void onCreate(Bundle
savedInstanceState) 20 {
21      super.onCreate(savedInstanceState); // вызов метода onCreate
класса super
22      setContentView(R.layout.main); // создание
структуры 23
24      // получение CannonView
25      cannonView = (CannonView) findViewById(R.id.cannonView);
26
27      // Инициализация GestureDetector
28      gestureDetector = new GestureDetector(this,
gestureListener);
29
30      // Настройка громкости игры с помощью кнопок
// регулирования громкости устройства
31      setVolumeControlStream(AudioManager.STREAM_MUSIC);
32  } // конец метода onCreate
33
34  // пауза в случае, если приложение выполняется в фоновом режиме
35  @Override
36  public void onPause()
37  {
38      super.onPause(); // вызов метода super
39      cannonView.stopGame(); // завершение игры
40  } // конец метода onPause 41
42  // освобождение ресурсов
43  @Override
44  protected void onDestroy() 45
  {
46      super.onDestroy();
47      cannonView.releaseResources();
48  } // конец метода onDestroy
49  49
```

Метод onPause (строки 35–40) гарантирует, что деятельность CannonGame не будет выполняться, если она находится в фоновом режиме. Если игра продолжает выполняться, пользователь не может взаимодействовать с ней в силу того, что другая деятельность находится в фокусе. Но в этом случае все равно потребляется энергия аккумуляторной батареи — самый ценный ресурс для мобильных устройств. При вызове метода onPause в строке 39 вызывается метод stopGame класса cannonView (см. листинг 8.19). Этот метод выполняет завершение потока игры — в этом примере невозможно сохранить состояние игры.

Как только деятельность будет завершена, метод `onDestroy` (строки 43–46) вызывает метод `releaseResources` класса `cannonView` (см. листинг 8.19), который освобождает ресурсы звука приложения.

Переопределение метода `onTouchEvent` класса `Activity`. В этом примере будет выполнено переопределение метода `onTouchEvent` (см. листинг 8.7) для определения момента касания пальцем экрана или перемещения пальца вдоль экрана. Параметр `MotionEvent` содержит информацию о происшедшем событии. В строке 55 используется метод `getAction` класса `MotionEvent` для определения типа происходящего события. В строках 58–59 идентифицируется факт касания экрана пользователем (`MotionEvent.ACTION_DOWN`) либо перемещения пальца вдоль экрана (`MotionEvent.ACTION_MOVE`). В том или ином случае в строке 61 вызывается метод `alignCannon` класса `cannonView` (см. листинг 8.16), с помощью которого выполняется нацеливание пушки в точку касания экрана пальцем. В строке 65 объект `MotionEvent` передается методу `onTouchEvent` класса `gestureDetector` для определения, имел ли место двойной тап.

Листинг 8.7. Переопределение метода `onTouchEvent` класса `Activity`

```
50  // вызывается, если пользователь касается экрана
    // при выполняемой деятельности

51  @Override
52  public Boolean onTouchEvent(MotionEvent event)
53  {
54      // тип int, представляющий действие, вызываемое этим событием
55      int action = event.getAction();
56
57      // пользователь коснулся экрана или провел пальцем по нему
58      if (action == MotionEvent.ACTION_DOWN ||
59          action == MotionEvent.ACTION_MOVE)
60      {
61          cannonView.alignCannon(event); // нацеливание пушки
62      } // end if
63
64      // вызов метода onTouchEvent класса GestureDetector
65      return gestureDetector.onTouchEvent(event);
66  } // конец метода onTouchEvent
```

Анонимный внутренний класс, расширяющий класс `SimpleOnGestureListener`. Код из листинга 8.8 создает класс `SimpleOnGestureListener` под именем `gestureListener`, который регистрируется в строке 28 с помощью `GestureDetector`. Класс адаптера `SimpleOnGestureListener` реализует все методы интерфейсов `OnGestureListener` и `OnDoubleTapListener`. Эти методы возвращают значение `false`, свидетельствующее о том, что события не обрабатываются. Мы переопределяем лишь метод `onDoubleTap` (строки 71–76), вызывающийся после двойного тапа экрана пользователем. В строке 74 вызывается метод `fireCannonBall` класса `CannonView` (см. листинг 8.15), который выстреливает пушечное ядро. Метод `fireCannonBall` получает координаты двойного тапа пальцем экрана с помощью аргумента `MotionEvent`. Эти сведения используются для нацеливания пушки под правильным углом. В строке 75 возвращается значение `true`, свидетельствующее, что событие было обработано.

Листинг 8.8. Анонимный внутренний класс, расширяющий класс `SimpleOnGesture-Listener`

```
67      // прослушивание событий, связанных с касаниями,
    // отправленных GestureDetector
```

```

68     SimpleOnGestureListener    gestureListener
        = new    SimpleOnGestureListener()

69     {
70         //вызывается после двойного тапа пользователем экрана
71         @Override
72         public boolean onDoubleTap(MotionEvent e)
73         {

74             cannonView.fireCannonball(e); // стрельба ядром
75             return true; // событие было обработано
76         } // конец метода onDoubleTap
77     }; // конец gestureListener
78 } // конец класса CannonGame

```

Подкласс CannonView класса View. Класс CannonView (см. листинги 8.9–8.21) — это пользовательский подкласс класса View, который реализует логику игры Cannon Game и рисует объекты игры на экране. Чтобы добавить класс в проект, выполните следующие действия:

1. Раскройте узел src проекта в окне Package Explorer.
2. Щелкните правой кнопкой мыши на пакете (com.deitel.cannongame) и выполните команды New Class (Создать Класс) для отображения диалогового окна New Java Class (Новый класс Java).
3. В поле Name (Имя) диалогового окна введите название CannonView, в поле Superclass (Суперкласс) введите android.view.View, а затем щелкните на кнопке Finish (Готово).
4. Введите код, приведенный в листингах 8.9–8.19, в файл CannonView.java.

Операторы package и import. В листинге 8.9 приведены операторы package и import для класса CannonView.

Листинг 8.9. Операторы package и import класса CannonView

```

1 // CannonView.java
2 // Отображение Cannon Game
3 package com.deitel.cannongame;
4
5 import java.util.HashMap;
6 import java.util.Map;
7
8 import android.app.Activity;
9 import android.app.AlertDialog;
10 import android.content.Context;
11 import android.content.DialogInterface;
12 import android.graphics.Canvas;
13 import android.graphics.Color;
14 import android.graphics.Paint;
15 import android.graphics.Point;
16 import android.media.AudioManager;
17 import android.media.SoundPool;
18 import android.util.AttributeSet;
19 import android.view.MotionEvent;
20 import android.view.SurfaceHolder;
21 import android.view.SurfaceView;

```

Переменные и константы экземпляра класса CannonView

В листинге 8.10 перечислены константы и переменные экземпляра класса CannonView. Большинство из них не требуют дополнительных объяснений, но тем не менее будут рассмотрены по мере их применения в коде.

Листинг 8.10. Поля класса CannonView

```

23 public class CannonView extends SurfaceView
24     implements SurfaceHolder.Callback
25 {
26     private CannonThread cannonThread; // контроль цикла игры
27     private Activity activity; // отображение диалогового
        окна
        // Game Over в потоке GUI
28     private boolean dialogIsDisplayed =
false; 29
30     // константы, используемые в игре
31     public static final int TARGET_PIECES = 7; // секции
        мишени
32     public static final int MISS_PENALTY = 2; // секунды,
        вычитаемые
        // в случае промаха
33     public static final int HIT_REWARD = 3; // секунды,
        добавляемые
        // при попадании
34
35     // переменные, используемые для цикла игры и отслеживания
        статистики
36     private boolean gameOver; // игра завершена?
37     private double timeLeft; // оставшееся время в секундах
38     private int shotsFired; // количество выстрелов
        пользователя
39     private double totalTimeElapsed; // количество
        прошедших секунд 40
41     // переменные, используемые при определении блока и цели
42     private Line blocker; // начальная и конечная точки блока
43     private int blockerDistance; // расстояние от блока
        слева
44     private int blockerBeginning; // расстояние от блока
        сверху
45     private int blockerEnd; // расстояние от нижнего края
        блока сверху
46     private int initialBlockerVelocity; // множитель
        начальной
        // скорости блока
47     private float blockerVelocity; // множитель скорости
        блока
        // во время игры
48
49     private Line target; // начальная и конечная точки мишени
50     private int targetDistance; // дистанция до мишени слева
51     private int targetBeginning; // дистанция до мишени
        сверху

```

```

52     private double pieceLength; // длина секции мишени
53     private int targetEnd; // дистанция сверху до нижнего
    края мишени
54     private int initialTargetVelocity; // множитель
    начальной скорости
                                // мишени

55     private float targetVelocity; // множитель скорости
    мишени
                                // во время игры

56

57     private int lineWidth; // ширина мишени и блока
58     private boolean[] hitStates; // попадание во все секции
    мишени?
59     private int targetPiecesHit; // количество пораженных
    секций
                                // мишени (из 7)

60

61     // переменные для пушки и пушечного ядра
62     private Point cannonball; // изображение ядра в левом
    верхнем углу
63     private int cannonballVelocityX; // скорость ядра по
    горизонтали
64     private int cannonballVelocityY; // скорость ядра по
    вертикали
65     private boolean cannonballOnScreen; // ядро на экране
66     private int cannonballRadius; // радиус ядра
67     private int cannonballSpeed; // скорость ядра
68     private int cannonBaseRadius; // радиус основания пушки
69     private int cannonLength; // длина ствола пушки
70     private Point barrelEnd; // конец ствола пушки
71     private int screenWidth; // ширина экрана
72     private int screenHeight; // высота
    экрана 73
74     // константы и переменные, используемые для управления звуком
75     private static final int TARGET_SOUND_ID = 0;
76     private static final int CANNON_SOUND_ID = 1;
77     private static final int BLOCKER_SOUND_ID = 2;
78     private SoundPool soundPool; // звуковые эффекты
79     private Map<Integer, Integer> soundMap; // отображение ID
    // на SoundPool

80

81     // переменные Paint, используемые для рисования на экране
82     private Paint textPaint; // Paint, используемая для
    рисования текста
83     private Paint cannonballPaint; // Paint, используемая
    // для рисования ядра

84     private Paint cannonPaint; // Paint, используемая
    // для рисования пушки

85     private Paint blockerPaint; // Paint, используемая
    // для рисования блока

86     private Paint targetPaint; // Paint, используемая
    // для рисования мишени

```

```

87     private Paint backgroundPaint; // Paint, используемая
      // для чистки области рисования
88

```

Конструктор класса CannonView. Код листинга 8.11 определяет конструктор класса CannonView. После «раздувания» класса View вызывается его конструктор, а в качестве аргументов передаются Context и AttributeSet. В этом случае Context представляет собой класс Activity (CannonGame), к которому присоединен CannonView, а AttributeSet (пакет android.util) содержит значения для любых атрибутов, установленных в разметке XML-документа. Эти аргументы должны быть переданы конструктору суперкласса (строка 92) для обеспечения правильного конфигурирования пользовательского объекта View с помощью значений для любых стандартных атрибутов View, определенных в XML.

В строке 93 хранится ссылка на родительский класс Activity, который можно использовать в конце игры для отображения окна AlertDialog из потока Activity GUI. В строке 96 выполняется регистрация класса (то есть CannonView) в качестве объекта, который реализует SurfaceHolder.Callback для получения вызовов метода, которые свидетельствуют о том, что класс SurfaceView создан, обновлен и уничтожен. Метод getHolder класса SurfaceView возвращает соответствующий объект SurfaceHolder, используемый для управления SurfaceView, и метод addCallback класса SurfaceHolder, который хранит объект, реализующий метод SurfaceHolder.Callback.

Листинг 8.11. Конструктор класса CannonView

```

89 // общедоступный конструктор
90 public CannonView(Context context,
  AttributeSet attrs) 91 {
92     super(context, attrs); // вызов конструктора super
93     activity = (Activity)
  context; 94
95     // регистрация слушателя SurfaceHolder.Callback
96     getHolder().addCallback(this);
97
98     // инициализация линий и точек, представляющих элементы игры
99     blocker = new Line(); // создание блока в виде линии
100    target = new Line(); // создание мишени в виде линии
101    cannonball = new Point(); // создание ядра в
  виде точки 102
103    // инициализация hitStates в виде булевского массива
104    hitStates = new boolean[TARGET_PIECES];
105
106    // инициализация SoundPool, используемого для воспроизведения
  // трех звуковых эффектов
107
108    soundPool = new SoundPool(1, AudioManager.STREAM_MUSIC, 0);
109
110    // создание карты звуков и предварительная загрузка звуков
111    soundMap = new HashMap<Integer, Integer>(); // создание
  // нового HashMap
112
113    soundMap.put(TARGET_SOUND_ID,
114        soundPool.load(context, R.raw.target_hit, 1));
115    soundMap.put(CANNON_SOUND_ID,
116        soundPool.load(context, R.raw.cannon_fire, 1));
117    soundMap.put(BLOCKER_SOUND_ID,
118        soundPool.load(context, R.raw.blocker_hit, 1));

```



```

117
118    // конструирует Paints для рисования текста, ядра, пушки
119    // блока и мишени; все они конфигурируются с помощью
    // метода onSizeChanged
120    textPaint = new Paint(); // Paint для рисования текста
121    cannonPaint = new Paint(); // Paint для рисования пушки
122
123    cannonballPaint = new Paint(); // Paint для рисования ядра
124    blockerPaint = new Paint(); // Paint для рисования
    блока
125    targetPaint = new Paint(); // Paint для рисования
    мишени
126    backgroundPaint = new Paint(); // Paint для рисования фона
127 } // конец конструктора CannonView

```

В строках 99–101 создается блок и мишень с помощью объектов `Lines` и пушечное ядро с помощью объекта `Point`. Затем создается булев массив `hitStates`, предназначенный для отслеживания попаданий в секции мишени (которые при этом исчезают с экрана).

В строках 107–116 конфигурируются звуки, используемые в приложении. Сначала создается объект `SoundPool`, применяемый для загрузки и воспроизведения звуковых эффектов, используемых в приложении. Первый аргумент представляет максимальное количество звуковых потоков, которые могут воспроизводиться одновременно. В игре воспроизводится лишь один звук, поэтому значение этого аргумента равно 1. Второй аргумент определяет аудиопоток, который будет использоваться для воспроизведения звуков. Используются семь аудиопотоков, идентифицируемых константами класса `AudioManager`. Обратите внимание, что в документации для класса `SoundPool` рекомендуется использовать поток воспроизведения музыки (`AudioManager.STREAM_MUSIC`) для воспроизведения звука в играх. Последний аргумент представляет качество звука, хотя в документации указывается, что в данное время это значение не используется (в качестве используемого по умолчанию значения указано 0).

В строке 110 создается объект `HashMap` (переменная `soundMap`). В строках 111–116 задаются значения с помощью констант, используемых в качестве ключей (строки 75–77). Соответствующие значения представляют собой возвращаемые значения метода `load` класса `SoundPool`, возвращающие идентификатор ID, который может использоваться для воспроизведения (или выгрузки) звука. Метод `load` класса `SoundPool` получает три аргумента — контент (`Context`) приложения, идентификатор (ID), представляет загружаемый звуковой файл, а также приоритет (`priority`) звука. В соответствии с документацией, описывающей этот метод, последний аргумент в настоящее время не используется и имеет значение 1.

В строках 120–125 создаются объекты `Paint`, используемые для рисования объектов игры. Эти объекты конфигурируются с помощью метода `onSizeChanged`, поскольку некоторые из настроек объекта `Paint` зависят от масштаба элементов игры, основанном на размере экрана устройства.

Overriding View Method `onSizeChanged`. В коде из листинга 8.12 выполняется переопределение метода `onSizeChanged` класса `View`, который вызывается в случае изменения размеров класса `View`, например при первом добавлении класса `View` в иерархию классов `View` в процессе «раздувания» разметки. Это приложение всегда отображается в портретном режиме, поэтому метод `onSizeChanged` вызывается только в том случае, если метод `onCreate` деятельности «раздувает» GUI. Этот метод

принимает новые значения ширины и высоты класса View. Прежним значениям ширины и высоты (при первом вызове метода) присваиваются значения 0. При выполнении вычислений, реализующих масштабирование изображения на экране, используются размеры экрана устройства (ширина и высота, выраженные в пикселях). С помощью метода «проб и ошибок» получаем значения множителей масштабирования. После выполнения вычислений в строке 173 вызывается метод newGame (см. листинг 8.13).

Листинг 8.12. Переопределенный метод onSizeChanged

```
128 // вызывается при изменении размеров представления, а также
129 // при первом добавлении в иерархию представлений
130 @Override
131 protected void onSizeChanged(int w, int h, int oldw,
    int oldh)
132 {
133     super.onSizeChanged(w, h, oldw, oldh);
134
135     screenWidth = w; // хранение значения ширины
136     screenHeight = h; // хранение значения высоты
137     cannonBaseRadius = h / 18; // радиус основания пушки,
    равный
    // 1/18 от высоты экрана

138     cannonLength = w / 8; // длина пушки равна 1/8 от
    ширины экрана
139
140     cannonballRadius = w / 36; // радиус ядра равен 1/36
    ширины экрана
141
142     cannonballSpeed = w * 3 / 2; // множитель
    скорости ядра
143
144     lineWidth = w / 24; // ширина мишени и блока равны 1/24
    // от ширины экрана

145
146     // конфигурирование переменных экземпляра, связанных с блоком
    blockerDistance = w * 5 / 8; // зазор слева от блока
    равен
    // 5/8 ширины экрана

147     blockerBeginning = h / 8; // зазор сверху от блока равен
    // 1/8 от высоты экрана

148     blockerEnd = h * 3 / 8; // зазор сверху от блока равен
    3/8
    // от высоты экрана

149     initialBlockerVelocity = h / 2; // начальный множитель
    // скорости блока

150     blocker.start = new Point(blockerDistance,
    blockerBeginning);
151     blocker.end = new Point(blockerDistance,
    blockerEnd);
152
153     // конфигурирование переменных экземпляра, связанных с мишенью
154     targetDistance = w * 7 / 8; // зазор слева от мишени
    равен
    // 7/8 от ширины экрана

155     targetBeginning = h / 8; // зазор сверху от мишени равен
    // 1/8 от высоты экрана
```

```

156     targetEnd = h * 7 / 8; // зазор сверху равен 7/8 от
        высоты экрана
157     pieceLength = (targetEnd - targetBeginning) /
        TARGET_PIECES;
158     initialTargetVelocity = -h / 4; // множитель
        начальной
        // скорости мишени

159     target.start = new Point(targetDistance,
        targetBeginning);
160     target.end = new Point(targetDistance,
        targetEnd); 161
162     // конечная точка ствола пушки (изначально направлен по
        горизонтали)
163     barrelEnd = new Point(cannonLength, h /
        2); 164
165     // конфигурирование объектов Paint для рисования элементов
        игры
166     textPaint.setTextSize(w / 20); // размер текста равен
        1/20
        // от ширины экрана

167     textPaint.setAntiAlias(true); // сглаживание текста
168     cannonPaint.setStrokeWidth(lineWidth * 1.5f); //
        настройка
        // толщины линии

169     blockerPaint.setStrokeWidth(lineWidth); // настройка
        толщины
        // линии

170     targetPaint.setStrokeWidth(lineWidth); // настройка
        толщины линии
171     backgroundPaint.setColor(Color.WHITE); // настройка
        фонового цвета
172
173     newGame(); // первоначальная настройка и запуск новой игры
174 } // end method onSizeChanged
175

```

Метод newGame класса CannonView. Метод newGame (см. листинг 8.13) сбрасывает начальные значения переменных, используемых для управления игрой. Если значение переменной gameOver равно true, что бывает только после окончания первой игры, в строке 197 переустанавливается значение переменной gameOver, а в строках 198–199 создается и запускается новый поток CannonThread для начала новой игры.

Листинг 8.13. Метод newGame класса CannonView

```

176 // переустановка всех элементов экрана и запуск новой игры
177 public void newGame()
178 {
179     // присваивание каждому элементу массива hitStates значения
        // false — восстановление секций мишени

180     for (int i = 0; i < TARGET_PIECES; ++i)
181         hitStates[i] = false;
182
183     targetPiecesHit = 0; // нет попаданий в секции мишени
184     blockerVelocity = initialBlockerVelocity; //
        настройка

```

```

// начальной скорости
185   targetVelocity = initialTargetVelocity;    //
      настройка
// начальной скорости

186   timeLeft = 10; // начало обратного отсчета 10 секунд
187   cannonballOnScreen = false; // ядро не отображается на
      экране
188   shotsFired = 0; // установка начального числа выстрелов
189   totalElapsedTime = 0.0; // обнуление времени игры
190   blocker.start.set(blockerDistance, blockerBeginning);
191   blocker.end.set(blockerDistance, blockerEnd);
192   target.start.set(targetDistance, targetBeginning);
193   target.end.set(targetDistance, targetEnd);
194
195   if (gameOver)
196   {
197       gameOver = false; // игра не завершена
198       cannonThread = new CannonThread(getHolder());
199       cannonThread.start();
200   } // конец блока if
201 } // конец метода newGame
202

```

Метод updatePositions класса CannonView. Метод updatePositions (см. листинг 8.14) вызывается методом run класса CannonThread (см. листинг 8.21), с помощью которого выполняется обновление положения элементов на экране и простое обнаружение столкновений. Новые положения элементов игры на экране вычисляются на основе времени (в миллисекундах), прошедшего между предыдущим и текущим кадром анимации. С помощью этого метода игра обновляет величину расстояния, на которое перемещается каждый элемент игры (при этом используется скорость обновления изображения на экране устройства). Эти вопросы будут разобраны более подробно при рассмотрении циклов игры (см. листинг 8.21).

Листинг 8.14. Метод updatePositions класса CannonView

```

203 // вызывается повторно с помощью CannonThread для обновления
      // элементов игры

204 private void updatePositions(double elapsedTimeMS)
205 {
206     double interval = elapsedTimeMS / 1000.0; //
      преобразование
// в секунды

207
208     if (cannonballOnScreen) // был ли текущий выстрел
209     {
210         // обновление положения ядра
211         cannonball.x += interval * cannonballVelocityX;
212         cannonball.y += interval *
cannonballVelocityY; 213
214         // проверка столкновения с блоком
215         if (cannonball.x + cannonballRadius > blockerDistance
&&
216             cannonball.x - cannonballRadius < blockerDistance &&
217             cannonball.y + cannonballRadius > blocker.start.y &&
218             cannonball.y - cannonballRadius < blocker.end.y)

```

```

219         {
220             cannonballVelocityX *= -1; // реверсирование
                // направления полета ядра

221             timeLeft -= MISS_PENALTY; // штраф для
пользователя 222
223             // воспроизведение звука блока
224             soundPool.play(soundMap.get(BLOCKER_SOUND_ID),
                1, 1, 1, 0, 1f)

225         } // конец блока if
226
227         // проверка столкновений с левой и правой стенками
228         else if (cannonball.x + cannonballRadius >
screenWidth ||
229             cannonball.x - cannonballRadius < 0)
230             cannonballOnScreen = false; // remove
                // изображения ядра с экрана

231
232         // проверка столкновений с верхней и нижней стенками
233         else if (cannonball.y + cannonballRadius >
screenHeight ||
234             cannonball.y - cannonballRadius < 0)
235             cannonballOnScreen = false; // скрывание
                // изображения ядра

236
237         // проверка столкновения ядра с мишенью
238         else if (cannonball.x + cannonballRadius >
targetDistance &&
239             cannonball.x - cannonballRadius < targetDistance
&&
240             cannonball.y + cannonballRadius > target.start.y
&&
241             cannonball.y - cannonballRadius < target.end.y)
242         {
243             // определение номера секции мишени (0 сверху)
244             int section =
245                 (int) ((cannonball.y - target.start.y) /
pieceLength); 246
247             // проверка попадания в секцию мишени
248             if ((section >= 0 && section < TARGET_PIECES) &&
249                 !hitStates[section])
250             {
251                 hitStates[section] = true; // попадание в секцию
252                 cannonballOnScreen = false; // удаление ядра
253                 timeLeft += HIT_REWARD; // добавление
                // дополнительного времени

254
255                 // воспроизведение звука столкновения с целью
256                 soundPool.play(soundMap.get(TARGET_SOUND_ID), 1,
257                     1, 1, 0, 1f);

258
259                 // если были попадания во все секции мишени
260                 if (++targetPiecesHit == TARGET_PIECES)

```

```

261     {
262         cannonThread.setRunning(false);
263         showGameOverDialog(R.string.win); // показать
                                         // диалоговое окно выигрыша

264         gameOver = true; // игра завершена
265     } // конец блока if
266 } // конец блока if
267 } // конец блока else if
268 } // конец блока if
269
270 // обновление позиции блока
271 double blockerUpdate = interval * blockerVelocity;
272 blocker.start.y += blockerUpdate;
273 blocker.end.y += blockerUpdate;
274
275 // обновление позиции мишени
276 double targetUpdate = interval * targetVelocity;
277 target.start.y += targetUpdate;
278 target.end.y += targetUpdate;
279
280 // при попадании в верхнюю или нижнюю часть блока
    // реверсировать направление движения

281 if (blocker.start.y < 0 || blocker.end.y >
    screenHeight)
282     blockerVelocity *= -1;
283
284 // при попадании в верхнюю или нижнюю часть блока
    // реверсировать направление движения

285 if (target.start.y < 0 || target.end.y >
    screenHeight)
286     targetVelocity *= -1;
287
288 timeLeft -= interval; // вычитание из оставшегося
    времени
289
290 // если показания таймера равны нулю
291 if (timeLeft <= 0)
292 {
293     timeLeft = 0.0;
294     gameOver = true; // игра завершена
295     cannonThread.setRunning(false);
296     showGameOverDialog(R.string.lose); // диалоговое окно
        проигрыша
297 } // end if
298 } // конец метода
    updatePositions
299

```

В строке 206 преобразуется значение времени, прошедшего с момента последнего кадра анимации (от нескольких миллисекунд до секунд). Это значение используется для изменения позиций различных элементов игры.

В строке 208 проверяется, отображается ли пушечное ядро на экране. Если ядро отображается на экране, происходит обновление его позиции путем добавления расстояния, которое должно быть пройдено с момента последнего события таймера. Расстояние вычисляется путем умножения значения скорости на величину

прошедшего времени (строки 211–212). В строках 215–218 проверяется, было ли столкновение ядра с блоком. Обнаружение столкновений мы будем производить, основываясь на рамке вокруг ядра. В случае столкновения ядра с блоком должны быть выполнены четыре условия:

- Результат суммирования координаты x пушечного ядра и радиуса ядра должен быть больше, чем расстояние от блока до левой границы экрана (`blockerDistance`), строка 215. Это означает, что ядро достигло блока, «пролетев» расстояние, отделяющее блок от левой границы экрана.
- Результат вычитания радиуса ядра из значения координаты x этого же ядра должен быть меньше, чем расстояние от блока до левого края экрана (строка 216). Это гарантирует, что ядро еще не миновало блок.
- Часть ядра должна располагаться ниже, чем верхняя часть блока (строка 217).
- Часть ядра должна располагаться выше, чем нижняя часть блока (строка 218).

Если все эти условия выполняются, направление движения ядра реверсируется (строка 220), пользователю начисляется штраф путем вычитания значения переменной `MISS_PENALTY` из значения переменной `timeLeft`, затем вызывается метод `play` класса `soundPool` для воспроизведения звука, сопровождающего попадание в блок. В качестве ключа `soundMap` используется `BLOCKER_SOUND_ID` для поиска идентификатора (ID) звука в `SoundPool`.

Изображение ядра исчезает с экрана после столкновения с любой из сторон экрана. В строках 228–230 проверяется, столкнулось ли ядро с левой либо с правой стенкой, и если это так, изображение ядра удаляется с экрана. В строках 233–235 удаляется изображение ядра в случае, если ядро сталкивается с верхним или нижним краем экрана.

Затем выполняется проверка столкновения ядра с мишенью (строки 238–241). При этом проверяются условия, подобные условиям столкновения ядра с блоком. Если ядро попало в мишень, идентифицируется секция мишени, в которую попало ядро. В строках 244–245 определяется секция мишени, в которую попало ядро, путем деления расстояния между ядром и нижней частью мишени на длину секции мишени. Значение этого выражения равно 0 для самой верхней секции мишени, и 6 — для самой нижней секции мишени. Чтобы проверить попадание ядра в секцию мишени, используется массив `hitStates` (строка 249). Если попадание имело место, соответствующему элементу массива `hitStates` присваивается значение `true`, а изображение ядра удаляется с экрана. Затем значение переменной `HIT_REWARD` добавляется в счетчик `timeLeft`, увеличивая тем самым значение оставшегося времени, и воспроизводится звук попадания в мишень (`TARGET_SOUND_ID`). Увеличивается значение переменной `targetPiecesHit`, определяется, равно ли оно значению переменной `TARGET_PIECES` (строка 260). Если это условие выполнено, игра завершается. При этом завершается выполнение потока `CannonThread` путем вызова метода `setRunning` с аргументом `false`, вызывается метод `showGameOverDialog` с идентификатором (ID) строкового ресурса, который определяет сообщение о выигрыше и присваивает переменной `gameOver` значение `true`.

Теперь, когда были проверены все возможные столкновения пушечного ядра, следует обновить значения позиций мишени и блока. В строках 271–273 изменяется значение позиции блока путем умножения значения переменной `blockerVelocity` на количество времени, которое прошло с момента последнего обновления и добавления значения к текущим координатам x и y . В строках 276–278 выполняются те же операции по отношению к мишени. Если блок сталкивается с верхней или нижней стенками, направление его движения реверсируется путем умножения значения скорости на -1 (строки 281–282). В строках 285–286 выполняются те же проверки и настройки для мишени, имеющей полную длину, включая секции мишени, в которые

было попадание ядра.

Значение переменной `timeLeft` уменьшается на время, которое прошло с момента воспроизведения предыдущего кадра анимации. Если значение переменной `timeLeft` становится равным нулю, игра завершается. Переменной `timeLeft` присваивается значение 0.0 только в том случае, если она имеет отрицательное значение. Если этого не сделать, на экране может отобразиться отрицательное значение времени. Затем переменной `gameOver` присваивается значение `true`, прекращается выполнение потока `CannonThread` путем вызова метода `setRunning` с аргументом `false`, а также вызова метода `showGameOverDialog` с ID строкового ресурса, используемого для генерирования сообщения о проигрыше.

Метод `fireCannonball` класса `CannonView`. Если пользователь выполнит двойной тап экрана, обработчик событий для этого события (см. листинг 8.8) вызывает метод `fireCannonball` (см. листинг 8.15), с помощью которого выполняется выстрел из пушки. Если изображение ядра уже имеется на экране, метод тут же завершается. В противном случае (изображение ядра на экране отсутствует) происходит выстрел из пушки. В строке 306 вызывается метод `alignCannon`, с помощью которого нацеливается пушка в точку двойного тапа, а также считывается значение угла наклона пушки. В строках 309–310 «загружается» пушка (то есть положение пушечного ядра внутри пушки). Затем в строках 313 и 316 вычисляются горизонтальная и вертикальная компоненты скорости пушечного ядра. Затем переменной `cannonballOnScreen` присваивается значение `true`, в результате чего рисуется пушечное ядро с помощью метода `drawGameElements` (см. листинг 8.17) и увеличивается значение переменной `shotsFired`. И в заключение воспроизводится звук выстрела пушки (`CANNON_SOUND_ID`).

Листинг 8.15. Метод `fireCannonball` класса `CannonView`

```
// выстрел из пушки ядром
300 public void fireCannonball(MotionEvent event)
301 {
302     if (cannonballOnScreen) // если ядро отображается на экране
303         return; // ничего не делать
304
305     double angle = alignCannon(event); //
306     // помещение ядра вовнутрь ствола пушки
307     cannonball.x = cannonballRadius; // наклон
308     cannonball.y = screenHeight / 2; //
309     // получение x-компонента общей скорости
310     cannonballVelocityX = (int)
311     // получение y-компонента общей скорости
312     cannonballVelocityY = (int) (-
313     cannonballOnScreen = true; // ядро
314     ++shotsFired; // увеличение значения
315     // воспроизведение звука выстрела из пушки
316     soundPool.play(soundMap.get(CANNON_SOUND_ID),
317     } // end method fireCannonball
318
319
320
321
322
323
```

Метод `alignCannon` класса `CannonView`. Метод `alignCannon` (см. листинг 8.16) нацеливает пушку на точку двойного тапа экрана пользователем. В строке 328 выполняется получение координат `x` и `y` двойного тапа (с помощью аргумента `MotionEvent`). Вычисляется расстояние по вертикали от места касания до центра экрана. Если это расстояние не равно нулю, вычисляется угол наклона ствола пушки

относительно горизонта (строка 338). Если касание экрана произошло в его нижней части, настройка угла выполняется с помощью метода `Math.PI` (строка 342). С помощью переменных `cannonLength` и `angle` определяются координаты *x* и *y* для конечной точки ствола пушки. Затем рисуется линия в направлении от центра основания пушки, находящегося возле левого края экрана, до конечной точки ствола пушки.

Листинг 8.16. Метод `alignCannon` класса `CannonView`

```

324 // изменения угла наклона пушки в ответ на касания экрана
325 public double alignCannon(MotionEvent event)
326 {
327     // получение места касания в этом представлении
328     Point touchPoint = new Point((int)
        event.getX(), (int) event.getY());
329
330     // вычисление расстояния от места касания до центра
331     // экрана по оси y
332     double centerMinusY = (screenHeight / 2 -
        touchPoint.y);
333
334     double angle = 0; // инициализация угла
        значением 0
335
336     // вычисление угла наклона ствола относительно горизонта
337     if (centerMinusY != 0) // предотвращает деление на 0
338         angle = Math.atan((double) touchPoint.x /
        centerMinusY);
339
340     // в случае касания нижней половины экрана
341     if (touchPoint.y > screenHeight / 2)
342         angle += Math.PI; // изменение угла
343
344     // вычисление конечной точки ствола пушки
345     barrelEnd.x = (int) (cannonLength * Math.sin(angle));
346     barrelEnd.y =
347         (int) (-cannonLength * Math.cos(angle) +
        screenHeight / 2);
348
349     return angle; // возвращение вычисленного угла
350 } // конец метода alignCannon
351

```

Рисование элементов игры. Метод `drawGameElements` (см. листинг 8.17) рисует пушку, пушечное ядро, блок и мишень с помощью класса `SurfaceView`. При этом используется компонент `Canvas`, который класс `CannonThread` получает из компонента `SurfaceHolder` класса `SurfaceView`.

Листинг 8.17. Метод `drawGameElements` класса `CannonView`

```

352 // рисование элементов игры с помощью выбранного Canvas
353 public void drawGameElements(Canvas canvas)
354 {
355     // очистка фона
356     canvas.drawRect(0, 0, canvas.getWidth(),
        canvas.getHeight(),
357         backgroundPaint);
358
359     // отображение оставшегося времени
360     canvas.drawText(getResources().getString(
361         R.string.time_remaining_format, timeLeft), 30, 50,
        textPaint);
362
363     // если пушечное ядро на экране, нарисуйте его

```



```

364     if (cannonballOnScreen)
365         canvas.drawCircle(cannonball.x, cannonball.y,
366             cannonballRadius,
367             cannonballPaint);

368     // рисование ствола пушки
369     canvas.drawLine(0, screenHeight / 2, barrelEnd.x,
370         barrelEnd.y,
371         cannonPaint);

372     // рисование основания пушки
373     canvas.drawCircle(0, (int) screenHeight / 2,
374         (int) cannonBaseRadius, cannonPaint);
375
376     // рисование блока
377     canvas.drawLine(blocker.start.x, blocker.start.y,
378         blocker.end.x,
379         blocker.end.y, blockerPaint);

380     Point currentPoint = new Point(); // начало текущей
381     секции мишени

382     // присваивание curPoint начальной точки мишени
383     currentPoint.x = target.start.x;
384     currentPoint.y =
385     target.start.y;
386     // рисование мишени
387     for (int i = 1; i <= TARGET_PIECES; ++i)
388     {
389         // если нет попадания в секцию мишени, нарисовать ее
390         if (!hitStates[i - 1])
391         {
392             // чередующееся раскрашивание частей мишени желтым и
393             синим
394             if (i % 2 == 0)
395                 targetPaint.setColor(Color.YELLOW);
396             else
397                 targetPaint.setColor(Color.BLUE);

398             canvas.drawLine(currentPoint.x, currentPoint.y,
399                 target.end.x,
400                 (int) (currentPoint.y + pieceLength),
401                 targetPaint);
402         }

403         // перемещение curPoint к началу следующей секции
404         currentPoint.y += pieceLength;
405     } // конец метода for

406     // конец метода drawGameElements

```

Сначала вызывается метод `drawRect` класса `Canvas` (строки 356–357), с помощью которого выполняется очистка объекта `Canvas`. В результате все элементы

игры отображаются в новых положениях. Этот метод получает в качестве аргументов координаты верхнего левого угла прямоугольника, ширину и высоту прямоугольника и объект Paint, определяющий характеристики рисования (вспомните, как с помощью `backgroundPaint` выбирался белый цвет рисования). Затем вызывается метод `drawText` класса `Canvas` (строки 360–361) для отображения оставшегося времени игры. В качестве аргументов передаются отображаемая строка, координаты *x* и *y* отображаемых объектов и компонент `textPaint` (skonfigurirovann в строках 166–167), с помощью которого описывается отображаемый текст (размер шрифта, цвет и другие атрибуты текста).

Если пушечное ядро отображается на экране, в строках 365–366 используется метод `drawCircle` класса `Canvas` для рисования ядра в его текущей позиции. Первые два аргумента представляют координаты центра окружности. Третий аргумент представляет собой радиус окружности. Последний аргумент — это объект Paint, определяющий характеристики, используемые при рисовании окружности.

С помощью метода `drawLine` класса `Canvas` отображается ствол пушки (строки 369–370), блок (строки 377–378) и части мишени (строки 398–399). Этот метод получает пять параметров — первые четыре параметра представляют координаты *x*-*y* начала и конца линии, а последний — объект Paint, определяющий характеристики линии, например ее толщину.

В строках 373–374 используется метод `drawCircle` класса `Canvas` для рисования полукруглого основания пушки. При этом рисуется окружность, центр которой находится в левом углу экрана. И поскольку центр окружности находится в районе левого края экрана, половина окружности рисуется за пределами левой границы экрана компонента `SurfaceView`.

В строках 380–404 рисуются секции мишени. Осуществляется последовательный обход секций мишени, для каждой из которых используется корректный цвет (синий для секций с нечетными номерами и желтый для других секций). На экране отображаются секции мишени, в которые не попало ядро.

Метод `showGameOverDialog` класса `CannonView`. После завершения игры метод `showGameOverDialog` (см. листинг 7.18) выводит диалоговое окно `AlertDialog`, в котором отображается сообщение о том, выиграл или проиграл пользователь, показано количество произведенных выстрелов и общее время игры. В строках 419–430 вызывается метод `setPositiveButton` класса `Builder`, с помощью которого создается кнопка переустановки. Метод `onClick` слушателя кнопок сигнализирует о том, что диалоговое окно больше не отображается, и вызывает метод `newGame` для настройки и запуска новой игры. Диалоговое окно должно отображаться в потоке GUI, поэтому в строках 432–440 вызывается метод `runOnUiThreadThread` класса `Activity` и передается объекту анонимного внутреннего класса, который реализует класс `Runnable`. Метод `run` класса `Runnable` проверяет возможность отображения диалогового окна и отображает его.

Листинг 8.18. Метод `showGameOverDialog` класса `CannonView`

```
407 // отображение диалогового окна AlertDialog после завершения
    игры
408 private void showGameOverDialog(int messageId)
409 {
410     // создание диалогового окна, отображающего данную строку
411     final AlertDialog.Builder dialogBuilder =
412         new AlertDialog.Builder(getContext());
413     dialogBuilder.setTitle(getResources().getString(messageI
        d));
414     dialogBuilder.setCancelable(false);
415 }
```

```

416 // отображение количества выстрелов и общего времени игры
417 dialogBuilder.setMessage(getResources().getString(
418     R.string.results_format, shotsFired,
419     totalElapsedTime));
419 dialogBuilder.setPositiveButton(R.string.reset_game,
420     new DialogInterface.OnClickListener()
421     {
422         // вызывается при нажатии кнопки "Reset Game"
423         @Override
424         public void onClick(DialogInterface dialog, int
425             which)
426         {
427             dialogIsDisplayed = false;
428             newGame(); // настройка и запуск новой игры
429         } // конец метода onClick
430     } // конец анонимного внутреннего класса
431 ); // конец вызова
432 setPositiveButton 431
432 activity.runOnUiThread(
433     new Runnable() {
434         public void run()
435         {
436             dialogIsDisplayed = true;
437             dialogBuilder.show(); // отображение диалогового
438             окна
439         } // конец метода run
440     } // конец класса Runnable
441 ); // конец вызова runOnUiThread
442 } // конец метода showGameOverDialog

```

Методы stopGame и releaseResources класса CannonView. Методы onPause и onDestroy (см. листинг 8.6) класса CannonGame (класс Activity) вызывают методы stopGame и releaseResources класса CannonView (см. листинг 8.19) соответственно. Метод stopGame (строки 444–448) вызывается из главного класса Activity и инициирует остановку игры после вызова метода onPause класса Activity. В целях упрощения в этом примере не хранятся сведения о состоянии игры. Метод releaseResources (строки 451–455) вызывает метод release класса SoundPool, с помощью которого выполняется освобождение ресурсов, связанных с SoundPool.

Листинг 8.19. Методы stopGame и releaseResources класса CannonView

```

443 // пауза в игре
444 public void stopGame()
445 {
446     if (cannonThread != null)
447         cannonThread.setRunning(false);
448 } // конец метода stopGame 449
450 // освобождает ресурсы; вызывается методом onDestroy
451 // класса CannonGame
452 public void releaseResources()
453 {
454     soundPool.release(); // освобождает все ресурсы,
455                         // используемые SoundPool
456     soundPool = null;

```

```

455 } // конец метода
456

```

Реализация методов SurfaceHolder.Callback. В листинге 8.20 реализованы методы `surfaceChanged`, `surfaceCreated` и `surfaceDestroyed` интерфейса `SurfaceHolder.Callback`. Метод `surfaceChanged` является «пустышкой» в коде приложения, поскольку приложение всегда выводит изображение в портретном режиме. Этот метод вызывается в случае изменения размеров или ориентации компонента `SurfaceView` и обычно используется для повторного отображения графики после изменений. Метод `surfaceCreated` (строки 465–471) вызывается после создания `SurfaceView` (то есть в случае, если приложение сначала загружается или восстанавливается из приостановленного состояния). С помощью метода `surfaceCreated` создается и запускается поток `CannonThread` для начала выполнения игры. Метод `surfaceDestroyed` (строки 474–492) вызывается в случае разрушения `SurfaceView` (в случае завершения выполнения приложения). Метод `CannonThread` гарантирует корректное завершение приложения. Сначала в строке 479 вызывается метод `setRunning` класса `CannonThread` с аргументом `false`, это свидетельствует о том, что поток будет завершен. В строках 481–491 задается интервал ожидания до завершения потока. Это гарантирует, что после завершения выполнения метода `surfaceDestroyed` не будут предприниматься попытки рисования в области `SurfaceView`.

Листинг 8.20. Реализация методов `SurfaceHolder.Callback`

```

457 // вызывается в случае изменения размеров поверхности
458 @Override
459 public void surfaceChanged (SurfaceHolder holder, int
    format,
460     int width, int height)
461 {
462 } // конец метода surfaceChanged 463
464 // вызывается при первом создании поверхности
465 @Override
466 public void SurfaceCreated (SurfaceHolder holder)
467 {
468     cannonThread = new CannonThread(holder);
469     cannonThread.setRunning(true);
470     cannonThread.start(); // начало потока цикла игры
471 } // конец метода surfaceCreated
472
473 // вызывается при разрушении поверхности
474 @Override
475 public void surfaceDestroyed (SurfaceHolder holder)
476 {
477     // корректное
478     boolean retry =
479     cannonThread.setRunni
480
481     while (retry)
482     {
483         try
484         {
485             cannonThread.joi
486             retry =
487         } // конец
488         catch (InterruptedException e)
489     {

```

```

490     }    // конец блока catch
491 }    // конец блока while
492 } // конец метода surfaceDestroyed
493

```

Поток CannonThread: использование потока для создания цикла игры

В листинге 8.21 определяется подкласс класса Thread при обновлении игры. Поток поддерживает ссылку на метод SurfaceHolder класса SurfaceView (строка 497), и с помощью булевого значения определяется, выполняется ли поток. Метод класса run (строки 514–543) управляет покадровыми анимациями (этот процесс известен как цикл игры). Каждое обновление элементов игры на экране выполняется на основании количества миллисекунд, прошедших с момента последнего обновления. В строке 518 считывается текущее системное время, выраженное в миллисекундах, прошедшее с момента начала выполнения потока. В строках 520–542 выполняется цикл до тех пор, пока значение переменной threadIsRunning не станет равным false.

Листинг 8.21. Класс Runnable, обновляющий игру через интервал времени, заданный переменной TIME_INTERVAL (в миллисекундах)

```

494 // Подкласс Thread, используемый для управления циклом игры
495 private class CannonThread extends Thread
496 {
497     private SurfaceHolder surfaceHolder; // управление canvas
498     private boolean threadIsRunning = true; // выполнение по
умолчанию 499
500     // инициализация контейнера поверхности
501     public CannonThread(SurfaceHolder holder)
502     {
503         surfaceHolder = holder;
504         setName("CannonThread");
505     } // конец конструктора
506
507     // изменение состояния выполнения
508     public void setRunning(boolean running)
509     {
510         threadIsRunning = running;
511     } // конец метода setRunning
512
513     // управление циклом игры
514     @Override
515     public void run()
516     {
517         Canvas canvas = null; // используется для рисования
518         long previousFrameTime =
System.currentTimeMillis(); 519
520         while (threadIsRunning)
521         {
522             try

```

```

523      {
524          canvas = surfaceHolder.lockCanvas (null) ;
525
526          // блокирование surfaceHolder для рисования
527          synchronized (surfaceHolder)
528          {
529              long currentTime = System.currentTimeMillis();
530              double elapsedTimeMS = currentTime -
previousFrameTime;
531              totalElapsedTime += elapsedTimeMS / 1000.0;
532              updatePositions (elapsedTimeMS); // обновление
// состояния игры
533
drawGameElements (canvas); // рисование
534              previousFrameTime = currentTime; // обновление
// предыдущего времени
535          } // конец синхронизированного блока
536      } // конец блока try
537      finally
538      {
539          if (canvas != null)
540              surfaceHolder.unlockCanvasAndPost (canvas);
541          } // конец finally
542      } // конец блока while
543  } // конец метода run
544 } // конец вложенного класса CannonThread
545 } // конец класса CannonView

```

Сначала получим объект Canvas, используемый для рисования, на SurfaceView путем вызова метода lockCanvas класса SurfaceHolder (строка 524). Для рисования на SurfaceView может использоваться одновременно лишь один поток, поэтому для выполнения синхронизации блока сначала нужно заблокировать SurfaceHolder. Затем снимаются показания текущего времени (в миллисекундах), вычисляется время выполнения программы и добавляется к счетчику суммарного времени. Это значение применяется для отображения количества оставшегося времени игры. В строке 532 вызывается метод updatePositions, аргументом которого является время выполнения приложения, заданное в миллисекундах. Этот метод перемещает все элементы игры, используя время выполнения приложения для облегчения масштабирования величины перемещения. В результате обеспечивается одна и та же скорость выполнения игры независимо от используемого устройства. Если интервал времени между соседними кадрами увеличивается (при использовании более медленного устройства), элементы игры будут перемещаться на большее расстояние при отображении очередного кадра анимации. Если же интервал времени между кадрами меньше (для более быстрого устройства), элементы игры будут перемещаться на меньшее расстояние при отображении очередного кадра. И наконец, в строке 533 рисуются элементы игры с помощью метода Canvas класса SurfaceView, а в строке 534 хранится значение переменной currentTime в виде предыдущего previousFrameTime для подготовки вычисления времени выполнения приложения в следующем кадре анимации.

8.5 Резюме

В этой главе вы создали приложение Cannon Game, которое ставит перед игроком задачу разрушить состоящую из семи секций мишень до того, как истечет

заданный интервал времени, равный 10 секундам. Пользователь наводит на цель пушку путем касания экрана. После двойного тапа экрана пушка выстреливает пушечным ядром.

Вы научились определять строковые ресурсы String для представления форматированных строк, которые используются для вызова метода getString класса Resource и метода format класса String, а также задавать спецификаторы форматов для выполнения локализации. Было создано пользовательское представление путем расширения класса SurfaceView. Вы узнали о том, что имена классов пользовательских компонентов должны быть полностью квалифицированы в элементе разметки XML, который представляет компонент.

Мы рассмотрели дополнительные методы «жизненного цикла» класса Activity, освоили вызов метода onPause из текущего класса Activity в случае, если другая деятельность становится активной и вызывается метод onDestroy, когда система завершает деятельность.

С помощью переопределения метода onTouchEvent класса Activity мы обрабатывали касания и одиночные тапы. Для обработки двойных тапов, которые вызывают выстрел из пушки, использовался GestureDetector. Событие двойного тапа вызывало метод SimpleOnGestureListener, который содержится в переопределенном методе onDoubleTap.

Мы добавили звуковые эффекты в папку приложения res/raw и управляли ими с помощью класса SoundPool, а также использовали системную службу AudioManager для получения установленного в устройстве текущего значения громкости звучания музыки и использования этого значения для установки громкости воспроизведения звука.

Это приложение вручную выполняет анимации путем обновления элементов игры на SurfaceView из отдельного потока выполнения. Был расширен класс Thread и использован метод run, с помощью которого отображается графика с привлечением методов класса Canvas. Мы использовали метод SurfaceHolder класса SurfaceView для получения соответствующего класса Canvas, а также научились создавать циклы, управляющие игрой на основе количества времени, которое прошло между кадрами анимации. В результате игра выполняется с одной и той же скоростью на различных устройствах.

Список используемых источников

1. Дейтел П.. Android для программистов. Создаём приложения. / Пер. с англ. [Текст] / Дейтел П., Дейтел Х., Дейтел Э., Моргано М. — СПб.: Питер, 2013. — 560 с. — ISBN 978-5-459-01646-8
2. Дэрс Л.. Android за 24 часа. Программирование приложений под операционную систему Google (Профессиональные компьютерные книги). / Пер. с англ. [Текст] / Дэрс Л., Кондер Ш. — М.: Рид Групп, 2011. — 464 с. — ISBN 978-5-4252-0318-2
3. Кен Арнольд. Язык программирования Java. 3-е изд.). / Пер. с англ. [Текст] / Кен Арнольд, Джеймс Гослинг, Дэвид Холмс — М.: Издательский дом «Вильямс», 2001. — 624 с.: ил. ISBN 5-8459-0215-0 (рус.).
4. Хорстманн Кей С.. Java`2. Библиотека профессионала, том 1. Основы. 8-е издание) / Пер. с англ. [Текст] / Хорстманн Кей С., Корнелл Гари — М.: ООО «И.Д. Вильямс», 2008. — 816 с.: ил. ISBN 978-5-8459-1378-4 (рус.).
5. Гаврилов А.В. Учебное пособие по языку Java. Часть 1.) [Текст] / Гаврилов А.В., Дегтярёва О.А., Лёзин И.А., Лёзина И.В. — Самара: Издательство СНЦ РАН, 2010. — 176 с. - ISBN 978-5-93424-508-6
6. Ноутон П., Шилдт Г. Java2) / Пер. с англ. [Текст] / Ноутон П., Шилдт Г. СПб.: БХВ-Петербург, 2005.
7. Варакин М. Разработка мобильных приложений под Android. Курс 1. [Текст] / Варакин М. — М.: УЦ «Специалист» при МГТУ имени Н.Э.Баумана. 2012.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»
(СГАУ)

Солдатова О. П.

Технологии и инструментальные средства разработки программного обеспечения для мобильных устройств

Задания для лабораторных работ

Семестр 9

Самара

2013

Задания на лабораторные работы в 9 семестре

Разработка приложений под iOS

Лабораторная работа №1 (занятие №1)

1. Изучите инструментарий разработки по ссылке <http://applifto.ru/article/nachnite-razrabatyvat-ios-prilozheniya-segodnya>.
2. Напишите простейшее графическое приложение, следуя рекомендациям, приведенным по ссылке <http://applifto.ru/article/vashe-pervoe-ios-prilozhenie>.

Лабораторная работа №2 (занятие №2)

1. Модифицируйте предыдущее приложение таким образом, чтобы пользователь печатал в поле ввода некоторое арифметическое действие (например, «35 + 7» или «22 – 4»), а приложение по нажатию на кнопку проводило разбор данного предложения, преобразовывало введенные операнды в числа и выводило результат вычисления операции на экран.
2. Четыре арифметические операции реализуйте в отдельных классах, наследниках общего класса Operation. Проверяйте корректность ввода и обрабатывать исключения.

Лабораторная работа №3 (занятие №3)

Используя табличный графический интерфейс, напишите простейший калькулятор с поддержкой четырех арифметических операций.

Лабораторная работа №4 (занятия №4-5)

1. Создайте многостраничное приложение, пользуясь примером <http://macscripter.ru/xcode/sozdaem-pervoe-prilozhenie-pod-ios-dlya-iphone-i-ipod-touch.html>.
2. На каждой странице должен быть изображен ее номер. Анимация при перелистывании должна быть плавной.

Лабораторная работа №5 (занятия №6-8)

Напишите пошаговую сетевую игру согласно своему варианту:

1. Шашки.
2. Морской бой.
3. Шахматы.
4. Балда.
5. Крестики-нолики.
6. Реверси.
7. Дуэль.
8. Очко.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»
(СГАУ)

Солдатова О. П.

Технологии и инструментальные средства
разработки программного обеспечения для
мобильных устройств

Задания для лабораторных работ

Семестр А

Самара

2013

Лабораторная работа № 1. «Отслеживание состояний Активности».

1. В имеющемся проекте HelloAndroidWorld откройте в редакторе класс HelloAndroidWorld.java.

2. Переопределите методы onPause, onStart, onRestart для класса и внесите изменения в метод onCreate:

```
package com.example.helloandroidworld;
import android.app.Activity;
import android.os.Bundle;
import android.widget.Toast;
public class HelloAndroidWorld extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Toast.makeText(this, "onCreate()", Toast.LENGTH_LONG).show();
    }
    @Override
    protected void onPause() {
        Toast.makeText(this, "onPause()", Toast.LENGTH_LONG).show();
        super.onPause();
    }
    @Override
    protected void onRestart() {
        super.onRestart();
        Toast.makeText(this, "onRestart()", Toast.LENGTH_LONG).show();
    }
    @Override
    protected void onStart() {
        super.onStart();
        Toast.makeText(this, "onStart()", Toast.LENGTH_LONG).show();
    }
}
```

Обратите внимание, что в методах, восстанавливающих состояние, вызов метода родительского класса производится до ваших действий, а в методах, состоянии сохраняющих – после ваших действий.

3. Запустите приложение на исполнение в эмуляторе (Ctrl+F11, Android Project) и убедитесь, что при изменении состояния приложения HelloAndroidWorld на экране эмулятора появляются соответствующие уведомления типа Toast.

4. Поэкспериментируйте с приложением, чтобы выяснить, при каких условиях вызываются реализованные обработчики событий.

Лабораторная работа № 2. «Использование значений строк и цветов».

1. В имеющемся проекте HelloAndroidWorld создайте файл colors в каталоге res/values:

File → New → Android XML File:

2. Отредактируйте содержимое файла res/values/colors.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="view_bkg_color">#FF0</color>
    <color name="screen_bkg_color">#F88</color>
    <color name="text_color">#8004</color>
</resources>
```

3. Отредактируйте содержимое файла res/values/strings.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello, Android World!</string>
    <string name="app_name">HelloAndroidWorld</string>
</resources>
```

4. Внесите изменения в файл разметки res/layout/main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    android:background="@color/screen_bkg_color">
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"
        android:background="@color/view_bkg_color"
        android:textColor="@color/text_color"/>
</LinearLayout>
```

5. Сохраните все несохраненные файлы и запустите приложение:

6. Убедитесь, что внешний вид приложения изменился в соответствии с определенными нами ресурсами.

7. Поэкспериментируйте со значениями цветов, прозрачностью и HTML-тэгами в строковых константах для изменения внешнего вида приложения.

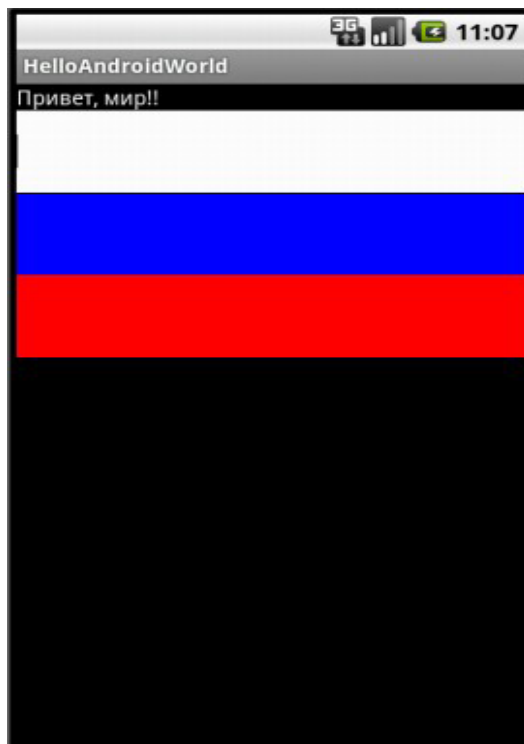
Лабораторная работа № 3. «Локализация приложения».

Реализовать класс адаптера, метод которого принимает в качестве параметра массив строк и записывает их по очереди в выходной байтовый поток (OutputStream), который он «адаптирует». Продемонстрировать работу в методе main().

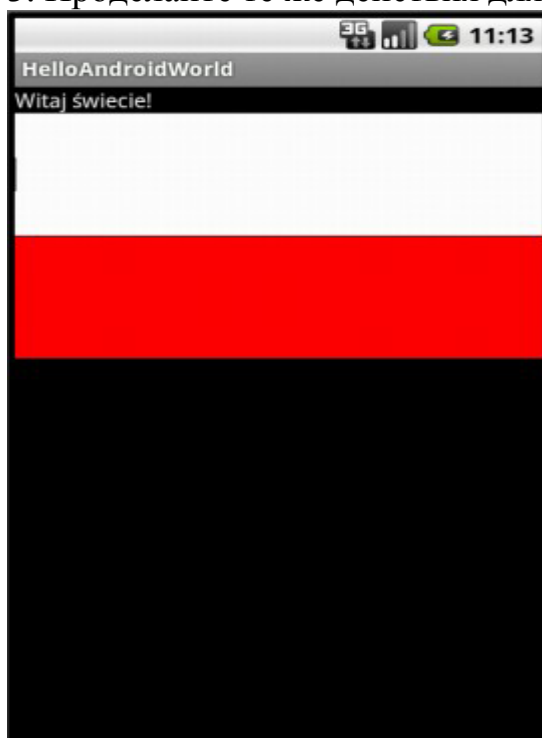
1. Измените ресурсы приложения HelloAndroidWorld так, чтобы оно выглядело следующим образом (на виртуальном устройстве должен быть установлен английский язык):



2. Добавьте нужные подкаталоги и ресурсы в каталог res проекта HelloAndroidWorld, чтобы при настройке русского языка приложение меняло свой вид на следующий:



3. Прodelайте те же действия для польского языка:



4. Восстановите языковые настройки на виртуальном устройстве.

Лабораторная работа № 4. «Использование анимации».

1. Создайте новый проект AnimSample.
2. В каталоге res создайте каталог anim, а в нем файл с именем ship_anim.xml, описывающий анимацию. Отредактируйте файл, чтобы он имел следующее содержимое:

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android" >
    <rotate
        android:duration="3333"
        android:fromDegrees="0"
        android:pivotX="50%"
        android:pivotY="50%"
        android:repeatCount="infinite"
        android:repeatMode="reverse"
        android:toDegrees="1080" />
    <translate
        android:duration="1900"
        android:fromXDelta="-50%p"
        android:repeatCount="infinite"
        android:repeatMode="reverse"
        android:toXDelta="50%p" />
    <translate
        android:duration="1300"
        android:fromYDelta="-50%p"
        android:repeatCount="infinite"
        android:repeatMode="reverse"
        android:startOffset="123"
        android:toYDelta="50%p" />
    <alpha
        android:duration="500"
        android:fromAlpha="1.0"
        android:repeatCount="infinite"
        android:repeatMode="reverse"
        android:toAlpha="0.3" />
    <scale
        android:duration="10000"
        android:fromXScale="0.0"
        android:fromYScale="0.0"
        android:pivotX="50%"
        android:pivotY="50%"
```

```
android:repeatCount="infinite"  
android:repeatMode="reverse"  
android:toXScale="2.5"  
android:toYScale="2.5" />
```

```
</set>
```

3. Добавьте рисунок, к которому будет применяться анимация (файл `lander_plain.png`), в каталог `res/drawable-mdpi`.

4. В файле разметки `res/layout/main.xml` замените элемент `TextView` на `ImageView` следующими атрибутами:

```
<ImageView  
    android:id="@+id/shipView"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"  
    android:src="@drawable/lander_plain" />
```

5. В конце метода `onCreate` Активности (она в этом проекте одна) добавьте следующие строки:

```
ImageView ship = (ImageView) findViewById(R.id.shipView);  
Animation shipAnim = AnimationUtils.loadAnimation(this,  
    R.anim.ship_anim);  
ship.startAnimation(shipAnim);
```

6. Запустите проект.

Лабораторная работа № 5. «Использование LinearLayout».

1. Продолжите локализацию приложения HelloAndroidWorld, теперь для французского языка. Для рисования флага используйте вложенный LinearLayout с вертикальной ориентацией дочерних элементов:



2. Обратите внимание, что при размещении элементов внутри вложенного LinearLayout удобно указывать атрибут `android:layout_weight="1"`, в этом случае дочерние виджеты будут размещены по горизонтали равномерно.

3. После получения нужного результата верните стандартные языковые настройки в виртуальном устройстве.

Лабораторная работа № 6. «Использование RelativeLayout».

RelativeLayout является очень полезным вариантом разметки, позволяющим создавать пользовательский интерфейс без избыточного количества вложенных элементов ViewGroup.

1. Создайте новый проект с именем RelativeLayoutSample.
2. Добавьте нужные строки в файл res/values/strings.xml и удалите строку с именем hello:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="label_text">Введите текст:</string>
    <string name="entry_hint">Поле ввода</string>
    <string name="app_name">RelativeLayoutSample</string>
</resources>
```

3. Отредактируйте файл разметки res/layout/main.xml так, чтобы он имел следующее содержимое:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >
    <TextView
        android:id="@+id/label"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/label_text" />
    <EditText
        android:id="@+id/entry"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/label"
        android:background="@android:drawable/editbox_background"
        android:hint="@string/entry_hint" />
    <Button
        android:id="@+id/ok"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_below="@id/entry"
        android:layout_marginLeft="10dip"
        android:text="@android:string/ok" />
    <Button
        android:layout_width="wrap_content"
```

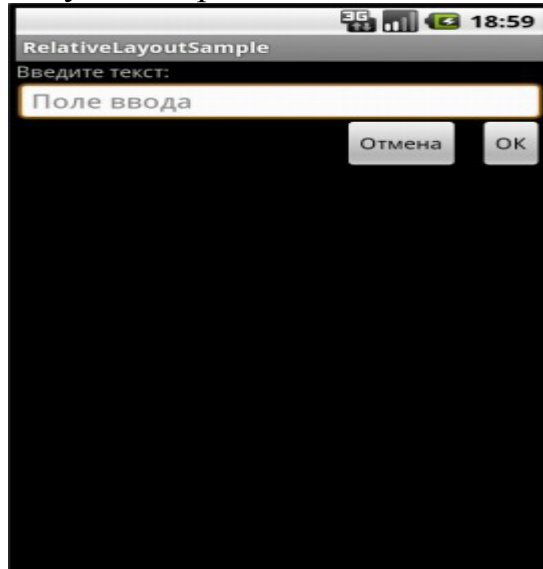
```

        android:layout_height="wrap_content"
        android:layout_alignTop="@id/ok"
        android:layout_toLeftOf="@id/ok"
        android:text="@android:string/cancel" />

```

</RelativeLayout>

4. Запустите приложение:



5. Отредактируйте файл AndroidManifest.xml, чтобы изменить тему, используемую приложением. Для это в узел <application> внесите следующие изменения:

...

```

<application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@android:style/Theme.Light" >

```

...

6. Запустите приложение с новой темой:



7. Мы использовали для текста кнопок в разметке стандартные строковые значения, которые предоставляет Android. Поэкспериментируйте с настройками языка в виртуальном устройстве и обратите внимание, как при запуске приложения меняются надписи на кнопках. Очевидно, что использование стандартных строковых значений позволяет минимизировать затраты на локализацию приложений.

Лабораторная работа № 7. «Использование TabWidget».

«Табулированная» разметка позволяет создавать UI, содержание вкладки. В этом случае используются такие виджеты, как TabHost и TabWidget.

TabHost является корневым узлом в разметке, содержащим TabWidget для отображения «вкладок», и FrameLayout для соответствующего им контента.

Отображение контента вкладок можно реализовать двумя способами:

- описав Представление (View) для содержимого каждой вкладки внутри одной и той же Активности;
- используя вкладки для переключения между различными Активностями.

Выбор конкретной реализации зависит от потребностей разработчика, но обычно более предпочтительным является второй вариант, так в этом случае разные вкладки обрабатываются разными Активностями, а не одной (достаточно громоздкой), что позволяет делать код более ясным и управляемым. В данной лабораторной работе мы используем вариант реализации с независимыми Активностями, поэтому для реализации трех вкладок нам потребуются четыре Активности (три для вкладок и одна «главная»).

1. Создайте новый проект с именем TabWidgetSample.

2. Опишите нужные строки в файле res/values/strings.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">TabWidgetSample</string>
    <string name="tab1_indicator">Students</string>
    <string name="tab2_indicator">Teachers</string>
    <string name="tab3_indicator">Classes</string>
    <string name="tab1_content">This is Students tab</string>
    <string name="tab2_content">This is Teachers tab</string>
    <string name="tab3_content">This is Classes tab</string>
</resources>
```

3. Отредактируйте файл res/layout/main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<TabHost xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@android:id/tabhost"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:padding="5dp">
```

```

        <TabWidget
            android:id="@android:id/tabs"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content" />
        <FrameLayout
            android:id="@android:id/tabcontent"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:padding="5dp" />
    </LinearLayout>
</TabHost>

```

4. Создайте три Активности с именами StudentsActivity, TeachersActivity и ClassesActivity.
5. В каждой из созданных Активностей переопределите метод onCreate следующим образом (внеся соответствующие изменения в имя ресурса tabX_content):

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Resources res = getResources();
    String contentText = res.getString(R.string.tab2_content);
    TextView textView = new TextView(this);
    textView.setText(contentText);
    setContentView(textView);
}

```

Обратите внимание на то, что в каждой Активности используются собственные строковые ресурсы.

6. Замените базовый класс для TabWidgetSampleActivity с Activity на TabActivity и переопределите метод onCreate следующим образом:

```

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    Resources res = getResources();
    String tab1Indicator = res.getString(R.string.tab1_indicator);
    String tab2Indicator = res.getString(R.string.tab2_indicator);
    String tab3Indicator = res.getString(R.string.tab3_indicator);
    TabHost tabHost = getTabHost();
    TabHost.TabSpec spec;
    Intent intent;
    intent = new Intent().setClass(this, StudentsActivity.class);
    spec = tabHost.newTabSpec("students").setIndicator(tab1Indicator)
        .setContent(intent);
    tabHost.addTab(spec);
}

```



```

        intent = new Intent().setClass(this, TeachersActivity.class);
        spec = tabHost.newTabSpec("teachers").setIndicator(tab2Indicator)
        .setContent(intent);
        tabHost.addTab(spec);
        intent = new Intent().setClass(this, ClassesActivity.class);
        spec = tabHost.newTabSpec("class").setIndicator(tab3Indicator)
        .setContent(intent);
        tabHost.addTab(spec);
        tabHost.setCurrentTab(1);
    }

```

В этом методе мы используем для запуска нужных Активностей так называемые явные Намерения (Intent). Намерения и их использование будут рассмотрены позже.

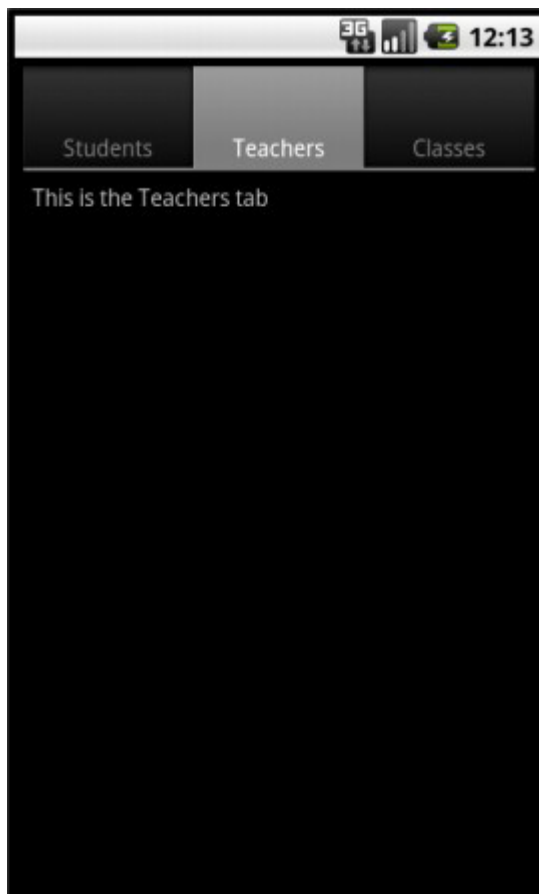
7. При попытке запуска проекта на выполнение произойдет аварийное завершение работы приложения, так как Активности, созданные для отображения контента вкладок неизвестны системе, потому что не описаны в Манифесте приложения. Требуется отредактировать файл AndroidManifest.xml так, чтобы все используемые Активности были в нем описаны. Кроме того, для улучшения внешнего вида интерфейса имеет смысл изменить используемую тему оформления на такую, где нет строки заголовка: Узел <application> должен выглядеть следующим образом:

```

<application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@android:style/Theme.NoTitleBar" >
    <activity
        android:name=".TabWidgetSampleActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category
                android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity android:name=".StudentsActivity" />
        <activity android:name=".TeachersActivity" />
        <activity android:name=".ClassesActivity" />
    </application>

```

8. Запустите приложение и поэкспериментируйте с вкладками:



9. Локализируйте приложение с помощью индивидуальных для разных языков строковых ресурсов.

Лабораторная работа № 8. «Использование WebView».

В данной лабораторной работе рассматривается использование виджета web-браузера и применяется итеративный подход к созданию приложения.

1. Создайте новый проект с именем WebViewSample.

2. Отредактируйте файл res/layout/main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<WebView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/webview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" />
```

3. Добавьте в конец метода onCreate Активности WebViewSampleActivity следующие строки:

```
WebView webView = (WebView) findViewById(R.id.webview);
webView.getSettings().setJavaScriptEnabled(true);
webView.loadUrl("http://www.ya.ru");
```

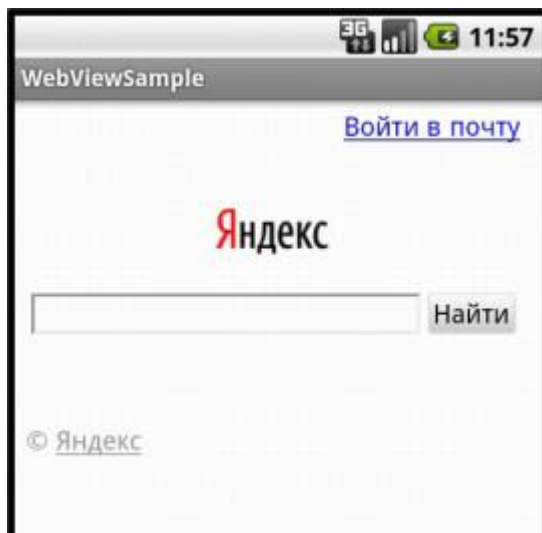
4. Запустите приложение:



5. Очевидно, что у приложения отсутствуют полномочия на доступ к сети. Добавим нужные полномочия в Манифест приложения, добавив элемент `<uses-permission>` внутри корневого узла `<manifest>`:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

6. Запустим проект (нижняя часть картинки отрезана):



7. Продолжим улучшать приложение. Для увеличения полезной площади экрана запретим показ заголовка, для этого укажем соответствующую тему в файле Манифеста:

```
<activity
    android:name=".WebViewSampleActivity"
    android:label="@string/app_name"
    android:theme="@android:style/Theme.NoTitleBar" >
```

8. Запустим проект и убедимся том, что (первая) цель достигнута.
9. В настоящий момент ссылки, ведущие за пределы сайта www.ya.ru обслуживаются стандартным web-браузером, а не нашим WebView, так как оно не в состоянии обработать эти запросы и виджет webView автоматически посылает системе соответствующее Намерение (Intent), обрабатываемое стандартным браузером. У этой проблемы есть два решения:

- Добавить в Манифест нужный Фильтр Намерений (Intent Filter)
- Переопределить внутри нашей Активности класс WebViewClient, чтобы приложение могло обрабатывать свои собственные запросы на отображение web-ресурсов с помощью имеющегося виджета WebView.

Второй вариант является более приемлемым еще и потому, что в этом случае не будет рассматриваться системой как альтернативный web-браузер, что произошло бы в случае добавления Фильтра Намерений в Манифест.

10. Вынесем объект webView из метода onCreate и сделаем его членом класса, после чего добавим внутри Активности новый класс WebViewSampleClient, расширяющий класс WebViewClient, после чего установим свой обработчик запросов на отображение web-ресурсов:

```
public class WebViewSampleActivity extends Activity {
    WebView webView;
    @Override
```

```

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    webView = (WebView) findViewById(R.id.webview);
    webView.getSettings().setJavaScriptEnabled(true);
    webView.loadUrl("http://www.ya.ru");
    webView.setWebViewClient(new WebViewSampleClient());
}
private class WebViewSampleClient extends WebViewClient {
    @Override
    public boolean shouldOverrideUrlLoading(WebView view, String
url) {
        view.loadUrl(url);
        return true;
    }
}
}

```

11. Запустим приложение и убедимся, что

12. Оставшийся серьезный недостаток – «неправильная» обработка нажатия кнопки «назад» устройства, приложение при этом заканчивает свою работу. Решение этой проблемы простое и прямолинейное: сделаем свой обработчик событий нажатия на кнопки, в котором будет реализовано только одно действие: если была нажата кнопка «назад», виджет WebView получит команду вернуться на предыдущую страницу (если у него есть такая возможность). Переопределим метод onKeyDown в Активности:

```

@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    if ((keyCode == KeyEvent.KEYCODE_BACK) &&
webView.canGoBack()) {
        webView.goBack();
        return true;
    }
    return super.onKeyDown(keyCode, event);
}

```

13. Запустим приложение и убедимся, что цель достигнута.

Лабораторная работа № 9. «Использование ListView».

1. Создайте новый Android проект ListViewSample.
2. В каталоге res/values создайте файл arrays.xml со следующим содержимым:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="stations">
        <item>Авиамоторная</item>
        <item>Автозаводская</item>
        <item>Академическая</item>
        <item>Александровский сад</item>
        <item>Алексеевская</item>
        <item>Алтуфьево</item>
        <item>Аннино</item>
        <item>Арбатская (Арбатско-Покровская линия)</item>
        <item>Арбатская (Филевская линия)</item>
        <item>Аэропорт</item>
        <item>Бабушкинская</item>
        <item>Багратионовская</item>
        <item>Баррикадная</item>
        <item>Бауманская</item>
        <item>Беговая</item>
        <item>Белорусская</item>
        <item>Беляево</item>
        <item>Бибирево</item>
        <item>Библиотека имени Ленина</item>
        <item>Битцевский парк</item>
        <item>Борисовская</item>
        <item>Боровицкая</item>
        <item>Ботанический сад</item>
        <item>Братиславская</item>
        <item>Бульвар адмирала Ушакова</item>
        <item>Бульвар Дмитрия Донского</item>
        <item>Бунинская аллея</item>
        <item>Варшавская</item>
        <item>ВДНХ</item>
        <item>Владыкино</item>
        <item>Водный стадион</item>
        <item>Войковская</item>
        <item>Волгоградский проспект</item>
        <item>Волжская</item>
```

<item>Волоколамская</item>
<item>Воробьевы горы</item>
<item>Выставочная</item>
<item>Выхино</item>
<item>Деловой центр</item>
<item>Динамо</item>
<item>Дмитровская</item>
<item>Добрынинская</item>
<item>Домодедовская</item>
<item>Достоевская</item>
<item>Дубровка</item>
<item>Жулебино</item>
<item>Зябликово</item>
<item>Измайловская</item>
<item>Калужская</item>
<item>Кантемировская</item>
<item>Каховская</item>
<item>Каширская</item>
<item>Киевская</item>
<item>Китай-город</item>
<item>Кожуховская</item>
<item>Коломенская</item>
<item>Комсомольская</item>
<item>Коньково</item>
<item>Красногвардейская</item>
<item>Краснопресненская</item>
<item>Красносельская</item>
<item>Красные ворота</item>
<item>Крестьянская застава</item>
<item>Кропоткинская</item>
<item>Крылатское</item>
<item>Кузнецкий мост</item>
<item>Кузьминки</item>
<item>Кунцевская</item>
<item>Курская</item>
<item>Кутузовская</item>
<item>Ленинский проспект</item>
<item>Лубянка</item>
<item>Люблино</item>
<item>Марксистская</item>
<item>Марьиная роща</item>
<item>Марьино</item>
<item>Маяковская</item>

<item>Медведково</item>
<item>Международная</item>
<item>Менделеевская</item>
<item>Митино</item>
<item>Молодежная</item>
<item>Мякинино</item>
<item>Нагатинская</item>
<item>Нагорная</item>
<item>Нахимовский проспект</item>
<item>Новогиреево</item>
<item>Новокузнецкая</item>
<item>Новослободская</item>
<item>Новаясеневская</item>
<item>Новые Черемушки</item>
<item>Октябрьская</item>
<item>Октябрьское поле</item>
<item>Орехово</item>
<item>Отрадное</item>
<item>Охотныйряд</item>
<item>Павелецкая</item>
<item>Парк культуры</item>
<item>Парк Победы</item>
<item>Партизанская</item>
<item>Первомайская</item>
<item>Перово</item>
<item>Петровско-Разумовская</item>
<item>Печатники</item>
<item>Пионерская</item>
<item>Планерная</item>
<item>Площадь Ильича</item>
<item>Площадь Революции</item>
<item>Полежаевская</item>
<item>Полянка</item>
<item>Пражская</item>
<item>Преображенская площадь</item>
<item>Пролетарская</item>
<item>Проспект Вернадского</item>
<item>Проспект Мира</item>
<item>Профсоюзная</item>
<item>Пушкинская</item>
<item>Речной вокзал</item>
<item>Рижская</item>
<item>Римская</item>

<item>Рязанский проспект</item>
<item>Савеловская</item>
<item>Свиблово</item>
<item>Севастопольская</item>
<item>Семеновская</item>
<item>Серпуховская</item>
<item>Славянский бульвар</item>
<item>Смоленская (Арбатско-Покровская линия)</item>
<item>Смоленская (Филевская линия)</item>
<item>Сокол</item>
<item>Сокольники</item>
<item>Спортивная</item>
<item>Сретенский бульвар</item>
<item>Строгино</item>
<item>Студенческая</item>
<item>Сухаревская</item>
<item>Сходненская</item>
<item>Таганская</item>
<item>Тверская</item>
<item>Театральная</item>
<item>Текстильщики</item>
<item>Теплый Стан</item>
<item>Тимирязевская</item>
<item>Третьяковская</item>
<item>Трубная</item>
<item>Тульская</item>
<item>Тургеневская</item>
<item>Тушинская</item>
<item>Улица 1905года</item>
<item>Улица Академика Янгеля</item>
<item>Улица Горчакова</item>
<item>Улица Подбельского</item>
<item>Улица Скобелевская</item>
<item>Улица Старокачаловская</item>
<item>Университет</item>
<item>Филевский парк</item>
<item>Фили</item>
<item>Фрунзенская</item>
<item>Царицыно</item>
<item>Цветной бульвар</item>
<item>Черкизовская</item>
<item>Чертановская</item>
</string-array>

</resources>

3. В каталоге res/layout создайте файл list_item.xml со следующим содержимым:

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="10dp"
    android:textSize="16sp" >
```

```
</TextView>
```

4. Модифицируйте метод onCreate вашей Активности:

```
@Override
```

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Resources r = getResources();
    String[] stationsArray = r.getStringArray(R.array.stations);
    ArrayAdapter<String> aa = new ArrayAdapter<String>(this,
        R.layout.list_item, stationsArray);
    setListAdapter(aa);
    ListView lv = getListView();
}
```

5. Измените базовый класс Активности с Activity на ListActivity.

6. Запустите приложение.

7. Для реакции на клики по элементам списка требуется добавить обработчик такого события, с помощью метода setOnItemClickListener. В качестве обработчика будет использоваться анонимный объект класса OnItemClickListener. Добавьте следующий код в нужное место:

```
lv.setOnItemClickListener(new OnItemClickListener() {
    public void onItemClick(AdapterView<?> parent, View v,
        int position, long id) {
        CharSequence text = ((TextView) v).getText();
        int duration = Toast.LENGTH_LONG;
        Context context = getApplicationContext();
        Toast.makeText(context, text, duration).show();
    }
});
```

8. Запустите приложение и «покликайте» по станциям метро.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»
(СГАУ)

Солдатова О. П.

Технологии и инструментальные средства разработки программного обеспечения для мобильных устройств

Вопросы к экзамену

Семестр 9

Самара

2013

Вопросы к экзамену в семестре 9 по курсу «Технологии и инструментальные средства разработки программного обеспечения для мобильных устройств».

1. Разработка приложений для мобильных устройств.
2. Платформы для разработки. Тестирование приложений.
3. Общие сведения о семействе ОС Mac OS. История ОС Mac OS.
4. Операционная система Mac OS X. Эволюция ОС Mac OS X.
5. Совместимость Mac OS X. Плюсы и минусы ОС Mac OS.
6. История iOS. Принципы пользовательского интерфейса iOS.
7. Характеристики платформы.
8. Разработка приложений под Mac OS X и iOS. Базовые элементы приложений.
9. Запуск приложения. Вывод на экран. Печать значений переменных.
10. Использование фреймворка Foundation. Компиляция и запуск программ.
11. Обзор среды разработки Xcode.
12. Основные понятия языка программирования Objective-C. Синтаксис языка.
13. Создание новых классов. Механизм сообщений. Протоколы.
14. Обработка исключений. Синхронизация. Создание и уничтожение объектов.
15. Категории. Объекты Class objects и Objective-C runtime.
16. Язык Objective-C 2.0.
17. Структура программы на языке Objective-C. Типы данных. Переменные и константы.
18. Массивы. Структуры. Указатели.
19. Основные операторы Objective-C. Операторы условия.
20. Циклы. Операторы Break и Continue.
21. Основы ООП. Жизненный цикл объекта.
22. Создание классов. Определение методов и свойств.
23. Указатели. Сообщения и методы. Создание объекта.
24. Протоколы. Категории. Потоки.
25. Исключения. Блоки. Наследование, инкапсуляция, полиморфизм.
26. Управление памятью в Cocoa.
27. Использование Autorelease пулов Cocoa.
28. Автоматический подсчет ссылок (ARC).
29. Характеристики строковых объектов в Cocoa. Характеристики строковых объектов.
30. Форматирование строковых объектов. Чтение и запись строк в файлы и URL.
31. Поиск, сравнение и сортировка строк. Поиск, сравнение и сортировка строк.
32. Символы и кластеры. Наборы символов.
33. Сканеры. Строковые представления файловых путей.

- 34.Отрисовка строк. Класс NSString. Класс NSMutableString.
- 35.Характеристики коллекций в Сосоа. Массивы: упорядоченные коллекции.
- 36.Словари: коллекции ключ-значение.
- 37.Наборы: неупорядоченные коллекции объектов. Индекс наборы: Хранение индексов в массиве.
- 38.Индекс пути: хранение пути через вложенные массивы. Копирование коллекций. Перечисление. Параметры указателя на функцию.
- 39.Паттерны iOS. Паттерн Prototype. Паттерн Factory Method. Паттерн Abstract Factory.
- 40.Паттерн Builder. Паттерн Singleton. Паттерн Adapter.
- 41.Паттерн Bridge. Паттерн Facade. Паттерн Mediator.
- 42.Паттерн Observer. Паттерн Composite. Паттерн Iterator.
- 43.Паттерн Visitor. Паттерн Decorator. Паттерн Chain of responsibility.
- 44.Паттерн Template Method. Паттерн Strategy. Паттерн Command.
- 45.Паттерн Flyweight. Паттерн Proxy. Паттерн Memento.
- 46.Архитектурный шаблон проектирования MVC.
- 47.Разработка приложений с графическим интерфейсом (GUI). Обзор Interface Builder.
- 48.Создание связей. Создание кода.
- 49.Поиск методов. Указатели.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»
(СГАУ)

Солдатова О. П.

Технологии и инструментальные средства разработки программного обеспечения для мобильных устройств

Вопросы к экзамену

Семестр А

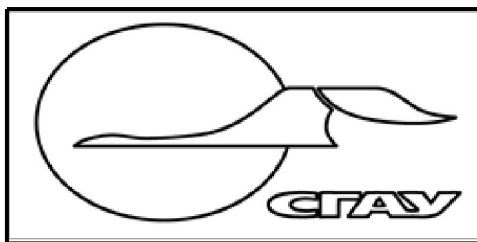
Самара

2013

Вопросы к экзамену в семестре А по курсу «Технологии и инструментальные средства разработки программного обеспечения для мобильных устройств».

1. История разработки платформы Android.
2. Архитектура платформы.
3. Новые функции Android 2.2, предназначенные для разработчиков.
4. Android 2.3 (Gingerbread).
5. Android 3.0 (Honeycomb).
6. Android Ice Cream Sandwich.
7. Загрузка приложений из Android Market. Пакеты.
8. Android Software Development Kit (SDK).
9. Тестирование приложения на виртуальном устройстве AVD.
10. Создание нового проекта в среде Eclipse.
11. Создание графического интерфейса пользователя приложения с помощью визуального макетного редактора модуля ADT.
12. Отладка приложений с помощью DDMS.
13. Основы языка Java. Классы и объекты.
14. Основы языка Java. Пакеты.
15. Основы языка Java. Понятие имени. Управление доступом.
16. Основы языка Java. Создание объектов. Конструкторы.
17. Основы языка Java. Блоки инициализации. Методы.
18. Основы языка Java. Типы данных.
19. Основы языка Java. Массивы.
20. Основы языка Java. Операторы.
21. Основы языка Java. Операторы условия.
22. Основы языка Java. Циклы.
23. Создание приложения Android с помощью Java.
24. Создание приложения с графическим интерфейсом.
25. Разработка приложения поисковых запросов в сетевых ресурсах.
26. Создание графического интерфейса приложения и файлов ресурсов.
27. Управление ресурсами приложения.
28. Конфигурирование файла манифеста ANDROID.
29. Проектирование каркаса игрового приложения ANDROID.
30. Реализация анимированного экрана-заставки.
31. Реализация экрана с основным меню.
32. Разработка экрана с инструкциями и экрана с результатами.
33. Разработка многопоточного игрового приложения.
34. Тестирование приложения. Обзор применяемых технологий.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
 ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
 ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
 «САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
 УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА
 (НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»
 (СГАУ)



СОГЛАСОВАНО

УТВЕРЖДАЮ

Управление образовательных программ

Проректор по учебной работе

_____ / А.В. Дорошин /

_____ / В.Н. Матвеев /

" ____ " _____ 20__ г.

" ____ " _____ 20__ г.

РАБОЧАЯ ПРОГРАММА ДИСЦИПЛИНЫ

Наименование модуля (дисциплины)

Технологии и инструментальные средства разработки ПО для мобильных устройств

Цикл, в рамках которого происходит освоение модуля (дисциплины)

М2 Профессиональный цикл

Часть цикла

М2.В.ОД.3 Вариативная часть

Код учебного плана

230100.68-1-2013-О-П-2г00м

Факультет

Информатики

Кафедра

Информационных систем и технологий

Курс

5

Семестр

9, А

Лекции (СЛ)

72

Семинарские и практические занятия (СП)

0

Лабораторные занятия (СЛР)

72

Экзамен

9, А

Контроль самостоятельной работы /
Индивидуальные занятия (КСР / ИЗ)

0

Зачет

Самостоятельная работа (СРС)

90

Всего (Всего с экзаменами)

306

Наименование стандарта, на основании которого составлена рабочая программа:

230100.68 "Информатика и вычислительная техника"

Соответствие содержания рабочей программы, условий ее реализации, материально-технической и учебно-методической обеспеченности учебного процесса по дисциплине всем требованиям государственных стандартов подтверждаем.

Составители:

А.Н.Калугин, к.ф.-м.н., О.П.Солдатова,
доцент, к.т.н.

(подпись)

Заведующий кафедрой:

С.А. Прохоров, профессор, д.т.н.

(подпись)

Рабочая программа обсуждена на заседании кафедры

Информационных систем и технологий

Протокол № ____ от " ____ " _____ 20 ____ г.

Наличие основной литературы в фондах научно-технической библиотеки (НТБ) подтверждаем:

Директор НТБ

(подпись)

/ _____ /
(расшифровка подписи)

Согласовано:

Декан

(подпись)

/ _____ /
(расшифровка подписи)

1 Цели и задачи модуля (дисциплины), требования к уровню освоения содержания

1.1 Перечень развиваемых компетенций

ОК-2, ОК-3. ОК-4, ОК-6;ПК-1, ПК-4, ПК-5

1.2 Цели и задачи изучения модуля (дисциплины)

Целями курса являются освоение технологий и инструментальных средств разработки ПО для мобильных устройств.

Задачи дисциплины: 1) изучение технологий разработки программного обеспечения для мобильных устройств под управлением операционных систем iOS и Android,
2) изучение инструментальных средств разработки ПО с использованием языков программирования Objective-C и Java,
3) освоение практических навыков реализации приложений для операционных систем iOS и Android,

1.3 Требования к уровню подготовки студента, завершившего изучение данного модуля (дисциплины)

По окончании изучения курса студенты должны
знать: технологии и средства разработки приложений для платформ iOS и Android;
уметь: грамотно разрабатывать приложения, в том числе с графическим интерфейсом, обработки данных, многостраничные и многопоточные приложения;
а также использовать для разработки ;
владеть: средствами разработки приложений на языках Objective-C и Java и навыками работы с системными службами, сетевые сервисами и стандартными контролами.

1.4 Связь с предшествующими модулями (дисциплинами)

Изложение материала базируется на знании студентами следующих дисциплин:

- объектно-ориентированное программирование;
- операционные системы;
- Математическая логика и теория алгоритмов.

1.5 Связь с последующими модулями (дисциплинами)

Дисциплина предшествует изучению курса "Технология разработки программного обеспечения" и является базовой для исследований в рамках НИР магистра и подготовки выпускной квалификационной работы магистра.

2 Содержание рабочей программы (модуля)

Семестр 1		

СЛ 0,2222 36 часов 0,9999 ЗЕТ	Активные 0	
	Интерактивные 0	
	Традиционные 1	Введение в разработку мобильных приложений. Обзор существующих мобильных платформ
		Mac OS X и iOS. История, архитектура. Основные парадигмы платформы.
		Платформа iOS. Средства разработки (XCode, iOS Simulator, Instruments), публикация и распространение приложений.
		Приложения для iOS. Жизненный цикл, особенности выполнения в фоне. Platform Characteristics и Human Interface Principles. Базовые элементы приложения
		Язык Objective-C. Структура программы на языке Objective C. Базовые конструкции языка (операторы и выражения, элементарные типы данных, литералы).
		Операторы и выражения. Ветвления и циклы, функции, сложные типы данных.
		Основы ООП. Жизненный цикл объекта. Указатели, сообщения и методы, управление памятью. Обзор базовых классов CoreFoundation, работа со строками, коллекциями
		Создание класса, определение методов и свойств, инициализаторы. Наследование, инкапсуляция и полиморфизм.
		Использование категорий и протоколов, блоки, исключения.
		Приложение с графическим интерфейсом для IOS. Введение в MVC. Проектирование View с использованием Interface Builder, связывание графических элементов с программной реализацией, Actions и Outlets, базовая реализация контроллера, жизненный цикл приложения
		Multitasking. Различные типы проектов в Xcode. Делегаты, Orientation Changes, Autolayout, View Properties (на примере UILabel и UIButton), Notifications, Особенности разработки для iPad

		Многостраничное приложение: storyboards, TabBarController, NavigationController, SplitViewController, контроллеры модальных представлений. Master-Detail, Storyboards, Memory Warnings.
		Использование различных стандартных контролов: UIButton, UITextEdit, UIImageView, Обработка ввода пользователя, UIImagePickerController, UIWebView, MapKit.
		UITableView, TableViewController, кастомизация cell. Делегаты, Orientation Changes, Autolayout, View Properties (на примере UILabel и UIButton), Notifications, Особенности разработки для iPad Делегаты, Orientation Changes, Autolayout, View Properties
		Кастомные кнопки, event handling, gesture recognizer, CoreAnimation.
		iOS: Постоянное хранение данных, работа с файлами, NSData, iCloud, NSUserDefaults, локализация
		iOS: сетевое взаимодействие.
		iOS: разработка многопоточных приложений, Grand Central Dispatch.
СП 0 0 часов 0 ЗЕТ	Активные 0	
	Интерактивные 0	
	Традиционные 0	
СЛР 0,2222 36 часов 0,9999 ЗЕТ	Активные 1	Инструментальная среда XCode, Базовые инструменты компиляции. Типы проектов. Создание графического приложения Hello World
		Разработка алгоритмов с линейной, циклической, ветвящейся структурой на языке Objective-C 2.0, определение и вызов функций, операторов, использование селекторов, использование блоков.

		Разработка иерархии классов с использованием базовых принципов ООП, особенности определения свойств, делегатов, использование категорий и протоколов, особенности управления памятью, обработка исключений.
		Разработка интерактивного приложения с графическим интерфейсом (табличный интерфейс).
		Разработка многостраничного приложения с графическим интерфейсом, использование анимаций.
		Разработка приложения использующего сетевые сервисы.
	Интерактивные 0	
	Традиционные 0	
КСР 0 0 часов 0 ЗЕТ	Активные 0	
	Интерактивные 1	
	Традиционные 0	
СРС 0,3333 54 часов 1,4998 ЗЕТ	Активные 0	
	Интерактивные 1	Подготовка к лабораторной работе №1.
		Подготовка к лабораторной работе №2.
		Подготовка к лабораторной работе №3.
		Подготовка к лабораторной работе №4.
		Подготовка к лабораторной работе №5.
		Подготовка к лабораторной работе №6.
		Подготовка к экзамену.
	Традиционные 0	
Семестр 2		
СЛ 0,25 36 часов 1 ЗЕТ	Активные 0	
	Интерактивные 0	
	Традиционные 1	Платформа Android. История и архитектура. Основные парадигмы платформы.
		Средства разработки, публикация и распространение приложений. Безопасность и права доступа. Обфускация.

		Приложения для Android. Жизненный цикл, базовые элементы, отличия от iOS приложений. Отличия в принципах организации пользовательского интерфейса.
		Язык Java: Операторы, функции, ООП. Особенности диалекта и функций стандартных библиотек для платформы Android.
		Android SDK. Обзор основных компонентов, принципы, паттерны и парадигмы.
		История развития SDK (версии 2.1 – 4.0).
		Разработка приложений с графическим интерфейсом: компоновка элементов, базовые элементы управления, командные элементы управления.
		Разработка приложений с графическим интерфейсом: отображение длительно выполняющихся задач, диалоговые окна, меню, Activity, фрагменты, работа с Action Bar.
		Разработка для устройств с различными разрешениями экрана. Вопросы локализации приложений.
		Ресурсы и обработка данных: взаимодействие с файловой системой, адаптеры данных и компоненты для отображения данных.
		Ресурсы и обработка данных: content provider, взаимодействие с базой данных SQLite, сохранение пользовательских настроек.
		Системные службы: компоненты Service, broadcast receiver.
		Управление домашним экраном, уведомлением в строке состояния, служба оповещений и буфер обмена, интен-ты.
		Использование сетевых сервисов (отправка SMS, управление Wi-Fi соединениями, определение местоположения и интеграция Google Maps).
		Создание анимаций. Создание нестандартных элементов управления.
		Создание виджетов.
		Разработка приложений с использованием NDK. Обзор, основные проблемы.

		Альтернативные способы разработки приложений для ОС Android и iPhone, разработка кросс-платформенных приложений. Обзор.
СП 0 0 часов 0 ЗЕТ	Активные 0	
	Интерактивные 0	
	Традиционные 0	
СЛР 0,25 36 часов 1 ЗЕТ	Активные 1	Инструментальная среда Eclipse, Базовые инструменты, ADT, эмулятор. Типы проектов. Создание графического приложения Welcome/Hello World.
		Разработка графического приложения использующего Shared Preferences, кнопки, вложенные структуры, интенеты, диалоговые окна, табличные элементы интерфейса.
		Разработка графического приложения. Обработка жестов и касаний.
		Разработка графического приложения, интегрированного с системными и сетевыми сервисами.
		Разработка собственных сервисов. Использование различных способов взаимодействия ui и сервисов. Activity launch modes
		Разработка анимаций и нестандартных элементов управления. Использование сторонних библиотек.
	Интерактивные 0	
	Традиционные 0	
КСР 0 0 часов 0 ЗЕТ	Активные 0	
	Интерактивные 0	
	Традиционные 0	
СРС 0,25 36 часов 1 ЗЕТ	Активные 0	
	Интерактивные 1	Подготовка к лабораторной работе №1.
		Подготовка к лабораторной работе №2.
		Подготовка к лабораторной работе №3.
		Подготовка к лабораторной работе №4.
		Подготовка к лабораторной работе №5.

		Подготовка к лабораторной работе №6.
		Подготовка к экзамену.
	Традиционные 0	

3 Инновационные методы обучения

1. Использование при самостоятельной подготовке электронных средств коммуникаций, в том числе специализированных сайтов и форумов.
2. Выполнение лабораторных работ с помощью современного программного обеспечения.
3. Использование тестирования для оценки знаний студентов.
4. Применение рейтинговой системы оценки знаний студентов.

4 Технические средства и материальное обеспечение учебного процесса

Лабораторные работы выполняются средствами вычислительной техники.
Требования к аппаратному обеспечению: наличие локальной сети и сети Internet.
Требования к программному обеспечению:
-операционные системы iOS и Android;

5 Учебно-методическое обеспечение

5.1 Основная литература

1. Гергель В. П. Высокопроизводительные вычисления для многопроцессорных многоядерных систем [Текст] : [учеб. для вузов по направлениям ВПО 010400 "Прикладная математика и информатика" и 010300 "Фундам. информатика и информ. технологии"] / В. П. Гергель ; Б-ка Нижегород. гос. ун-та им. Н. И. Лобачевского, [Суперкомпьютер. консорциум ун-тов России]. - М. : Изд-во Моск. ун-та [и др.], 2010. - 543 с. - (Серия Суперкомпьютерное образование : СКО). - ISBN 978-5-211-05937-5. - ISBN 978-5-9221-1312-0 : 220.00 р. -34 экз.
3. Хорстманн, Кей С. Java 2. Т. 1. Основы [Текст] / Кей С. Хорстманн, Гари Корнелл. - 8-е изд. -М.; СПб : Вильямс, 2012. - 813 с. - (Библиотека профессионала). - ISBN 978-5-8459-1378-4.- 50 экз.
4. Хорстманн, Кей С. Java 2. Т. 2. Тонкости программирования [Текст] / Кей С. Хорстманн, Гари Корнелл. - 8-е изд. - М.; СПб : Вильямс, 2012.— 983 с. – (Библиотека профессионала). - ISBN 978-5-8459-1482-8. - 50 экз.

5.2 Дополнительная литература

1. Брауде, Эрик Дж. Технология разработки программного обеспечения [Текст] : [перевод] / Эрик Дж. Брауде. - СПб. [и др.] : Питер : Питер принт, 2004. - 654 с. Экземпляров всего:7

5.3 Электронные источники и интернет ресурсы

Материалы сайтов <http://www.software-testing.ru> и <http://www.protesting.ru>

5.4 Методические указания и рекомендации

Текущий контроль знаний студентов в семестре завершается на отчетном занятии, результатом которого является допуск или недопуск студента к экзамену по дисциплине. Основанием для допуска к экзамену является выполнение всех лабораторных работ. Пропуск лекционных занятий может быть основанием для дополнительного вопроса на экзамене.

Промежуточный контроль знаний студентов проводят в семестре в виде экзамена. Экзамен проводится согласно положению о текущем и промежуточном контроле знаний студентов, утвержденному ректором университета. Оценка на экзамене ставится на основании письменного решения задачи в виде текста программы на языке Objective C или Java и устного ответа студента на вопрос из списка контрольных вопросов, а также, при необходимости, ответов на дополнительные вопросы. В качестве дополнительного задания может быть предложен как теоретический вопрос, так и задача.