Самоучитель

Денис Колисниченко

Программирование для Android 5

Android SDK 5.0 (API 21)

Выбор Android-устройства

Использование Android-эмулятора

Разработка интерфейса пользователя

Графика и анимация

Доступ к Интернету, отправка SMS

Взаимодействие с аппаратными средствами смартфона

СУБД SQLite

Платформа Titanium Mobile

Play Market: продвижение и продажа ваших приложений

bhv

Денис Колисниченко

Самоучитель Программирование для Android 5

Санкт-Петербург
«БХВ-Петербург»
2015

УДК 004.4
ББК 32.973.26-018.2
К60

Колисниченко Д. Н.

К60 Программирование для Android 5. Самоучитель. — СПб.: БХВ-Петербург, 2015. — 303 с.: ил.

ISBN 978-5-9775-3548-9

Рассмотрены все основные аспекты создания приложений для платформы Android 5 (API 21): установка необходимого программного обеспечения (JDK, Eclipse и Android SDK), использование эмулятора Android, создание интерфейса пользователя, работа с графикой, воспроизведение звука и видео, методы хранения данных (в том числе СУБД SQLite), взаимодействие с аппаратными средствами смартфона, платформа Titanium Mobile, публикация приложений на Play Market и отладка приложений.

Особое внимание уделено взаимодействию с аппаратными средствами смартфона. Показано, как получить информацию об устройстве и определить его состояние, использовать его датчики (акселерометр, датчик света, датчик температуры, датчик давления), камеру, Bluetooth-адаптер.

Приведены решения для различных нештатных ситуаций (отказ эмулятора, проблема с установкой программного обеспечения и т. д.), что поможет начинающему программисту.

Для программистов

УДК 004.4
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Екатерина Капалыгина*
Редактор	*Григорий Добин*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Марины Дамбиевой*

Подписано в печать 30.12.14.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 24,51.
Тираж 1200 экз. Заказ №
"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

Первая Академическая типография "Наука"
199034, Санкт-Петербург, 9 линия, 12/28

ISBN 978-5-9775-3548-9

Оглавление

Введение

> Я всегда мечтал о том, чтобы моим компьютером можно было пользоваться так же легко, как телефоном; моя мечта сбылась: я уже не могу разобраться, как пользоваться моим телефоном.
>
> *Бьёрн Страуструп (Bjarne Stroustrup)*

Современный мобильный телефон давно уже перестал быть просто телефоном. Помню свой первый мобильный телефон, который был приобретен в 2001 году: маленький монохромный экран (хотя сам телефон был размером с небольшой молоток), огромная клавиатура и колесико для управления меню. Телефон умел звонить, принимать звонки, а в качестве приятного бонуса имел возможность принимать и отправлять короткие сообщения — SMS. И это все.

Конечно, присутствовала поддержка WAP 1.1[1], но WAP-странички имелись даже не у всех сотовых операторов, не говоря уже об обычных компаниях. Понятно, что WAP-доступом практически никто не пользовался. Во-первых, это было весьма дорого. Во-вторых, как уже отмечалось, WAP-страниц существовало очень мало.

Подключить такой телефон к компьютеру возможность имелась (например, для работы в режиме факса), но загрузить дополнительные мелодии или выполнить какие-либо полезные и привычные сейчас действия оказывалось проблематично. О картах памяти и речи не шло, а для загрузки мелодии нужно было использовать особый дата-кабель и специальные программы. Эта операция не была описана в руководстве на устройство, и большинство пользователей даже не подозревало о такой возможности.

Мобильный телефон был именно мобильным телефоном: по нему можно было поговорить вне дома без необходимости таскать за собой многометровый телефонный кабель или подключать полутораметровую антенну, как на популярных в то время радиотелефонах Senao.

Сейчас мобильный телефон просто телефоном давно уже быть перестал. Скорее всего — это компьютер с функциями телефона. Да и называют «мобильники» те-

---

[1] WAP (Wireless Application Protocol) — беспроводной протокол передачи данных. Протокол создан специально для сетей GSM и позволял устанавливать связь портативных устройств (мобильный телефон, КПК, пейджеры, устройства двусторонней радиосвязи, смартфоны и другие терминалы) с сетью Интернет.

перь *смартфонами* — чтобы подчеркнуть тот факт, что в руках мы держим весьма «умное» устройство, а не просто какой-то там телефон. Смартфон отличается от мобильного телефона наличием полноценной развитой операционной системы, что облегчает жизнь как разработчикам приложений для мобильных телефонов, так и пользователям. Впрочем, в продаже можно встретить и обычные мобильные телефоны — как правило, это устройства из так называемой «бюджетной» категории.

Итак, вы приобрели современный смартфон. У вас теперь есть возможность загружать и устанавливать новые мелодии, вы можете использовать его в качестве медиапроигрывателя и органайзера, Интернет и почта тоже доступны в вашем мобильном устройстве. Кроме того, вы можете расширить его функциональность путем установки дополнительных приложений.

Стоп! Так всеми этими функциями обладают и обычные «продвинутые» телефоны, пришедшие в свое время на смену простым «говорилкам», но не носящие гордого названия «смартфон». В чем же разница? А в том, что при разработке мобильного приложения для «продвинутых» телефонов разработчику приходится как минимум учитывать требования производителя того или иного телефона, а обычно — и особенности его модели. Вспомните всевозможные сервисы загрузки на телефон приложений и другого контента: вы должны отправить SMS на определенный номер, но SMS не пустое, а содержащее некий код, позволяющий идентифицировать ваш телефон. Например, 1 — для Nokia, 2 — для Samsung и т. п. Как правило, после рекламы игры или другого какого-нибудь доступного для закачки приложения размещался целый список, включающий коды не только для телефонов разных производителей, но и для разных моделей одного и того же производителя. Пользователю легко было ошибиться — случайно отправить другой код вместо требуемого. В итоге с его счета снимались деньги, но приложение не работало. А разработчикам приходилось писать несколько версий одного и того же приложения, чтобы учесть особенности всех популярных моделей мобильных телефонов.

С появлением смартфонов все стало гораздо проще. Разработчики пишут приложения не под конкретный телефон, а под определенную мобильную операционную систему. А таких операционных систем гораздо меньше, чем моделей мобильных телефонов. Чаще всего теперь встречаются последние версии iOS (для смартфонов фирмы Apple), Windows Phone (для смартфонов серии Lumia) и Android. Есть еще предложения смартфонов на устаревающих системах Windows Mobile и Symbian OS, но их все меньше и меньше, поддержка этих ОС прекращена, и приложения для них разрабатывать смысла сейчас уже нет.

В настоящее время именно ОС Android является самой популярной мобильной операционной системой. Судите сами: если зайти в каталог, например, того же поисковика Яндекс.Маркет, где представлены смартфоны ведущих производителей (Acer, Fly, LG, Samsung, Sony, HTC и др.), то с ОС Android всех версий (от 1.5 до 5.0) вы найдете не менее полутора тысяч предложений, с Windows Phone — всего 50, а с iOS — не больше 20. Android устанавливается на смартфонах разного класса — от «бюджетных» моделей за 1,5–2 тыс. рублей до дорогих устройств стоимостью свыше 30 тыс. рублей.

Так что теперь разработчику достаточно написать для ОС Android всего лишь одно приложение, и оно будет работать на всех тех 1500 моделях смартфонов. Во всяком случае, почти на всех. И если вы написали приложение для платформы 4.0, то оно будет работать на смартфонах с Android 4.1, 4.2, 4.3, 4.4 и 5.0, поскольку Android обладает обратной совместимостью. Совсем другое дело, если вы желаете полностью использовать все возможности новой, 5-й версии Android, тогда, понятное дело, ваше приложение не будет работать на телефонах с более ранними версиями этой ОС.

Читателям книги «Программирование для Android. Самоучитель»

Прежде всего, разрешите выразить вам благодарность за то, что в свое время вы купили мою предыдущую книгу «Программирование для Android. Самоучитель». Эту же книгу, посвященную новейшей, 5-й версии Android, вы можете рассматривать как второе издание той, предыдущей, книги.

Что нового вы здесь найдете? В предыдущей книге, вышедшей в 2011 году, описывались версии Android 1.5–2.3. Отчасти немного была затронута версия 3.0. Эта книга ориентирована, как уже было сказано, на последнюю версию: 5.0. Android меняется, и много чего пришлось изменить и в книге: от описания процесса установки среды разработчика и новых типов разметки приложения до замены устаревших методов и т. п. Многие листинги пришлось переделать, адаптировав их под новую версию Android. На самом деле изменений больше, чем я предполагал, когда начинал работать над этим изданием. Описывать, что и в какой главе изменилось, я не стану — вы и сами скоро все увидите. А в качестве бонуса вас ждет описание второй версии среды App Inventor (см. *приложение*).

Как читать эту книгу?

Книга разделена на три части. В первой мы поговорим об установке необходимого программного обеспечения, напишем самое простое Android-приложение и научимся использовать эмулятор Android, позволяющий создавать приложения для Android без наличия физического устройства, что снижает стоимость разработки приложения.

Во второй части вы узнаете, как создать интерфейс пользователя для вашего приложения, как использовать в нем графику и звуки, мы также рассмотрим способы хранения данных.

Третья часть книги — самая сложная, поэтому даже не пытайтесь ее читать, не прочитав предварительно вторую часть. Вы узнаете, как создавать службы, потоки, широковещательные приемники, как использовать SQLite и подключаться к Интернету. Будет также рассмотрена подготовка вашего приложения для публикации на Play Маркет, взаимодействие с аппаратными средствами устройства и отладка приложения.

ЧАСТЬ I

Введение в Android

ГЛАВА 1

Знакомство с Android

1.1. Вкратце об Android

Раз вы держите в руках эту книгу, то, скорее всего, уже знаете, что такое Android. Да, Android — это операционная система для мобильных телефонов и других мобильных устройств (в последнее время становятся популярными планшеты на базе этой системы, существуют цифровые проигрыватели и даже нетбуки с Android на борту).

Изначально операционная система разрабатывалась компанией Android Inc., впоследствии купленной Google. Затем Google создала Open Handset Alliance — организацию, которая занимается поддержкой и развитием платформы.

Разработка приложений для Android осуществляется, как правило, на языке Java, но вы можете использовать и другие языки программирования — об этом чуть позже (в *главе 2*).

Первая версия Android появилась в сентябре 2008 года. С тех пор вышло девять версий системы и одна находится в разработке. Самая последняя версия на момент написания книги — 5.0. Если мы уж заговорили о версиях Android, то вы должны о них кое-что знать, прежде чем начнете программировать для Android. Несколько лет назад самой популярной была версия 2.*x*. Под ее управлением работали как бюджетные, так и дорогие модели смартфонов. Но в 2011 году стал набирать популярность другой класс устройств — планшеты (они существовали и раньше, просто не были так популярны, как сейчас). Вторая версия Android не очень подходила для планшетов — из-за их больших экранов (от 7 дюймов) и более высокого разрешения таких экранов. Поэтому в феврале 2011 года выпустили версию 3.*x*. Считалось, что если вы разрабатываете приложение для планшета, то вам нужно ориентироваться на Android 3.*x*, а если для смартфона, то — на версию 2.*x*. Однако третья версия, оказавшаяся во многом неудачной, популярность так и не завоевала, и уже в октябре 2011 года Google представила версию 4.0. Четвертая версия устанавливалась как на смартфоны (экраны которых уже достигли к тому времени размеров в 4–5 дюймов), так и на планшеты, поэтому разрабатывать приложения стало в некотором роде проще, поскольку оказались не нужны отдельные версии для смартфона и для планшета. Версия 4.*x* стала самой популярной, поскольку появившись

в 2011 году, она «продержалась» на рынке ровно три года — до октября 2014 года, когда вышла версия Android 5. При этом вторая версия, которая тоже была весьма популярной, продержалась на рынке менее двух лет: с октября 2009 года по февраль 2011 года, хотя устройства с версией 2.*x* выпускались вплоть до конца 2011 года и распродаются до сих пор. Та же участь ждет и версию 4.*x* — думаю, устройства с «четверкой» на борту будут продаваться на протяжении всего 2015 года.

В табл. 1.1 приведены даты выхода основных версий Android и их кодовые названия.

Таблица 1.1. *Основные версии Android*

Версия	Дата выхода	Кодовое название
1.0	21 октября 2008 года	Applebread
1.1	9 февраля 2009 года	Bender
1.5	30 апреля 2009 года	Cupcake
1.6	15 сентября 2009 года	Donut
2.0/2.1	26 октября 2009 года	Éclair
2.2	20 мая 2010 года	Froyo
2.3	7 декабря 2010 года	Gingerbread
3.0	3 февраля 2011 года	Honeycomb
3.1	10 мая 2011 года	Honeycomb
3.2	15 июля 2011 года	Honeycomb
4.0	19 октября 2011 года	Ice Cream Sandwich
4.1	27 июня 2012 года	Jelly Bean
4.2	29 октября 2012 года	Jelly Bean
4.3	25 июня 2013 года	Jelly Bean
4.4	31 октября 2013 года	KitKat
5.0	16 октября 2014 года	Lollipop

1.2. Выбор Android-устройства

Систему Android можно встретить на самых разных мобильных устройствах: мобильных телефонах (смартфонах), планшетах, цифровых проигрывателях, нетбуках. Впрочем, нетбуки с ОС Android на борту — явление достаточно редкое. Первым нетбуком с Android, добравшимся до российского рынка, стал AC100 от Toshiba. На этом нетбуке установлена только ОС Android. До него были и другие нетбуки — на некоторых моделях нетбуков от Acer Android устанавливалась как

вторая система при первой Windows. Пользователи могли ознакомиться с Android, попробовать, что это такое, но, как правило, работали все в Windows. Такая мультисистемность на нетбуках от Acer объясняется просто — компания не желала рисковать, ведь нетбуки с непривычной системой могли плохо продаваться. Компания Toshiba рискнула — выпустила нетбук с Android в качестве единственной системы. Могу сказать только одно: пока AC100 и подобные нетбуки, к сожалению, не очень распространены... Хотя сам нетбук получился очень ничего, но покупатели, в отличие от менеджеров Toshiba, рисковать не желают, поэтому предпочитают покупать устройства с проверенной ОС. Их тоже можно понять — многие покупают нетбук не просто как приятный гаджет, а для работы, каждый привык к своим любимым... Windows-приложениям. Вот этим и объясняется низкая популярность нетбуков с Android.

Первый Android-нетбук

Прочитать об AC100 можно по адресу:
http://hi-tech.mail.ru/review/misc/Toshiba_AC100-rev.html

Нетбуки iRU

В настоящее время вывести нетбуки с ОС Android на борту на российский рынок старается компания iRU. В продаже встречаются ее нетбуки серий K и W на Android 4.2.

Смартфоны и планшеты часто покупаются как приятные гаджеты. В большинстве случаев пользователям все равно, какая операционная система установлена на их смартфоне или планшете. Многие сначала обращают внимание на набор функций устройства, а уже затем на установленную операционную систему. Поэтому пользователи, заинтересовавшись обширным набором возможностей, сами того не ведая, покупают устройство с Android на борту, и только потом начинается знакомство с системой, попытка установки приложений и т. д.

Первым устройством, работающим под управлением Android, стал смартфон HTC T-Mobile G1, презентация которого состоялась 23 сентября 2008 года. Этот смартфон оказался настолько удачен, что вскоре другие производители мобильных телефонов заявили о намерении выпускать устройства с этой системой.

Что влияет на стоимость устройства? В первую очередь — бренд. Если вы следите за ценами на рынке мобильных устройств, то уже заметили, что устройства от LG и Samsung оказываются, как правило, дешевле таких же по характеристикам устройств, но других производителей (HTC, Sony). Не говоря уже о менее именитых брендах типа Fly или Prestigio.

Во-вторых, на цену того или иного устройства влияет набор его характеристик как мобильного устройства: размер и качество экрана, наличие Wi-Fi, объем встроенной памяти, производительность процессора, характеристики встроенных камер (чем они лучше, тем дороже телефон).

Сама операционная система Android, по сути, на стоимость устройства не влияет. Скорее всего, наоборот — характеристики устройства влияют на установленную версию Android. Так, выбор версии операционной системы зависит в основном от предпочтений производителя и характеристик устройства. Например, для работы

версии 4.4 необходимо, как минимум, 512 Мбайт оперативной памяти, а рекомендуется — 1 Гбайт. Следовательно, на слабые устройства версии 4.4 и 5.0 не устанавливаются. Если вы покупаете недорогое устройство, то оно будет работать под управлением, как правило, версии 4.1, 4.2 или 4.3. Версия 4.0 уже морально устарела и не используется, а самые современные версии (4.4 и 5.0) требуют более мощного устройства.

В предыдущей своей книге «Программирование для Android. Самоучитель» я приводил основные технические характеристики многих моделей, которые были в то время актуальными на рынке. Однако поскольку всевозможных моделей стало сейчас очень много, описывать все имеющиеся модели просто нет смысла. Пока книга выйдет из типографии, появится десяток новых моделей.

При выборе Android-устройства (независимо от его типа: планшет или смартфон) нужно обратить внимание на следующие основные характеристики: тип процессора, объем оперативной памяти (объем встроенной памяти обычно значения не имеет, т. к. почти во всех моделях предусмотрена возможность установить microSD-карту объемом 32–64 Гбайт[1]), размер и тип экрана и тип видеоускорителя.

1.2.1. Процессор

Общие сведения

Первым делом нужно обратить свое внимание на процессор. Тут все просто: чем мощнее процессор (выше его тактовая частота и больше ядер), тем быстрее будет работать устройство и меньше раздражать вас своей нерасторопностью. Однако есть и другая прямая зависимость: чем мощнее процессор, тем дороже устройство. Но обратной зависимости нет — наивно полагать, что чем дороже устройство, тем мощнее в нем процессор. В качестве примера можно привести два смартфона: Acer Liquid E2 Duo и LG Optimus L7 II Dual P715. Второй стоит дороже, но процессор мощнее в первом. Разница в цене не очень велика и, скорее всего, при выборе одной из этих моделей вы будете руководствоваться другими характеристиками, — например, у Acer Liquid E2 Duo есть поддержка двух SIM-карт, есть две камеры (основная и неплохая фронтальная) и немного больше дисплей. Остальные параметры примерно одинаковые. Можно привести и еще один пример: Samsung Galaxy S III. Он стоит существенно дороже, чем Acer Liquid E2 Duo, а процессор у него вообще на 1,0 ГГц.

Итак, при выборе процессора обратите внимание на число ядер и рабочую частоту. Устройства начального уровня имеют два ядра и частоту 1,0 ГГц. Устройства чуть дороже (но не намного) — 1,2 ГГц и те же два ядра. Более дорогие устройства оснащены четырехъядерным процессором с частотой 1,5–2,2 ГГц. Такая производительность достойна даже ноутбука начального уровня. Например, у Sony Xperia Z

[1] Тем не менее, намечающаяся сейчас тенденция в угоду все меньшей толщине устройства не устанавливать в него слот для внешней карты памяти не может не беспокоить, и приобретая такие устройства следует иметь в виду невозможность расширения их памяти.

Ultra C6802 частота процессора как раз 2,2 ГГц — именно как у ноутбука начального уровня! Правда, стоит этот смартфон, как *два* таких ноутбука...

И еще один момент, ставший актуальным в самое последнее время. Если вы хотите иметь возможность подключаться со своего устройства к скоростному мобильному Интернету 4-го поколения (4G) по технологии LTE, процессор вашего устройства должен такую возможность обеспечивать.

Подробнее об ARM-процессорах

Если такое объяснение вас устроило, тогда можете смело переходить к следующему подразделу, а самым любопытным читателям предлагаю «копнуть» глубже. Без понимания сути вы можете судить о производительности процессора только по его косвенным признакам — например, по рабочей частоте. Но это не совсем правильно.

Итак, процессоры всех Android-устройств, будь то смартфон, будь то планшет, основаны на архитектуре ARM (Advanced RISC Machine). Вы будете поражены, но это более продвинутая архитектура, чем архитектура CISC (Complex Instruction Set Computer), на базе которой создано большинство процессоров привычных нам компьютеров. Разница между этими архитектурами в наборе команд и в самом подходе к построению этих команд. У архитектуры CISC более сложные команды. У архитектуры RISC (Reduced Instruction Set Computing) команды более простые. И то, что в CISC делается одной командой, в RISC делается несколькими. CISC более похожа на мышление человека, а RISC — на «мысли» компьютера. Приведу пример. Вы просите кого-то включить свет. Это пример CISC-команды: «включи свет». Чтобы добиться того же результата в RISC, нужен набор команд: «встань, подойди к выключателю, включи его». В общем, что-то вроде этого. Конечно, я сильно все упростил, но это не та книга, чтобы все усложнять.

Далее мы рассмотрим популярные ARM-процессоры, которые устанавливаются на современные смартфоны и планшеты.

Архитектура ARM Cortex-A5

Архитектура процессора, используемая на устройствах начального уровня. Архитектура довольно старая, и если сейчас вы видите в продаже устройство на базе Cortex-A5, то перед вами или очень устаревшая модель, или же самая дешевая модель китайского производителя вроде Prestigio. Я не рекомендую покупать устройство на базе этой архитектуры, хотя она совместима с более современными Cortex-A8/A9, что позволяет запускать на таких устройствах современные версии Android.

Архитектура ARM Cortex-A7/Cortex-A8

На бюджетные модели устанавливается одноядерный процессор ARM Cortex-A8 или его модификации вроде Boxchip A13. Рабочая частота Cortex-A8 — от 600 МГц до 1 ГГц (хотя модификации типа A13 могут работать на более высоких частотах — до 1,2 ГГц).

Интересно, что Cortex-A7 лучше по своим характеристикам, чем Cortex-A8, несмотря на «семерку» в наименовании архитектуры. При этом Cortex-A8 менее

распространена, чем Cortex-A7, поэтому в большинстве случаев вы будете наблюдать модели гаджетов на базе Cortext-A7. Здесь рабочая частота может варьироваться от 600 МГц до 3 ГГц. Однако, как правило, частота современных процессоров на базе Cortex-A7 находится в пределах 1 ГГц. Серьезным преимуществом этой архитектуры является поддержка многоядерных конфигураций. Теоретически, на базе этой архитектуры можно построить 8-ядерный процессор, т. к. допускается два кластера на кристалл, а в одном кластере может быть от 1 до 4 ядер. Но таких «монстров» на базе Cortex-A7 я не встречал, поскольку производители предпочитают использовать более совершенную архитектуру: Cortex-A9.

Архитектура ARM Cortex-A9

Процессоры семейства на базе архитектуры ARM Cortex-A9 гораздо лучше своих предшественников. Они, как правило, многоядерные — могут содержать до четырех ядер. Двухъядерная версия Cortex-A9 обычно работает на частоте 1,2 ГГц. Максимальная частота для Cortex-A9 — до 3 ГГц, минимальная — 800 МГц.

Некоторые планшеты поставляются с процессорами Rockchip 29*xx*. Эти процессоры — модифицированная версия Cortex-A8, в то время как Rockchip 3*xxx* — это модифицированная версия Cortex-A9. Ясно, что лучше предпочесть Rockchip 3*xxx*.

Процессор Amlogic 8726-M6 — это модифицированный двухъядерный Cortex-A9.

Есть еще она «темная» лошадка — процессоры MediaTek, в частности, MTK8317T, который устанавливается на довольно популярный (судя по продажам) планшет Acer Iconia Tab. На базе процессоров тайваньской фирмы MediaTek построено множество мобильных устройств, о линейке этих процессоров вы можете подробно прочитать здесь: **http://ru.wikipedia.org/wiki/MediaTek**. Так вот, процессор MTK8317T является ничем иным, как Cortex-A9. Процессор двухъядерный, рабочая частота — 1,2 ГГц. То есть вполне нормальный процессор на сегодняшний день.

И еще один процессор, который следует упомянуть, — Action ATM7029. Это четырехъядерный (!) процессор на архитектуре ARM Cortex-A9 с частотой до 1,5 ГГц.

Архитектура Cortex-A15

Cortex-A15 — самая совершенная на сегодняшний день Cortex-архитектура. Процессоры на базе Cortex-A15 могут содержать до четырех ядер, а рабочая частота может варьироваться от 800 МГц до 3 ГГц. Если судить по количеству ядер и частоте, то эта архитектура недалеко ушла от Cortex-A9, но вся суть в вычислительном конвейере, позволяющем выполнять 3 команды за такт, в то время как аналогичный конвейер в Cortex-A9 выполняет только две команды за такт. Ядро Cortex-A15 обрабатывает до восьми микроопераций за такт против четырех микроопераций у Cortex-A9. Это означает, что при одинаковом количестве ядер и одинаковой частоте процессор на Cortex-A15 будет работать в среднем в 2 раза быстрее, чем на Cortex-A9. Да и кэш второго уровня L2 здесь 8 Мбайт, а не 4 Мбайт, как у Cortex-A9. Размер кэша первого уровня (L1) у этих двух архитектур одинаковый (32 Кбайт), но у A15 используется 128-битная шина кэша, а у A9 — только 64-битная.

Процессоры VIA и Samsung Exynos

Нельзя не упомянуть и процессоры VIA. Компания VIA известна своими процессорами и для настольных компьютеров, и для ноутбуков. На планшеты часто устанавливают процессор VIA WM8950, который не что иное, как тот же Cortex-A9, но количество ядер удручает, — это одноядерный процессор. Так что, если вы увидели в описании устройства заветную архитектуру Cortex-A9, не спешите с выводами — посмотрите на число ядер. Нужно отметить, что этот процессор часто устанавливается на планшеты ASUS, так что будьте с ними осторожны. ASUS — хороший производитель, но одно ядро на сегодняшний день уже не актуально.

Зато на самом доступном планшете Samsung установлен двухъядерный процессор, хотя и с частотой 1 ГГц. На более дорогих планшетах от Samsung устанавливаются процессоры Samsung Exynos 4212 и даже Intel Atom Z2560. Процессоры Exynos — это собственная разработка Samsung на базе архитектур ARM Cortex-A8, ARM Cortex-A9 и ARM Cortex-A15 (табл. 1.2).

Таблица 1.2. *Процессоры Samsung Exynos*

Процессор	К-во ядер	Архитектура	Частота, ГГц
Exynos 3	1	Cortex-A8	0,8 – 1,2
Exynos 4	2 или 4	Cortex-A9	1,2 – 1,6
Exynos 5	2 или 4	Cortex-A15	до 1,7

Процессоры от Intel

Процессор Intel Atom Z2560 — двухъядерный процессор с частотой 1,6 ГГц. И это не ARM-процессор, а самый обычный процессор архитектуры *x*86, который можно встретить на нетбуках и некоторых дешевых настольных системах. Для нормальной настольной системы такого процессора будет маловато, а для планшета — в самый раз.

Кроме процессора Intel Atom Z2560 на планшетах Samsung можно встретить также более мощные процессоры Intel Atom Z2760 с частотой 1,8 ГГц.

Выводы

Теперь давайте подытожим. От процессоров Cortex-A8 рекомендую отказаться. Они слабые. Частота, как правило, не превышает 1 ГГц и всего одно ядро. Когда речь идет о Cortex-A9, обычно число ядер 2 или 4, но VIA умудрилась выпустить одноядерный процессор. Поэтому перед покупкой устройства нужно точно знать, какой в нем установлен процессор. Опять-таки, цена конечного устройства с одноядерным устройством может быть даже выше, чем устройства с двухъядерным процессором.

Таблица 1.3 содержит сводные характеристики упомянутых в этой главе процессоров.

Таблица 1.3. *Некоторые ARM-процессоры*

Процессор	К-во ядер	Архитектура	Частота, ГГц
Boxchip A13	1	Cortex-A8	1.0
Allwinner A1x	1	Cortex-A8	1-1.5
Rockchip 29xx	1	Cortex-A8	до 1.0
Rockchip 3xxx	2 или 4	Cortex-A9	до 3.0
Amlogic 8726-M6	2	Cortex-A9	1.5
MediaTek MTK8317T	2	Cortex-A9	1.2
Action ATM7029	4	Cortex-A9	1.5
VIA WM8950	1	Cortex-A9	1.0
Intel Atom Z2560	2	x86	1.6
Intel Atom Z2760	2	x86	1.8

1.2.2. Память

Оперативная память используется для хранения данных, обрабатываемых в текущий момент. Android не очень требовательная к «оперативке» система, просто в последнее время сами данные слишком объемны. У смартфонов и планшетов начального уровня объем оперативной памяти равен 512 Мбайт. Помните, что 512 Мбайт — это необходимый минимум для Android 4.4 и 5.0. Хотя производители и не устанавливают Android 4.4 на бюджетные устройства, но если вам вдруг такое встретится, то лучше обойти его стороной, — рекомендуемый объем памяти для Android 4.4 — 1 Гбайт.

Так что, учитывая, что в большинстве случаев вы не можете модернизировать оперативную память смартфона/планшета (это не ноутбук, где можно легко и быстро расширить ОЗУ), и что обрабатываемые данные меньше уже не станут, я бы в любом случае присмотрелся к моделям, где установлено не менее 1 Гбайт ОЗУ. Ну, или хотя бы 768 Мбайт (есть такие модели смартфонов у HTC, планшетов с 768 Мбайт я не встречал). Конечно, покупать так покупать — если позволяют финансы, можно купить устройство с 2 Гбайт ОЗУ, вот только назвать бюджетными эти модели язык не поворачивается.

С объемом встроенной памяти все ясно и просто: чем ее больше, тем больше данных вы сможете хранить до покупки microSD-карты. В большинстве случаев Android-устройства поддерживают microSD-карты большого размера (до 32 Гбайт). В общем-то, особой разницы нет, сколько встроенной памяти установлено в выбранной вами модели смартфона/планшета — всегда можно купить microSD-карту (разумеется, если устройство имеет слот для их установки, на наличие которого тоже следует обращать повышенное внимание). У бюджетных моделей смартфонов обычно 4 Гбайт встроенной памяти, у моделей подороже — 8 Гбайт, а у самых дорогих моделей — 16 Гбайт. Планшеты начального уровня обычно поставляются

с 8 Гбайт встроенной памяти, а самые дорогие оснащены SSD-дисками, емкость которых может достигать 256 Гбайт.

1.2.3. Дисплей

Размер дисплея приобретаемого устройства зависит от личных предпочтений. Мне вот не очень нравятся модели с очень большим (5 дюймов и более) дисплеем, но на вкус и цвет, как говорится... Зато тип экрана играет большую роль:

- TFT — общее название дисплеев, обычно под ним скрывается TN-матрица: углы обзора и цветопередача будут не на высоте;
- IPS — еще недавно этот тип дисплея считался слишком дорогим, а сейчас устанавливается уже и не в самые дорогие устройства, что не может не радовать. Отличная цветопередача, глубокий цвет, широкие углы обзора. TN/TFT-матрицу я не могу порекомендовать, а вот IPS — смело можете покупать (или ее усовершенствованную модификацию — PLS);
- SuperLCD — разработка Sony, они потребляют меньше энергии, чем обычные LCD, и также отображают яркую и четкую картинку;
- AMOLED — фирменная светодиодная технология Samsung. Такие экраны считаются одними из лучших, но на данный момент дорогие и сложные в производстве. Используются только на лучших смартфонах Samsung — начиная с Galaxy S III;
- NOVA — основана на технологии IPS, разработка LG. Обеспечивает высокую яркость и экономию заряда батареи.

1.2.4. Видеоускоритель

При выборе устройства следует обратить внимание не только на процессор, но и на видеоускоритель, о чем упоминалось ранее. Понятное дело, что чем лучше ускоритель, тем лучшие игры можно запустить на устройстве, где он установлен. Есть и такая зависимость: чем лучше процессор устройства, тем лучше видеоускоритель. Поэтому, даже если в характеристиках устройства не указано, какой ускоритель в нем установлен, о видеоускорителе можно судить по установленному процессору. Например, ускоритель Adreno 200 не может быть установлен с современными процессорами вроде S4 Pro. Еще год назад лучшей связкой считалось устройство на базе процессоров MSM7230/MSM8255 и ускорителя Adreno 205. Но за этот год много чего поменялось, и ускоритель Adreno 205 — уже далеко не идеальный вариант. Сейчас неплохие варианты представляют собой связки Adreno 320/330 и процессоры Snapdragon 600/800 соответственно. Таблица 1.4 поможет вам определить, какой видеоускоритель установлен в том или ином устройстве.

Платформа Snapdragon является наиболее сбалансированной. Устройства, построенные на этой платформе, можно одинаково эффективно использовать как для работы, так и для развлечений.

Впрочем, ускорители Adreno уступают ускорителям PowerVR, но это уже другая песня. Устройства на ускорителях PowerVR строятся больше для игр, нежели для

Таблица 1.4. *Видеоускорители и процессоры*

Видеоускоритель	Используется с процессорами
Adreno 200	MSM7x27, QSD8x50, Freescale i.MX51, i.MX53, MSM7x25A, MSM7x27A
Adreno 203	S4 Play, Snapdragon 200 (Cortex-A5: 8225Q, 8625Q)
Adreno 205	MSM7x30, MSM8x55, APQ8055
Adreno 220	APQ8060, MSM8x60
Adreno 225	APQ8060A, MSM8x60A, MSM8960
Adreno 230	MSM8225, MSM8625
Adreno 302	Snapdragon 200 (Cortex-A7: 8210, 8610, 8212, 8612)
Adreno 305	MSM8x2x, MSM8x3x, APQ8030, Snapdragon 400
Adreno 320	S4 Pro, S4 Prime Snapdragon 600
Adreno 330	Snapdragon 800
Adreno 420	Snapdragon 805

работы. И если поиграть получится, то для работы такие устройства покупать не рационально.

Устройства с видеоускорителем PowerVR не так многочисленны на рынке, как устройства с ускорителем Adreno, и с полным списком таких устройств можно ознакомиться на этой страничке (здесь приводятся не только мобильные телефоны): **http://en.wikipedia.org/wiki/List_of_PowerVR_products**.

То, что ускорители PowerVR установлены на iPhone и iPod Touch Apple, говорит о многом. Продукция от Apple всегда славилась своей графикой. И отменное качество гарантирует здесь именно видеоускоритель PowerVR.

Высокое качество графики видеоускорителей PowerVR послужило предпосылкой для появления их и на некоторых Android-устройствах. Поскольку видеоускорители на этих устройствах и на iPhone одинаковые, то многие игры, ранее написанные для iOS, были портированы на Android. Одним словом, если у вас раньше был iPhone, и вы играли на нем в любимую игрушку, теперь она доступна и на платформе Android.

Вообще же, самыми актуальными сегодня являются ускорители Adreno 420 и PowerVR GX6650. Если сравнивать их между собой, то лучший выбор — PowerVR, у него еще есть запас мощности, которого уже не осталось у Adreno. Да и по API ускорители PowerVR выглядят лучше: GX6650 поддерживает OpenGL версии 3.2, а Adreno 420 — только 3.1.

Но и ускорители PowerVR — это далеко не предел! Все было бы хорошо и спокойно в мире Android, все бы пользовались ускорителями Adreno и PowerVR, но это спокойствие было нарушено компанией NVIDIA. Думаю, не нужно объяснять, что это за компания и на чем она специализируется. Ее платформа NVIDIA Tegra 3/4 буквально взорвала мир игр Android. Графика, которую способна передавать

NVIDIA Tegra 3/4, сопоставима с графикой персональных компьютеров, только на экране небольшого смартфона. Подробно об этой платформе написано по этой ссылке: **http://www.nvidia.ru/object/tegra-ru.html**.

Говорить о платформе NVIDIA Tegra можно очень долго, но пока лучше этой платформы никто ничего не придумал. Чтобы просмотреть, на что она способна, посетите страничку на YouTube: **http://www.youtube.com/watch?v=YhA0cbu1BxI**.

Вот примеры устройств, на которых используется платформа Tegra 3/4 (в списке не приведены модели на базе Tegra 2, которая уже считается устаревшей):

- HTC One X/One X+ (Tegra 3);
- LG Optimus 4X (Tegra 3);
- Xiaomi Mi3 (Tegra 4);
- LG Optimus Vu (Tegra 3).

Теперь подытожим. Что же выбрать из всего этого многообразия? Относительное дешевое решение на базе Adreno, более дорогое устройство с PowerVR или же перейти в высшую лигу и купить устройство на базе Tegra?

Все зависит от того, графику какого класса вы намереваетесь создавать. Для сложных игр лучше подойдет Tegra или PowerVR. А для несложной графики вполне будет достаточно и Adreno.

1.3. Архитектура Android

Архитектура Android состоит из четырех уровней: уровень ядра, уровень библиотек и среды выполнения, уровень каркаса приложений (application framework) и уровень приложений. Начнем с ядра.

Система Android основана на ядре Linux версии 2.6. Тем не менее, Android не является Linux-системой в прямом смысле этого слова. У Android свои механизмы распределения памяти, другая система межпроцессного взаимодействия (Inter-Process Communication, IPC), специфические модули ядра и т. д. На уровне ядра происходит управление аппаратными средствами мобильного устройства. На этом уровне работают драйверы дисплея, камеры, клавиатуры, Wi-Fi, аудиодрайверы. Особое место занимают драйверы управления питанием и драйвер межпроцессного взаимодействия (IPC).

Уровень ядра — самый низкий уровень архитектуры Android. Следующий уровень — это уровень библиотек и среды выполнения. Он представлен библиотеками Bionic (в Linux она называется glibc), OpenGL (поддержка графики), WebKit (движок для отображения веб-страниц), FreeType (поддержка шрифтов), SSL (зашифрованные соединения), SGL (2D-графика), библиотеки поддержки SQLite, Media Framework (нужна для поддержки мультимедиа).

Разработчики Android создали собственную версию библиотеки glibc — Bionic. Эта библиотека загружается в каждый процесс, стандартная же библиотека glibc просто огромная по меркам мобильных устройств, поэтому было принято решение ее пе-

реписать и сделать более компактной. Конечно, пришлось кое-чем пожертвовать: Bionic не поддерживает исключения C++ и не совместима с GNU libc и POSIX.

На этом же уровне работает DVM (Dalvik Virtual Machine) — виртуальная машина Java, предоставляющая необходимую функциональность для Java-приложений.

Следующий уровень — уровень каркаса приложений. На этом уровне работают различные диспетчеры:

- Диспетчер активности (Activity Manager) — управляет жизненным циклом приложения;
- Диспетчер пакетов (Package Manager) — управляет установкой пакетов прикладных программ;
- Диспетчер окон (Window Manager) — управляет окнами приложений;
- Диспетчер ресурсов (Resource Manager) — используется для доступа к строковым, графическим и другим типам ресурсов;
- Контент-провайдеры (Content Providers) — службы, предоставляющие приложениям доступ к данным других приложений;
- Диспетчер телефонии (Telephony Manager) — предоставляет API, с помощью которого можно контролировать основную телефонную информацию: статус подключения, тип сети и т. д.;
- Диспетчер местоположения (Location Manager) — позволяет приложениям получать информацию о текущем местоположении устройства;
- Диспетчер уведомлений (Notification Manager) — позволяет приложению отображать уведомления в строке состояния;
- Система представлений (View System) — служит для создания внешнего вида приложения (позволяет организовать кнопки, списки, таблицы, поля ввода и другие элементы пользовательского интерфейса).

На уровне приложений работает большинство Android-приложений: браузер, календарь, почтовый клиент, навигационные карты и т. д. Нужно отметить, что Android не делает разницы между приложениями телефона и сторонними программами, поэтому любую стандартную программу можно заменить альтернативной. При разработке приложений программист имеет полный доступ ко всем функциям операционной системы, что позволяет полностью переделать систему под себя.

1.4. Google Play Маркет

Популярности платформе Android добавляет сервис Google Play Маркет (ранее Android Market), представляющий собой онлайн-магазин приложений для платформы Android. Сервис Google Play Маркет (или просто Play Маркет) был открыт 22 октября 2008 года. С помощью Play Маркет вы можете не только распространять свои приложения, но и зарабатывать, получая при этом 70 % прибыли от ваших проданных приложений.

На мой взгляд, у Play Маркет есть единственный недостаток — доступ к этому сервису не одинаков для программистов из разных стран. Программисты из одних стран вообще не имеют право размещать на Play Маркет свои приложения, из других — могут размещать приложения бесплатно, из третьих — имеют право свои программы продавать. С другой стороны, бесплатное распространение программы тоже хорошо. Вы можете написать ограниченную версию своей программы и распространять ее бесплатно. Сервис Play Маркет будет рассмотрен в предпоследней главе этой книги. Сейчас же самое время приступить к установке необходимого программного обеспечения.

ГЛАВА 2

Подготовка среды разработки

2.1. Необходимое программное обеспечение

Для разработки приложений под Android вам нужно установить JDK (Java Development Kit), IDE Eclipse, Android SDK и Android Development Tools. Не беспокойтесь: здесь будет подробно рассмотрена установка всего этого.

Да, разработка Android-приложений осуществляется на языке Java. Очень хорошо, если вы уже знаете Java. Если нет, не беда — придется учить Java параллельно чтению этой книги. Разрешите порекомендовать вам книгу В. Монахова «Язык программирования Java и среда NetBeans, 3-е изд.», ознакомиться с которой можно по адресу: **http://bhv.ru/books/book.php?id=188402**.

Отличие для нас здесь только одно — разработка Java-приложений описана в этой книге для среды NetBeans, мы же для разработки Android-приложений воспользуемся средой Eclipse, поскольку она подходит нам больше. Но ничего страшного — здесь вы познакомитесь с Eclipse в полном объеме. Главное, чтобы вы успевали параллельно осваивать и Java, и Android.

Не скрою, разработку Android-приложений можно выполнять и на других языках программирования — например, на C#. Компания Novell выпустила среду разработки Mono for Android. Благодаря этой среде, разработчики смогут создавать приложения для операционной системы Android, используя C# и .NET. Тем не менее, среда Mono в этой книге не рассматривается. Скачать же среду Mono for Android можно по адресу: **http://mono-android.net/**, а получить о ней дополнительную информацию — по адресу: **http://www.dkws.org.ua/phpbb2/viewtopic.php?p=34562**.

Перед установкой программ поговорим о системных требованиях и материальной базе, необходимой для разработки Android-приложений. Прежде всего, вам понадобится компьютер, причем совершенно неважно, под управлением какой операционной системы он будет работать, — весь набор необходимых программ может работать под управлением и Windows, и Linux, и Mac OS.

Какую операционную систему лучше использовать? Разработка Android-приложений не зависит от конкретной операционной системы, поскольку запуск и

отладка Android-приложения будет осуществляться в эмуляторе мобильного устройства с поддержкой Android.

Лично я для разработки Android-приложений использую 64-битную версию Windows 7. И если ранее с запуском необходимого для разработки Android программного обеспечения в 64-битной версии Windows имелись некоторые проблемы, то их давно устранили, и сейчас никаких проблем более нет.

Если вы предпочитаете Linux, то желательно использовать последнюю или хотя бы предпоследнюю версию вашего дистрибутива. А пользователям Mac OS нужна операционная система версии 10.4.8 или более новая.

Думаю, проблем с установкой программ у вас не возникнет — выбор операционных систем довольно широк. Чего греха таить, даже если вы фанат Linux, то наверняка на одном из ваших компьютеров все равно установлена Windows — хотя бы для полноценного запуска Windows-игр. Так что, даже если не получится установить весь набор программ в Linux (в чем я сильно сомневаюсь, ведь та же среда разработки Eclipse мне в первую очередь знакома по операционной системе Linux), вы сможете использовать компьютер с Windows. Далее все иллюстрации будут соответствовать среде разработки, запущенной под управлением Windows 7.

Наличие физического устройства хоть и не обязательно, но весьма желательно — для тестирования программы, так сказать, в боевых условиях. Эмулятор есть эмулятор, а реальное устройство может показать недочеты вашей программы, которые невозможно будет заметить в эмуляторе. Так что, понадобится устройство с поддержкой Android той версии, под которую вы планируете разрабатывать программы. Если же планируется разработка программ под разные версии Android, желательно обзавестись несколькими устройствами по возможности разных производителей, — ведь везде есть свои нюансы, а чем «разношерстнее» оборудование, тем больше вероятность возникновения всякого рода непредвиденных обстоятельств, — то, что и нужно для процесса отладки программы.

Понимаю, что все сказанное хорошо только на бумаге, а в реальной жизни — это лишние затраты. Одно дело, если вы работаете в компании, которая занимается (или планирует заниматься) разработкой для Android, тогда все необходимые устройства будут куплены за ее счет. Другое дело, если вы желаете заняться разработкой самостоятельно — тогда покупка нескольких Android-устройств разных версий может нанести ощутимый удар по домашнему бюджету. Но тут решать только вам — только вы знаете, сколько можете позволить себе потратить на покупку всевозможных гаджетов. Рассматривайте эти вложения как инвестицию в себя — ведь с помощью Play Маркет вы всегда сможете продать свои приложения.

Можно пойти и по другому пути. Если сейчас выкладывать большую сумму не хочется, можно использовать эмулятор и распространять свои программы бесплатно. И если ваша программа будет работать неправильно, вы получите соответствующие сообщения-жалобы от скачавших ее пользователей. А исправив все возможные «глюки», вы сможете далее распространять свою программу на коммерческой основе. Однако, если вы планируете серьезно заниматься разработкой для Android, рано или поздно потратиться, все же, придется.

2.2. Установка JDK

Для запуска программ, написанных на Java, необходима среда выполнения Java — Java Runtime Environment (JRE). А для разработки Java-программ понадобится комплект разработчика Java-приложений — Java Development Kit (JDK), включающий: компилятор Java, стандартные библиотеки классов Java, документацию, примеры и саму JRE. Так что, вам не нужно сначала устанавливать JRE, а потом — JDK, можно сразу комплектно установить JDK.

Рис. 2.1. Установка JDK SE 8u11 для Windows

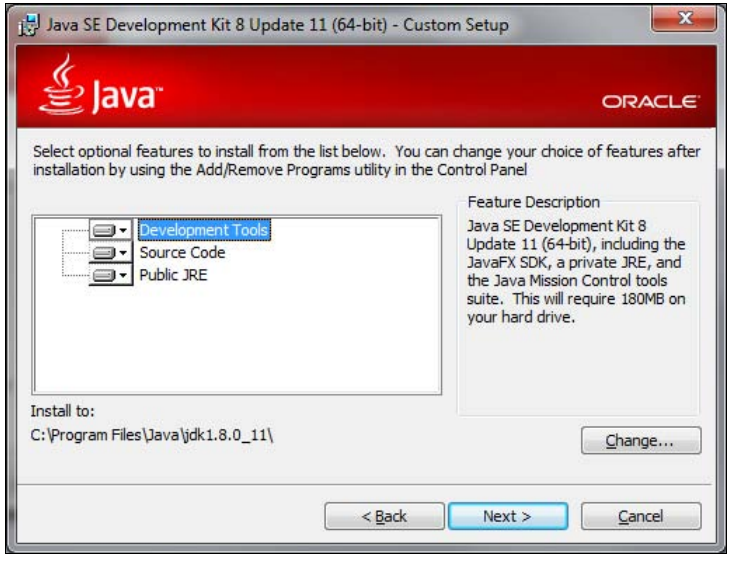


Рис. 2.2. Каталог установки JDK

Скачать JDK можно по адресу:
http://www.oracle.com/technetwork/java/javase/downloads/index.html.

При загрузке обратите внимание на версию вашей операционной системы — необходимо скачивать JDK, подходящий именно для вашей ОС. На момент написания этих строк доступна версия 8u11, поэтому пользователям 64-битной версии Windows нужно скачать файл jdk-8u11-windows-x64.exe, а пользователям 32-битной версии — файл jdk-8u11-windows-i586.exe.

После загрузки запустите загруженный файл (рис. 2.1). В процессе установки JDK нет ничего сложного — просто нажимайте кнопку **Next** и следуйте инструкциям мастера установки. Запомните каталоги, в которые вы установите JDK (рис. 2.2) и JRE (рис. 2.3).

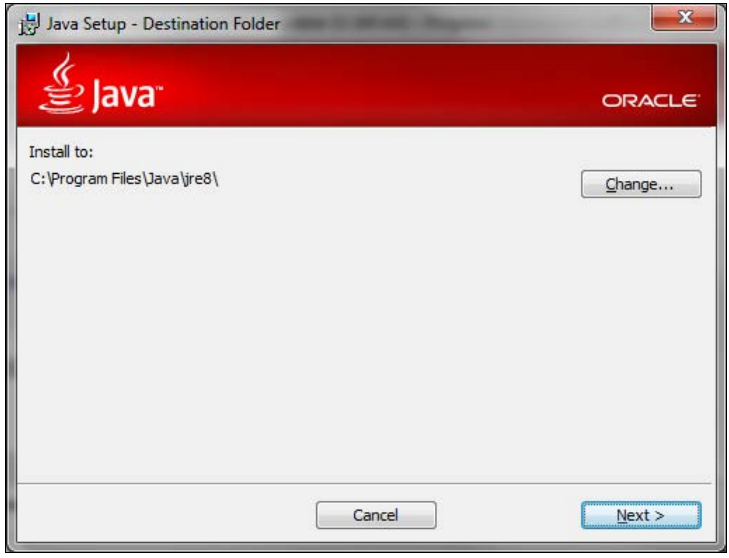


Рис. 2.3. Каталог установки JRE

2.3. Установка среды разработки

Ранее установка среды разработки была немного сложной. Приходилось отдельно устанавливать каждый компонент, потом связывать эти компоненты воедино. Все это несколько усложняло процесс установки. Но, начиная с четвертой версии Android, вам не придется загружать все необходимое программное обеспечение по отдельности — загружается единый пакет, в составе которого имеется все необходимое программное обеспечение, а именно:

- среда Eclipse и ADT-плагин (служит для интеграции средств разработчика Android в среду Eclipse — ранее как раз этот плагин нужно было устанавливать отдельно);
- инструменты Android SDK;

- инструменты платформы Android;
- версию платформы Android;
- образ Android для эмулятора.

Загрузить единый пакет можно по адресу:
https://developer.android.com/sdk/index.html?hl=i.

Зайдя на указанную страницу, нажмите большую синюю кнопку **Download Eclipse ADT with the Android SDK for Windows**. На следующей страничке нужно согласиться с лицензионным соглашением и выбрать архитектуру Windows (рис. 2.4).

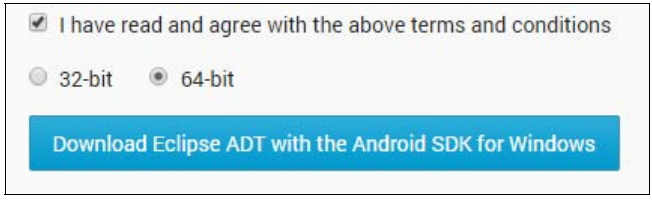


Рис. 2.4. Выберите тип системы

Загруженный ZIP-архив распакуйте, куда вам будет удобно. Я распаковал его в корневой каталог диска C:\ — получился каталог C:\adt-bundle-windows-x86_64-20140702, который я для удобства переименовал просто в C:\adt.

Если вы все сделали, как и я, то перейдите в каталог c:\adt\eclipse\ и запустите исполнимый файл eclipse.exe. При первом запуске среда спросит вас, в каком каталоге будет храниться ваше рабочее пространство. Обычно это подкаталог workspace вашего каталога профиля (рис. 2.5). Просто отметьте флажок **Use this as the default and do not ask again** и нажмите кнопку **OK** — откроется окно среды Eclipse (рис. 2.6).

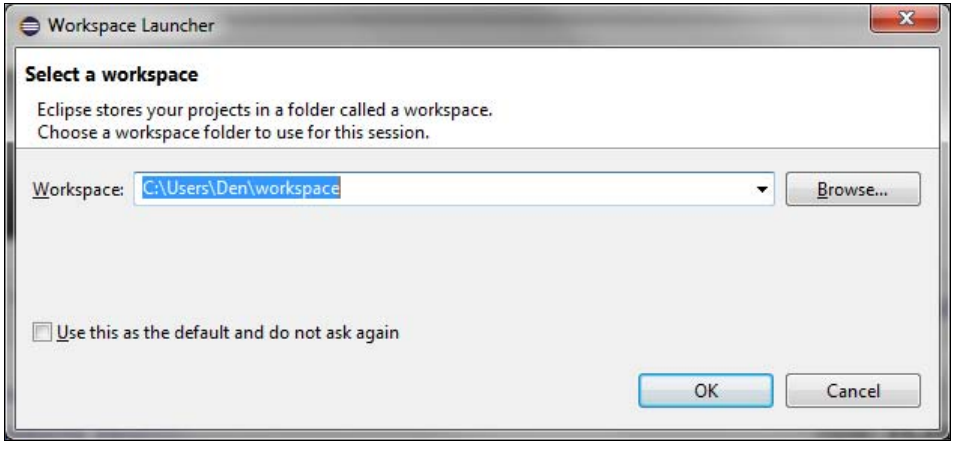


Рис. 2.5. Установка каталога для рабочего пространства

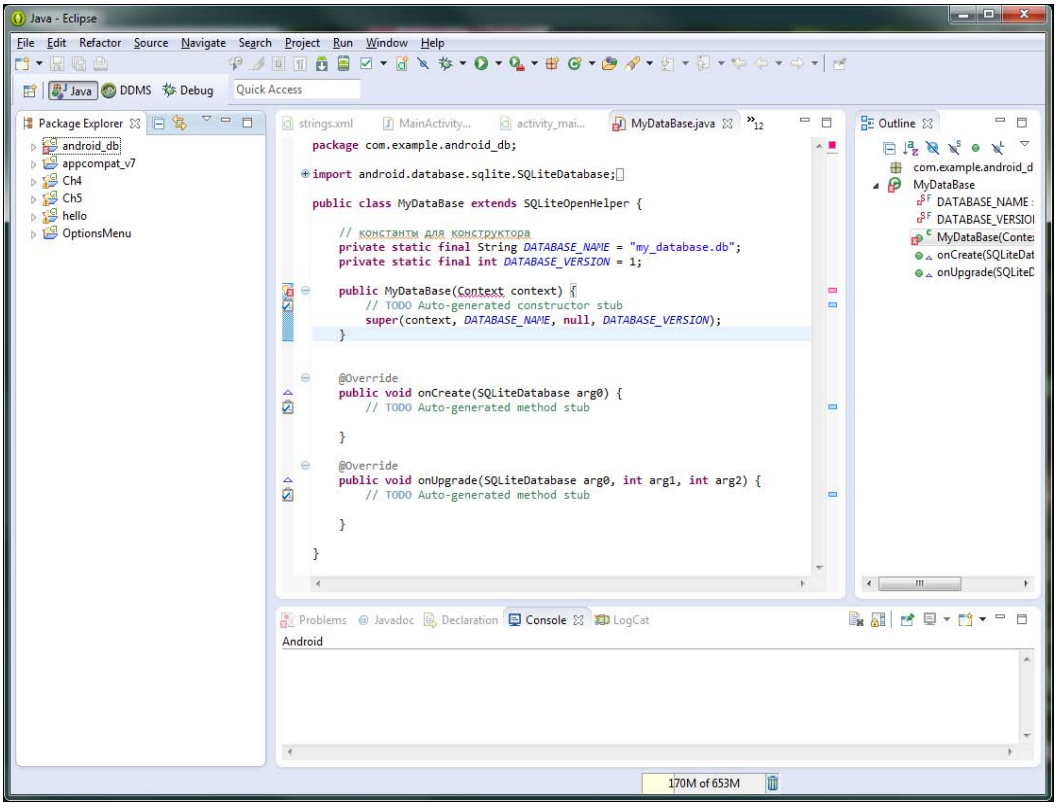


Рис. 2.6. Среда Eclipse

> ***Имя рабочего каталога***
>
> Если у вас имя пользователя содержит кириллицу, то во избежание проблем с работой среды лучше установить в качестве рабочего каталога *C:\workspace* или любой другой каталог, в названии которого нет кириллицы.

Выполните команду меню **Window | Android SDK Manager**. Основная задача этой команды — загрузить и установить из Интернета все необходимое для разработки приложений для ОС Android выбранной версии. Вы можете выбрать все версии сразу (поддерживаются версии от 1.5 до 5.0) или только интересующие вас (рис. 2.7).

Как можно видеть, версии Android 5.0 соответствует уровень **API 21**. Выберите все пакеты, связанные с API 21, и нажмите кнопку **Install *N* packages** (на рис. 2.7 — **Install 25 packages**, у вас может быть другое число). В следующем окне (рис. 2.8) включите переключатель **Accept License** и нажмите кнопку **Install**.

Поскольку все устанавливаемые пакеты хранятся на серверах Google, то для продолжения установки необходимо соединение с Интернетом — это на тот случай, если вы по каким-то причинам разорвали соединение. Установка пакетов длится мучительно долго, но у вас нет другого выбора, — нужно просто ждать (рис. 2.9).

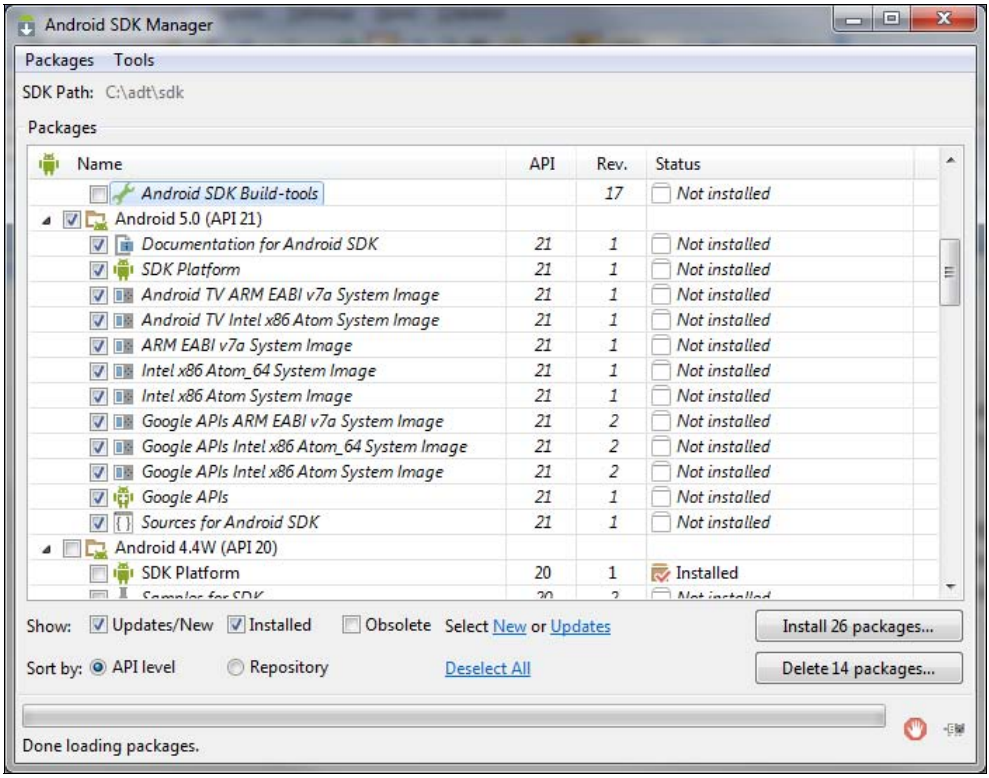


Рис. 2.7. Окно **Android SDK Manager**

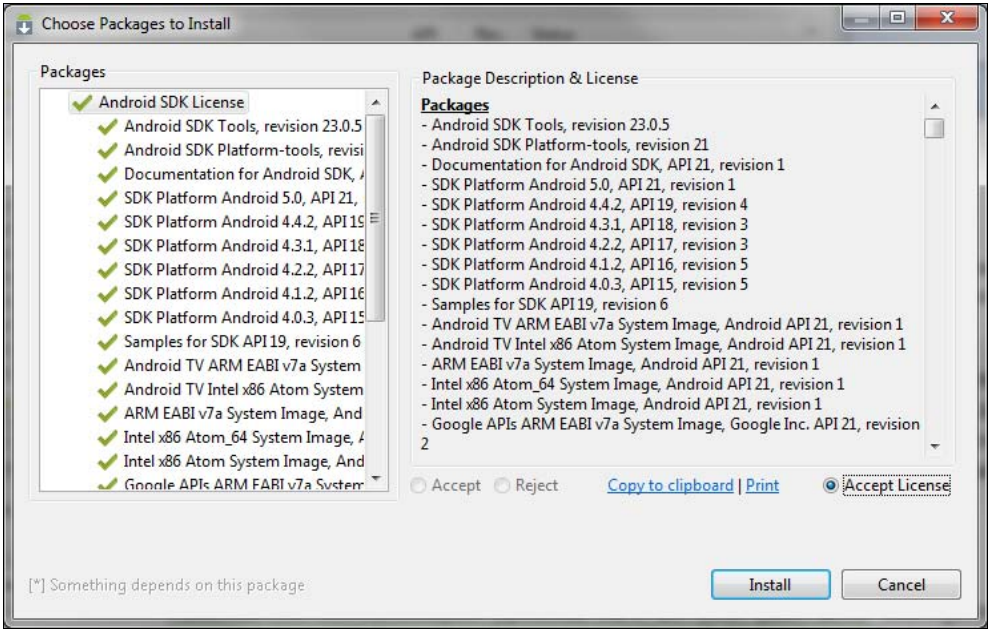


Рис. 2.8. Выбор пакетов для установки

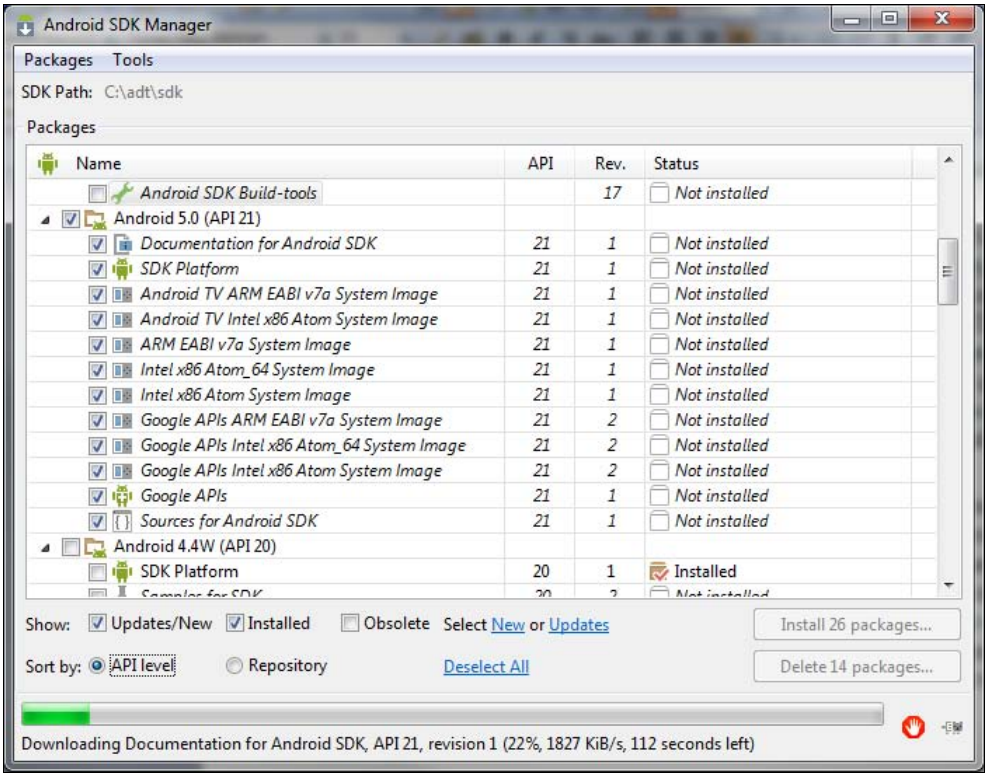


Рис. 2.9. Идет загрузка и установка выбранных пакетов

Спустя некоторое время (вы могли выбрать меньшее количество пакетов, и установка прошла бы быстрее) вы увидите сообщение (рис. 2.10) о том, что все выбранные пакеты установлены/обновлены. В этом окне также содержится рекомендация закрыть окно SDK Manager, открыть его заново, а также выполнить из окна среды Eclipse (см. рис. 2.4) команду **Help | Check for Updates**, чтобы проверить, нуждается ли плагин Android в обновлении. Выполните эту команду. В моем случае Eclipse сообщила, что обновлений не найдено (рис. 2.11), но в вашем, возможно, понадобятся обновления (с момента написания этих строк до момента покупки вами этой книги может пройти достаточно много времени).

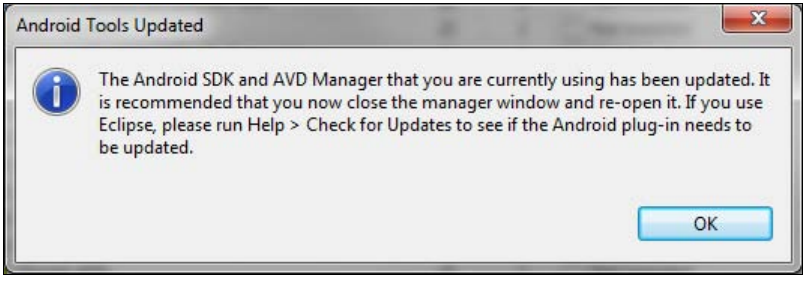


Рис. 2.10. Пакеты установлены/обновлены

Рис. 2.11. Обновлений для Eclipse не найдено

Наверное, вам интересно, что же мы только что установили? А установили мы следующие компоненты:

- API-библиотеки, необходимые для разработки Android-приложений;
- документацию по Android;
- примеры программ, демонстрирующие основные возможности Android;
- эмулятор Android-устройства (Android Virtual Device, AVD), позволяющий запускать и тестировать ваши программы без наличия физического мобильного устройства;
- инструментальные средства для разработки, позволяющие компилировать и отлаживать ваши приложения.

2.4. Плагин ADT

Как мы уже отмечали ранее, в Android версий до 4.0 нужно было самостоятельно устанавливать плагин ADT в среду Eclipse. Сейчас этого делать не придется, т. к. скачанный пакет содержит уже настроенную среду Eclipse. Плагин ADT упрощает разработку Android-приложений. Давайте разберемся, за счет чего осуществляется это упрощение. Итак, ADT:

- добавляет в Eclipse мастер создания проекта Android;
- добавляет редактор Layout Editor, необходимый для создания графического интерфейса приложения;
- добавляет редакторы XML-ресурсов приложения;
- а также позволяет запускать эмулятор Android и добавляет средства отладки непосредственно в среду Eclipse, что позволяет отлаживать программы, не выходя из привычной IDE.

2.5. Уровни API

Прежде чем приступить к чтению следующего раздела, поговорим об уровне API. Уровень API — это число, которое однозначно идентифицирует версию Android. Именно это число указывается напротив версии платформы в окне настроек Eclipse (см. рис. 2.12). Уровни API представлены в табл. 2.1.

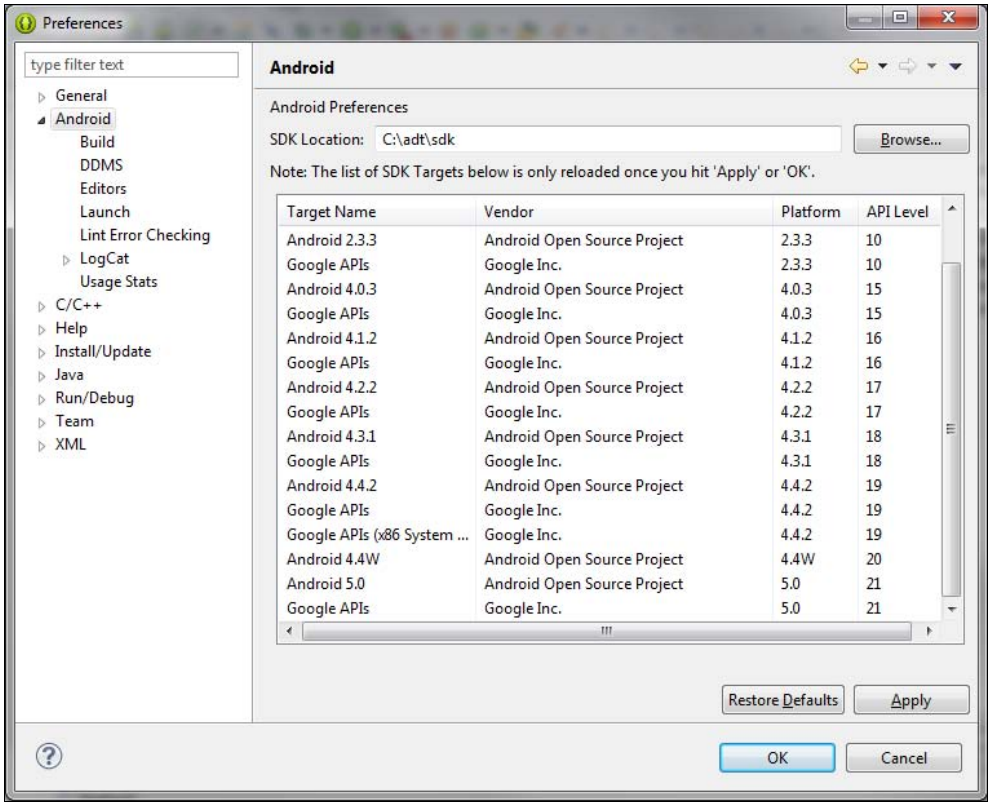


Рис. 2.12. Поддерживаемые уровни API

Таблица 2.1. *Уровни API Android*

Уровень	Версия платформы	Уровень	Версия платформы
1	1.0	12	3.1
2	1.1	13	3.2
3	1.5	14	4.0, 4.0.1, 4.0.2
4	1.6	15	4.0.3, 4.0.4
5	2.0	16	4.1, 4.1.1
6	2.0.1	17	4.2, 4.2.2
7	2.1-update1	18	4.3
8	2.2	19	4.4
9	2.3.1	20	4.4W
10	2.3.3, 2.3.4	21	5.0
11	3.0		

Пусть пока эта табличка носит просто информационный характер, а далее (в *главе 3*) будет показано, как использовать ее на практике.

2.6. Подробнее о составе Android SDK

Ранее было сказано, что в состав Android SDK входят библиотеки, документация, эмулятор, примеры программ и инструментальные средства. Сейчас речь пойдет о последнем компоненте — об инструментальных средствах. Конечно, прямо сейчас они вам не понадобятся, не потребуются и в *главе 3*, когда мы станем разрабатывать первое Android-приложение, но об их существовании вы обязаны знать. Инструментальные средства Android SDK находятся в каталоге tools каталога, в который вы установили Android SDK:

- android.bat — позволяет создавать, удалять и настраивать виртуальные устройства из командной строки. Все эти операции можно сделать в окне **Android SDK and AVD Manager** визуально;
- ddms.bat (Dalvik Debug Monitor Service) — позволяет управлять процессами на эмуляторе или на физическом устройстве, что помогает в отладке приложения;
- draw9patch.bat — графический редактор, позволяющий создавать графику для ваших Android-приложений;
- emulator.exe — собственно, сам эмулятор Android-устройства. Не спешите его запускать, он нуждается в предварительной настройке, которая будет описана в следующем разделе;
- hierarchyviewer.bat — позволяет оптимизировать графический интерфейс разрабатываемого приложения;
- mksdcard.exe — инструмент создания образа диска SD-карты, который можно использовать в эмуляторе для имитации внешней карты мобильного устройства;
- sqlite3.exe — инструмент для доступа к файлам данных SQLite.

В каталоге tools вы найдете и другие утилиты, но они менее важные, чем перечисленные здесь.

2.7. Эмулятор Android-устройства

2.7.1. Создание Android Virtual Device (AVD)

Самая важная утилита из набора Android SDK — это эмулятор Android-устройства. Эмулятор позволяет отлаживать и тестировать приложения в реальной среде выполнения без необходимости их установки на физическое устройство. Даже если у вас есть физическое устройство, не спешите на нем запускать ваше приложение, которое может работать нестабильно. Сначала нужно запустить устройство в эмуляторе, и если приложение будет работать нормально, можно пробовать запускать его на реальном устройстве.

Эмулятор сразу не готов к использованию. Перед его запуском нужно создать Android Virtual Device (AVD) — виртуальное устройство Android. AVD определяет

настройки целевого устройства: вы можете создать несколько AVD — например, одно для платформы 2.2, другое — для 3.0. Одно с экраном 320×480, другое — 480×800, что позволяет тестировать приложение в разных условиях.

Создать AVD можно утилитой android.bat из каталога tools или визуально с помощью окна **Android SDK and AVD Manager** (его можно вызвать из меню **Window** среды Eclipse). Лично я предпочитаю второй способ, поэтому перейдите к менеджеру SDK и AVD. На вкладке **Android Virtual Devices** перечислены созданные вами AVD — по умолчанию ни одного виртуального устройства не создано, но на рис. 2.13 вы видите окно моего рабочего эмулятора, в котором уже созданы некоторые AVD. Для создания нового устройства нажмите кнопку **Create**. В появившемся окне (рис. 2.14) введите название устройства (**Name**), выберите целевую платформу (**Target**). Обратите внимание, что вы, хотя и скачали SDK 5, сам эмулятор, как и среда Eclipse, на текущий момент старые, поэтому выбрать можно только **4.4W**. Позже я покажу, как создать приложение для платформы Android 5.0.

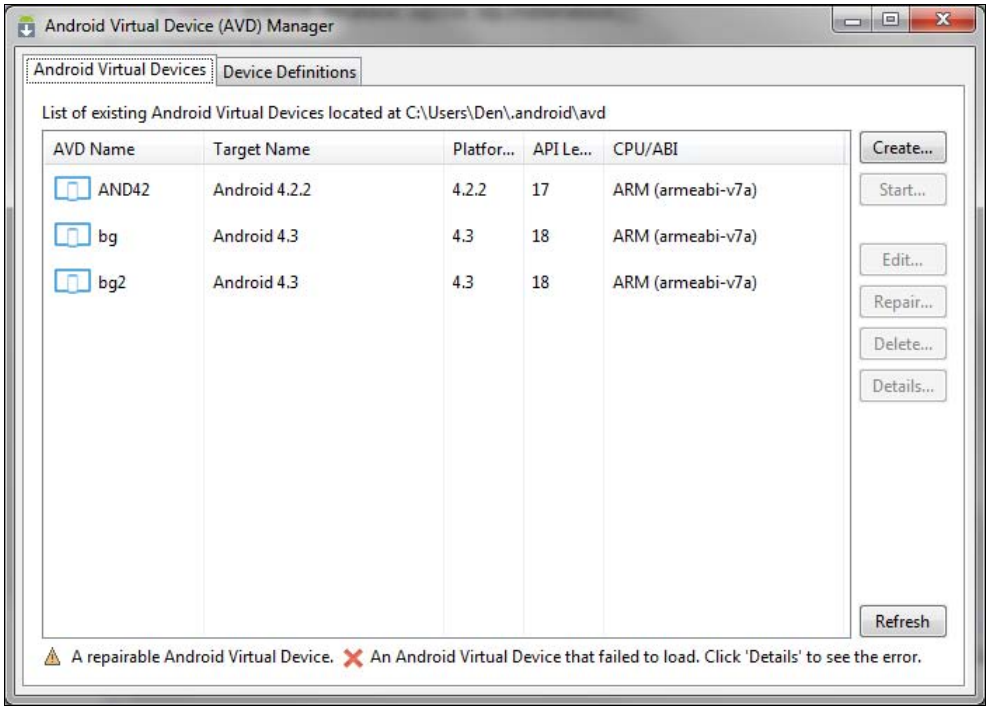


Рис. 2.13. Окно **Android Virtual Device (AVD) Manager**

Параметр **SD Card** позволяет создать внешнюю карту устройства. Вы можете или указать размер новой карты (не менее 9 Мбайт), или выбрать ранее созданный программой mksdcard.exe образ SD-карты.

Очень важный параметр **Skin**, позволяющий выбрать разрешение экрана устройства:

- **WVGA800** (Wide Video Graphics Array) — высокая плотность, нормальный (полноразмерный) экран, размер 480×800;

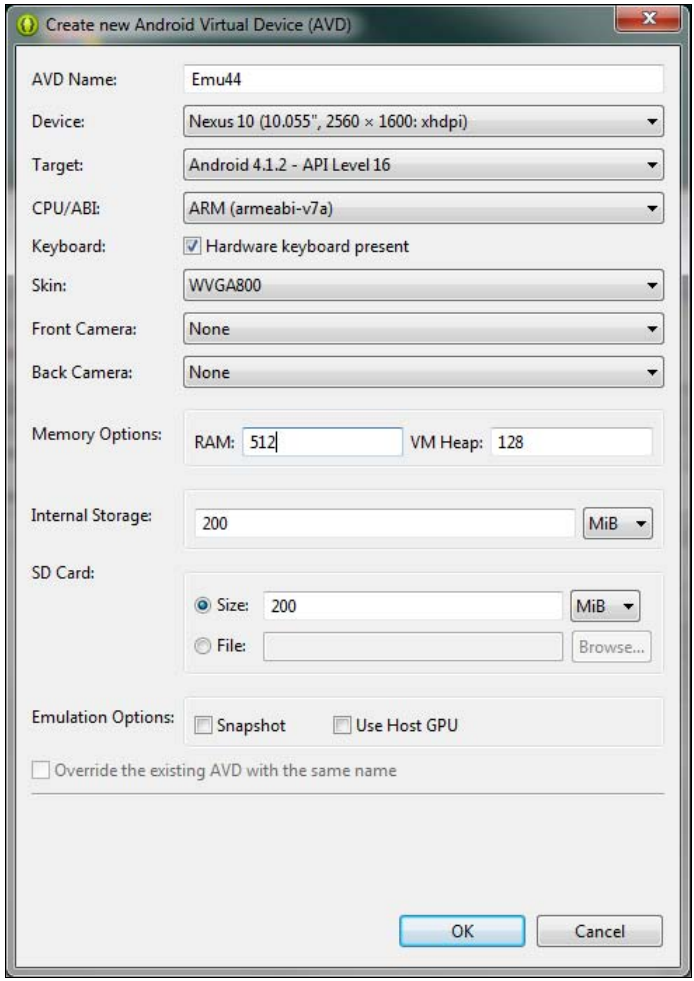


Рис. 2.14. Создание нового AVD

- **HVGA** (Half-size VGA Video Graphics Array) — средняя плотность, нормальный экран, размер 320×480;
- **WVGA854** (Wide Video Graphics Array) — высокая плотность, нормальный экран, разрешение 480×854;
- **QVGA** (Quarter Video Graphics Array) — низкая плотность, разрешение 240×320, малый экран (как у Mini-версий смартфонов);
- **WQVGA** (Wide Quarter Video Graphics Array) — низкая плотность, нормальный экран, разрешение 240×400.

Выбирайте разрешение, которое поддерживает ваше реальное устройство (или то устройство, для которого вы хотите разработать программу).

Для создания AVD нажмите кнопку **OK**, и созданное вами устройство появится на вкладке **Android Virtual Devices** (рис. 2.15).

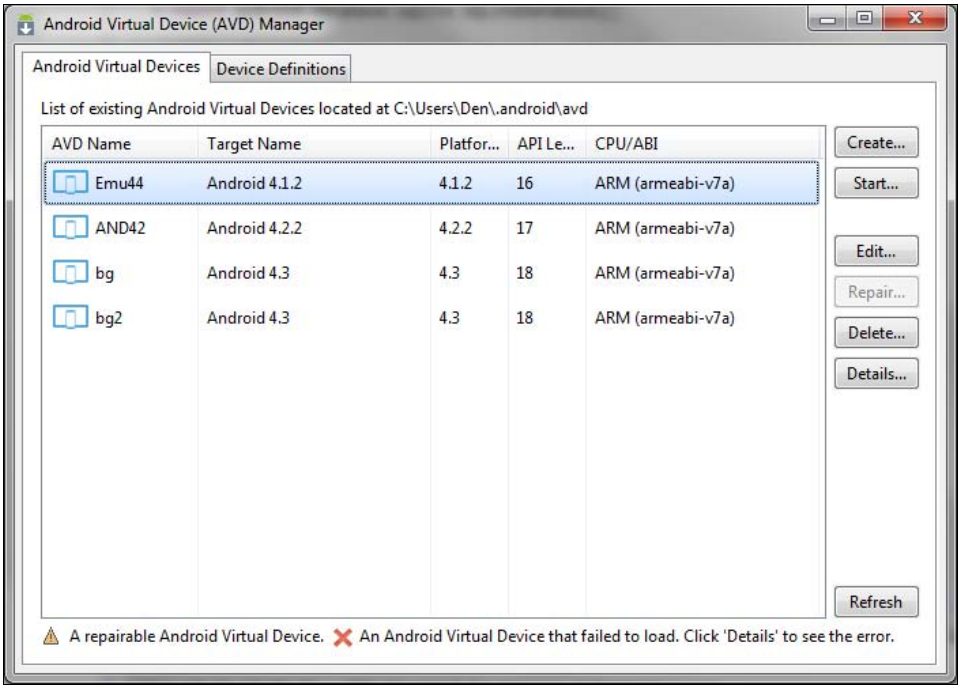


Рис. 2.15. Только что созданное виртуальное устройство

2.7.2. Запуск эмулятора и проблемы при запуске

Для запуска эмулятора выделите нужное вам виртуальное устройство и нажмите кнопку **Start** (см. рис. 2.15). Откроется окно, в котором можно установить дополнительные параметры виртуального устройства, но, как правило, достаточно просто нажать кнопку **Launch** (рис. 2.16).

В идеале все должно запуститься, но вы можете получить следующее сообщение об ошибке (рис. 2.17).

Дело в том, что эмулятор по умолчанию читает конфигурацию из следующего каталога: C:\Documents and Settings\<имя пользователя>\.android\. И если имя пользователя содержит русские буквы, то эмулятор не сможет прочитать конфигурацию, и вы увидите такое сообщение об ошибке.

Решить проблему можно одним из двух способов (выберите подходящее для себя решение):

- можно дать пользователю другое имя, используя при этом только латинские буквы, и перезапустить компьютер. Затем запустить менеджер AVD, удалить все созданные виртуальные устройства и создать их заново;
- можно установить должным образом переменные среды.

Если переименовывать пользователя вам не с руки, придется устанавливать переменные среды. В Windows XP (если им еще кто-то пользуется) нужно выполнить следующие действия:

1. Щелкнуть правой кнопкой на значке **Мой компьютер**, выбрать команду **Свойства**.
2. В открывшемся окне перейти на вкладку **Дополнительно**, нажать кнопку **Переменные среды**.
3. Добавить переменную среды (см. далее).

В Windows Vista/Windows 7 действия примерно те же:

1. Нажать кнопку **Пуск**, щелкнуть правой кнопкой мыши на пункте **Компьютер**, выбрать команду **Свойства**.
2. В открывшемся окне выбрать команду **Дополнительные параметры системы** (слева).
3. Нажать кнопку **Переменные среды**.
4. Добавить переменную среды.

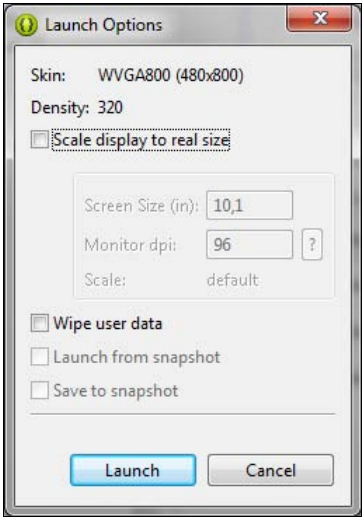


Рис. 2.16. Параметры запуска виртуального устройства

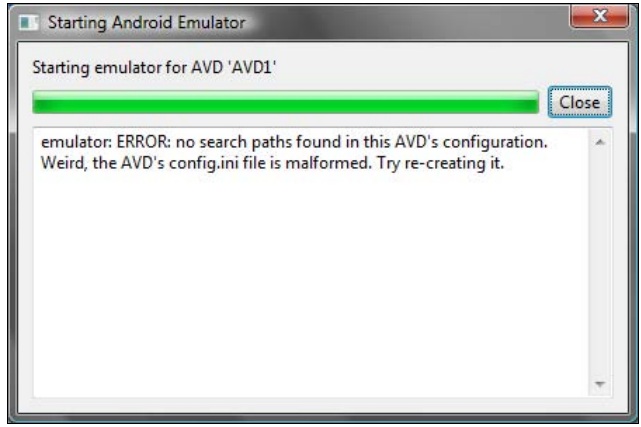


Рис. 2.17. Ошибка при запуске эмулятора

Итак, вне зависимости от операционной системы (XP, Vista или 7), вы «добрались» до окна **Переменные среды** (рис. 2.18). Нажмите кнопку **Создать** и создайте переменную среды с именем **ANDROID_SDK_HOME**, в качестве значения переменной установите любой существующий каталог, в имени которого отсутствуют русские буквы — например, **C:\Android** (или **c:\workshop** — как вам нравится больше), — главное, чтобы этот каталог существовал.

Также в области **Системные переменные** найдите переменную **PATH** и нажмите кнопку **Изменить**. Добавьте к ее значению строку:

```
;C:\Android
```

Понятно, что имя каталога должно соответствовать тому, которое вы создали.

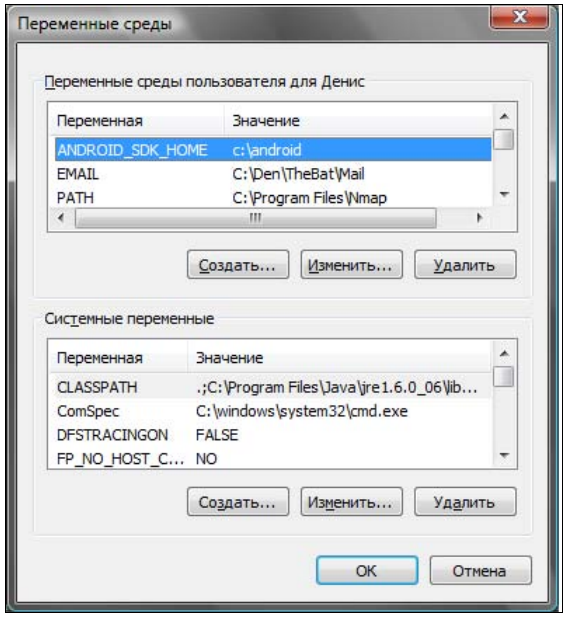


Рис. 2.18. Окно **Переменные среды**

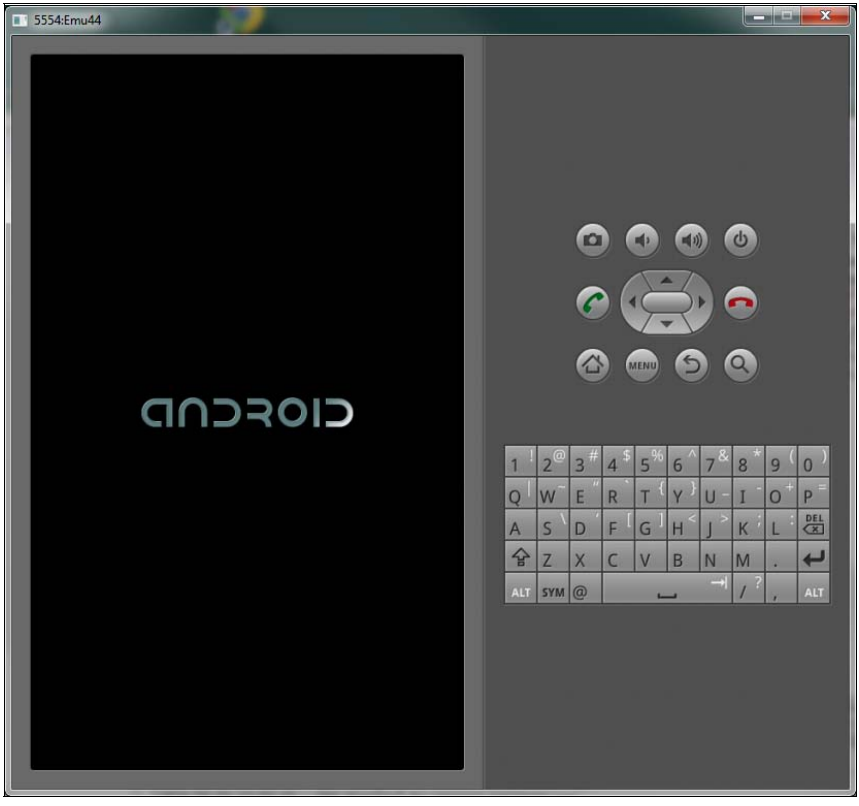


Рис. 2.19. Запуск эмулятора

Вот и все. Перезагружать компьютер не нужно. Достаточно перезапустить менеджер SDK и AVD. Правда, после этого список виртуальных устройств будет пуст — ведь конфигурационный каталог уже не содержит никаких настроек, в том числе и профилей виртуальных устройств. Ничего страшного — просто заново создайте устройство. Затем нажмите кнопку **Start**, а потом — **Launch**. Вы увидите окно эмулятора (рис. 2.19).

Нужно отметить, что запуск эмулятора — процесс очень длительный. Сначала в окне эмулятора будет красоваться текстовая надпись ANDROID_, затем логотип Android (см. рис. 2.19) и только спустя несколько минут вы увидите сам эмулятор. Внешний вид эмулятора различен в зависимости от выбранной платформы и параметра **Skin** виртуального устройства. На рис. 2.20 изображен эмулятор для платформы 4.1 и экрана WVGA800, на рис. 2.21 — список установленных приложений эмулятора (Android 4.1). На рис. 2.22 изображен эмулятор для версии Android 4.4.

НЕ ЗАКРЫВАЙТЕ ОКНО ЭМУЛЯТОРА

Не нужно закрывать окно эмулятора, даже если вам кажется, что он завис, а в его заголовке есть надпись **Не отвечает**. Нужно просто немного подождать, и эмулятор запустится.

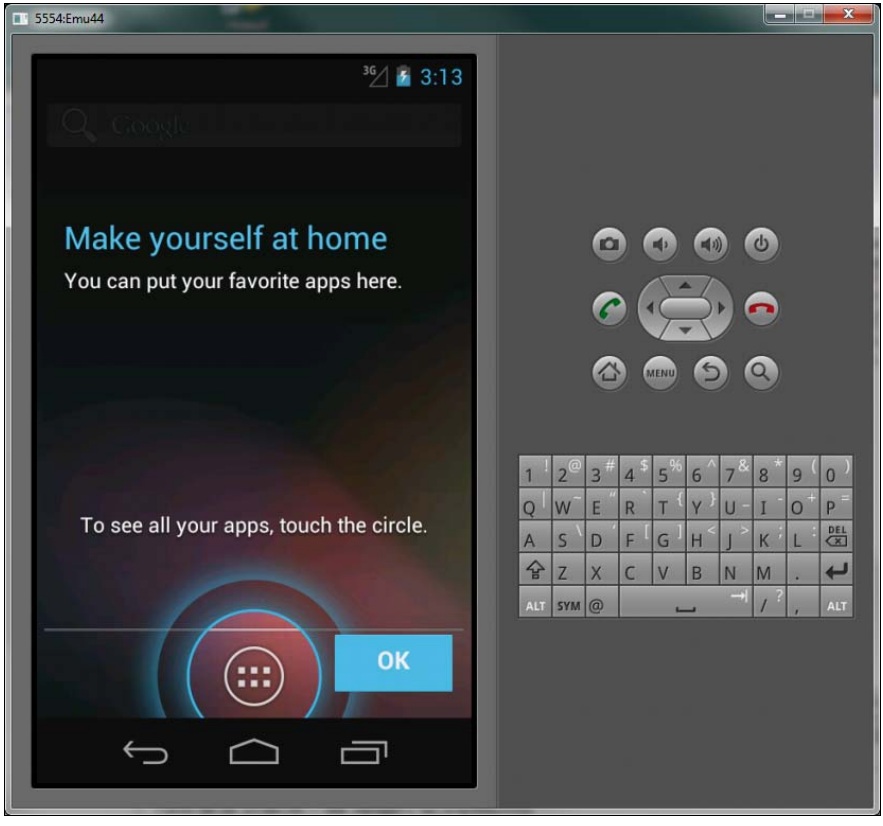


Рис. 2.20. Эмулятор Android (версия 4.1)

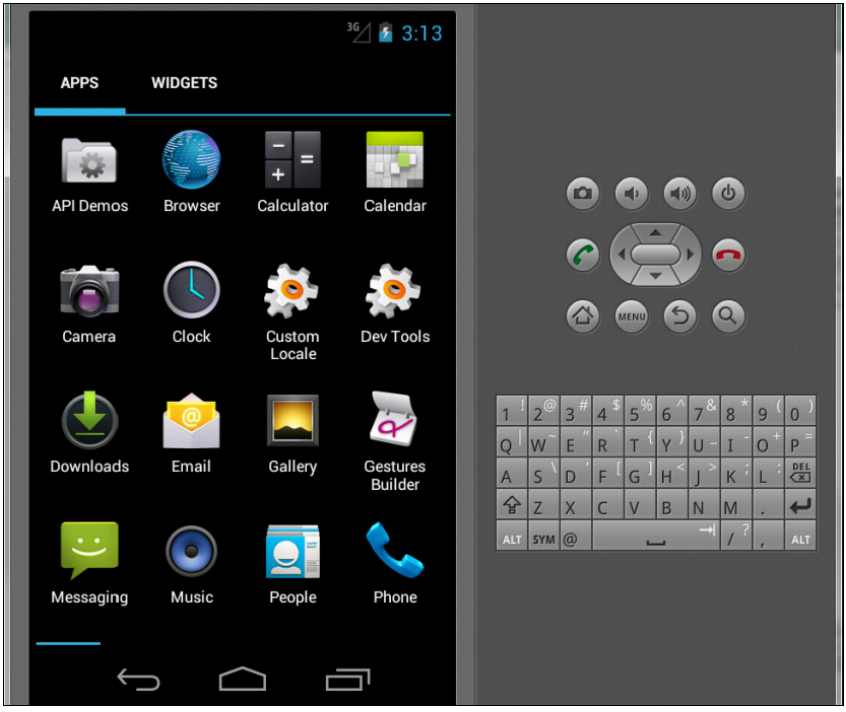


Рис. 2.21. Установленные приложения (версия 4.1)

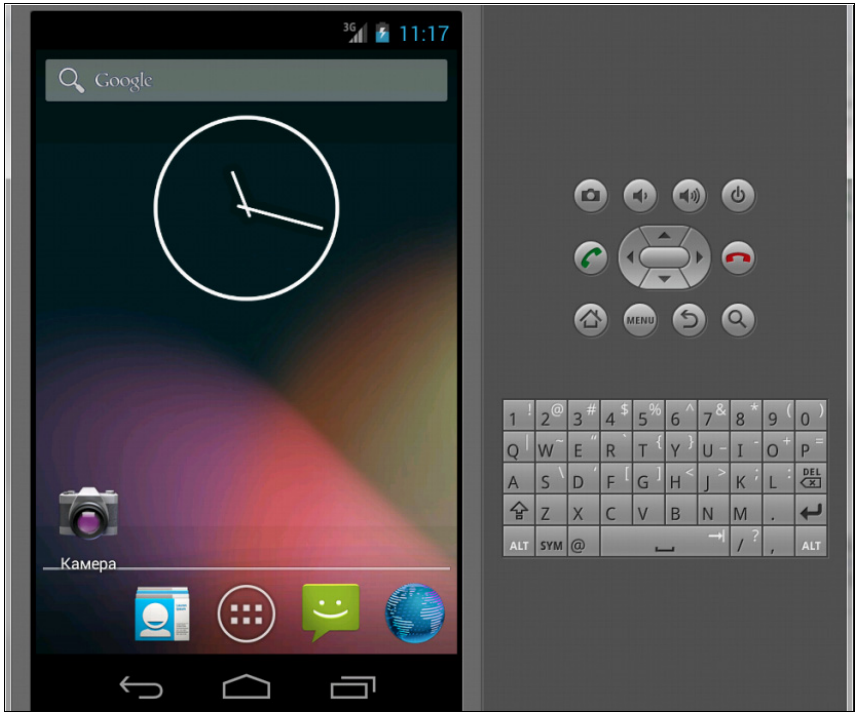


Рис. 2.22. Эмулятор Android (версия 4.4)

2.7.3. Комбинации клавиш эмулятора

При работе с эмулятором вы можете использовать комбинации клавиш, описанные в табл. 2.2.

Таблица 2.2. *Управление эмулятором Android с помощью клавиатуры*

Клавиша	Действие
<Esc>	Нажать кнопку **Back**
<Home>	Нажать кнопку **Home**
<F2>	Нажать кнопку **Menu**
<Shift>+<F2>	Нажать кнопку **Start**
<F3>	Нажать кнопку **Call/Dial**
<F4>	Положить «трубку»
<F5>	Нажать кнопку **Search**
<F7>	Нажать кнопку **Power**
<Ctrl>+<F3>	Вызвать камеру
<Ctrl>+<F5>	Увеличить громкость
<Ctrl>+<F6>	Уменьшить громкость
<Alt>+<Enter>	Полноэкранный режим
<Ctrl>+<F11>, <Ctrl>+<F12>	Вращение экрана

2.7.4. Управление виртуальным устройством с помощью команды *adb*

В каталоге c:\adt\sdk\platform-tools\ (при условии, что вы все установили в c:\adt) находится утилита adb. ADB (Android Debug Bridge) — это «отладочный мост» для Android. Утилита полезна тем, что она может вывести список подключенных устройств (как виртуальных, так и физических), а также выполнить различные действия над ними. Например, программа может установить приложение на устройство. Вот список самых полезных действий:

- `adb install <путь к APK-файлу>` — устанавливает приложение на устройство;
- `adb uninstall <имя APK-файла>` — удаляет приложение из устройства;
- `adb devices` — отображает список подключенных устройств;
- `adb reboot` — перезагрузка Android-устройства;
- `adb reboot recovery` — быстрая перезагрузка Android-устройства в режим **Recovery**;
- `adb shell` — вход в консоль, вы должны увидеть символ # в строке;

- `adb shell cat /proc/mtd` — смотрим информацию о «разделах» памяти аппарата;
- `adb shell df` — информация о разделах и свободных ресурсах;
- `adb push` — копирует файл в устройство.

На рис. 2.23 показан результат выполнения команды `adb devices` — в данный момент подключен только эмулятор.

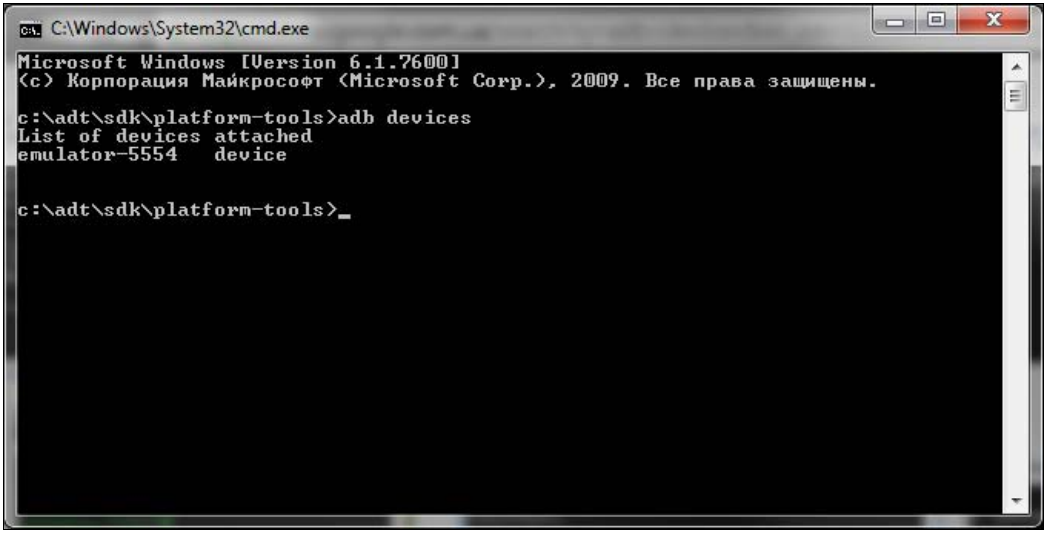


Рис. 2.23. Результат выполнения команды `adb devices`

2.8. Как подключить физическое устройство для запуска на нем приложений?

Первым делом откройте окно **Android SDK Manager** и установите **Google USB Driver** (рис. 2.24). Дождитесь, пока драйвер будет установлен.

Далее включите режим USB-отладки на вашем устройстве и подключите его с помощью USB-кабеля к вашему компьютеру. Компьютер запросит указать путь к драйверам — они находятся в каталоге extras\google\usb_driver. Теперь выполняем команду `adb devices` и видим, что устройство подключено (рис. 2.25).

Если вы не видите подключенного устройства, убедитесь, что вы-таки включили режим USB-отладки, и устройство не подключается, как устройство хранения данных. Если это так, тогда введите команды:

```
adb kill-server
adb start-server
adb devices
```

Вы должны увидеть свое устройство.

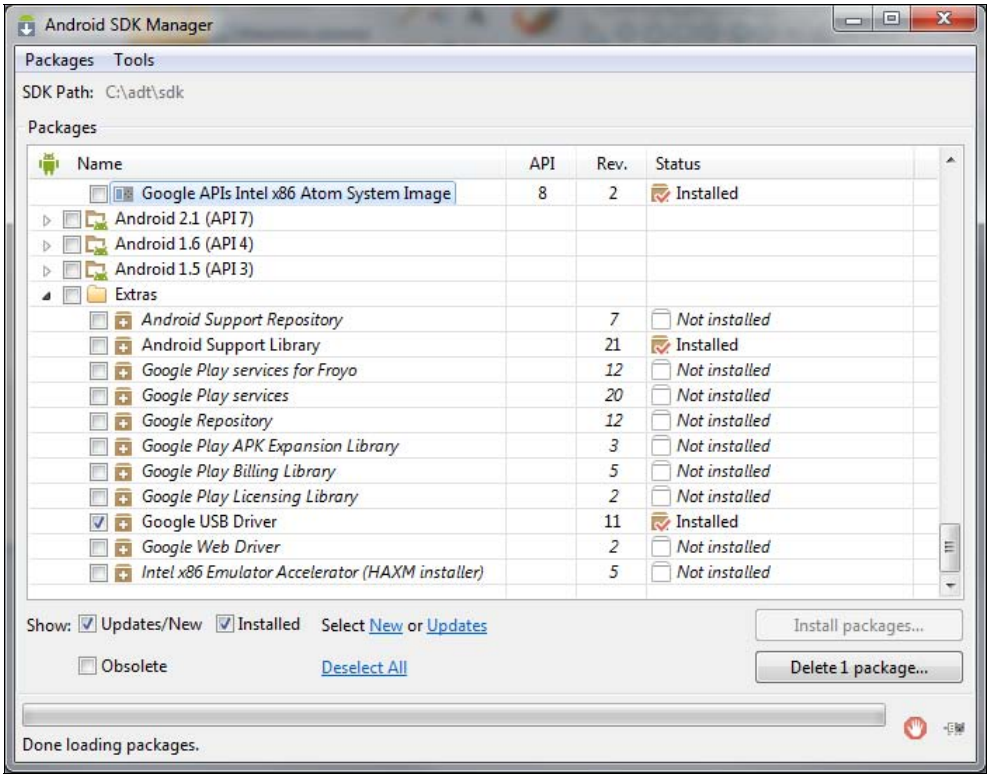


Рис. 2.24. Google USB Driver установлен

Рис. 2.25. ID физического Android-устройства

2.9. Правильное включение отладки по USB

До версии 4.1 (включительно) для включения режима отладки по USB достаточно было зайти на устройстве в меню **Настройки | Для разработчиков** (или **Параметры разработчика** — в зависимости от версии Android), переместить переключатель в положение **ВКЛ**, после чего включить параметр **Отладка по USB** и нажать **OK**.

Однако, начиная с версии 4.2, пункт **Параметры разработчика** скрыт, и просто так включить отладку по USB уже нельзя. Чтобы открыть меню с параметрами для разработчиков, перейдите в раздел **Настройки | О планшетном ПК** (или **Об устройстве** — смотря, какое у вас устройство: планшет или смартфон). Несколько раз нажмите на поле **Номер сборки** (рис. 2.26), после чего над полем появится сообщение: **Шагов до включения режима разработчика**. Нажмите это поле столько раз, сколько будет указано в этом сообщении. И только после этого откроется раздел **Параметры разработчика**, в котором нужно включить параметр **Отладка по USB** (рис. 2.27).

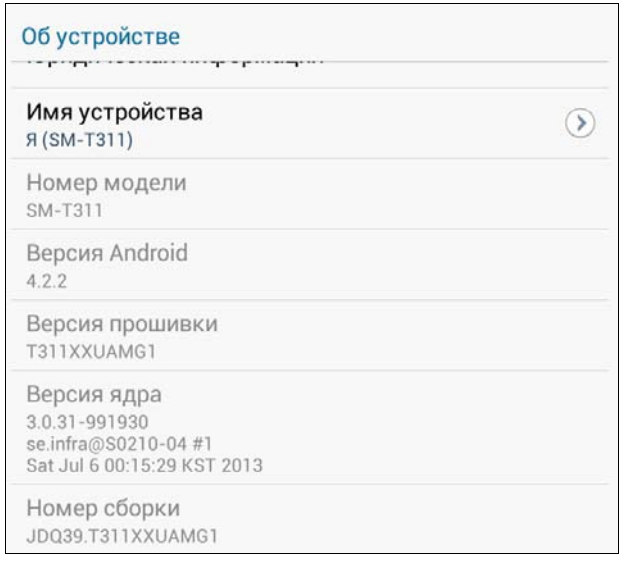


Рис. 2.26. Об устройстве

Что ж, теперь можете вздохнуть с облегчением — у вас есть все необходимое для разработки Android-приложений. В следующей главе мы напишем первое Android-приложение и поговорим о подводных камнях, с которыми вы столкнетесь при запуске его в тестовом режиме.

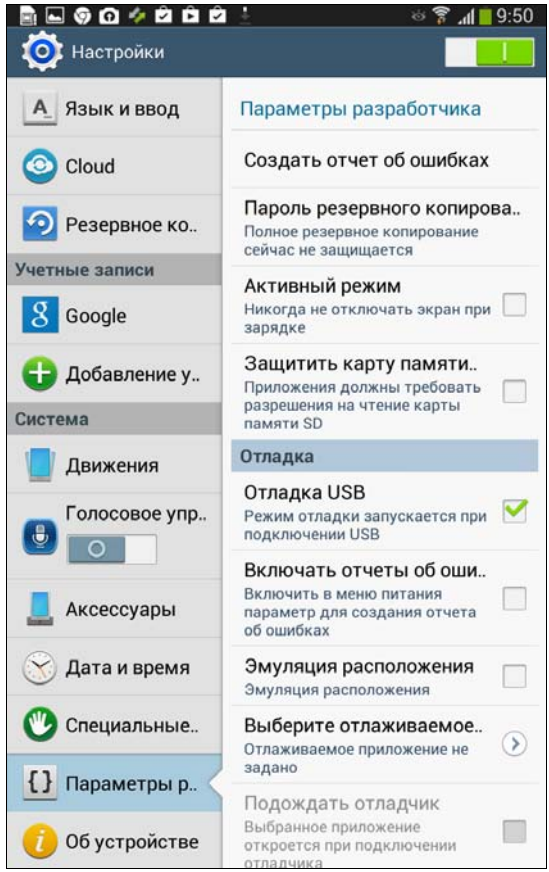


Рис. 2.27. Параметры разработчика

ГЛАВА 3

Первое Android-приложение

3.1. Разработка приложения в Eclipse

В этой главе мы создадим самую простую программу для Android — приложение, выводящее традиционную строку «Hello world!». Такая программа — традиция в мире программирования, и отступать от нее не хочется.

Запустите Eclipse и выполните команду **File | New | Project**. В открывшемся окне (рис. 3.1) для создания Android-проекта выберите **Android | Android Application**

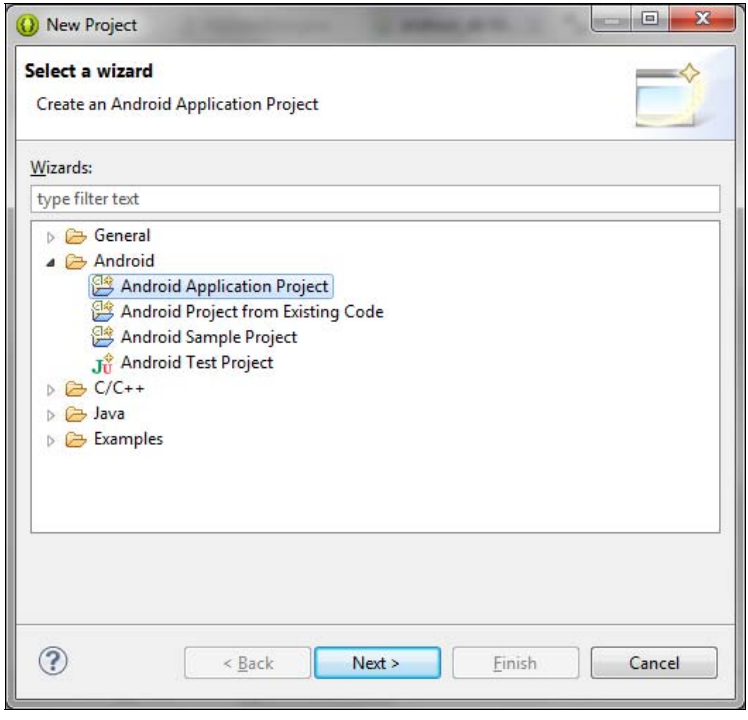


Рис. 3.1. Создание нового проекта

Project и нажмите кнопку **Next**. Далее в окне **New Android Application** (рис. 3.2) нужно установить параметры так:

- **Application Name** — название приложения. Эта строка появится в верхней части экрана устройства — можете ввести сюда любой подходящий текст на свое усмотрение;
- **Project Name** — имя проекта. Введите: `HelloWorld` — это имя будет служить в качестве имени каталога для вашего проекта;
- **Package Name** — имя пакета. Введите: `com.sample.helloworld`;
- **Minimim Required SDK** — минимальная версия Android, на которой будет работать ваше приложение. У нас простой проект, поэтому можете выбрать любой вариант. Для более сложных проектов нужно учитывать уровень используемого API. Если вы решили работать с API, доступным только начиная с версии 4.0, то его нужно и выбирать;
- **Target SDK** — целевая версия платформы. Здесь вы можете выбрать версию посовременнее. Eclipse пока считает Android 5 версией **Android 4.X (L Preview)**, но уровень API уже предлагается, как у версии 5.0 — **API 21**.

Остальные параметры можно оставить без изменения — просто нажмите кнопку **Next**.

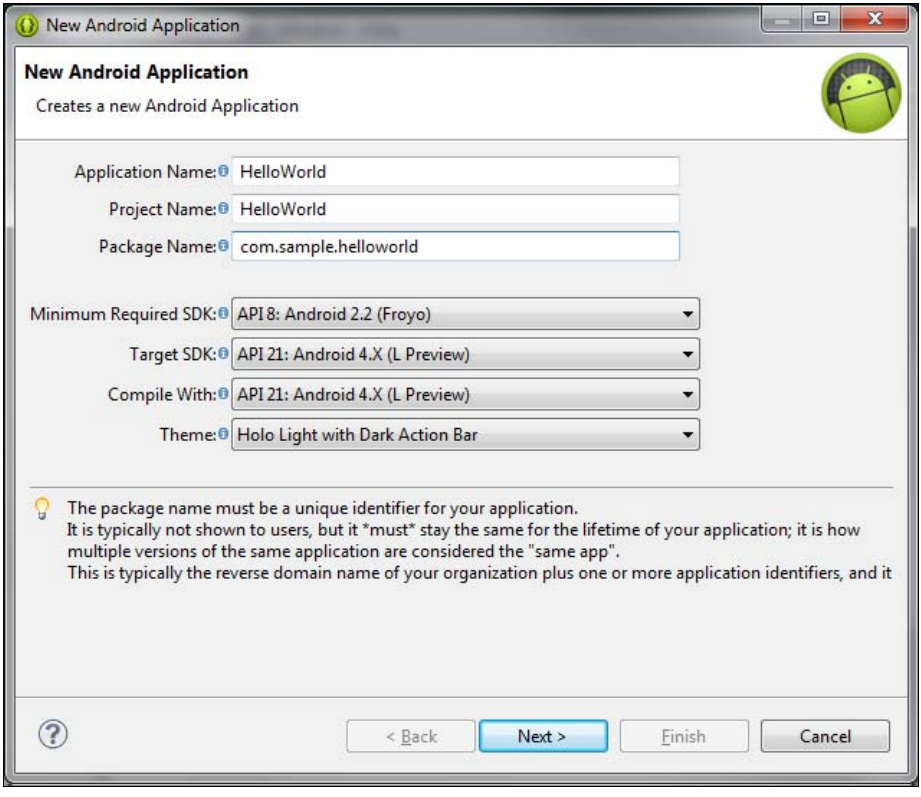


Рис. 3.2. Основные параметры проекта

В следующем окне (рис. 3.3) нужно включить такие параметры:

- **Create custom launcher icon** — создать пользовательскую картинку приложения (которую вы потом можете изменить). Помните, что картинка приложения (значок, пиктограмма) — это лицо вашего приложения. Когда вы будете создавать приложения для Play Маркет, то первое, что увидят ваши пользователи после установки приложения, — это его пиктограмма. Если она окажется невзрачной (или стандартной, что еще хуже), первое впечатление о вашем продукте будет испорчено;
- **Create activity** — создать ли деятельность приложения? Да, создать. Позже вы узнаете, что это такое.

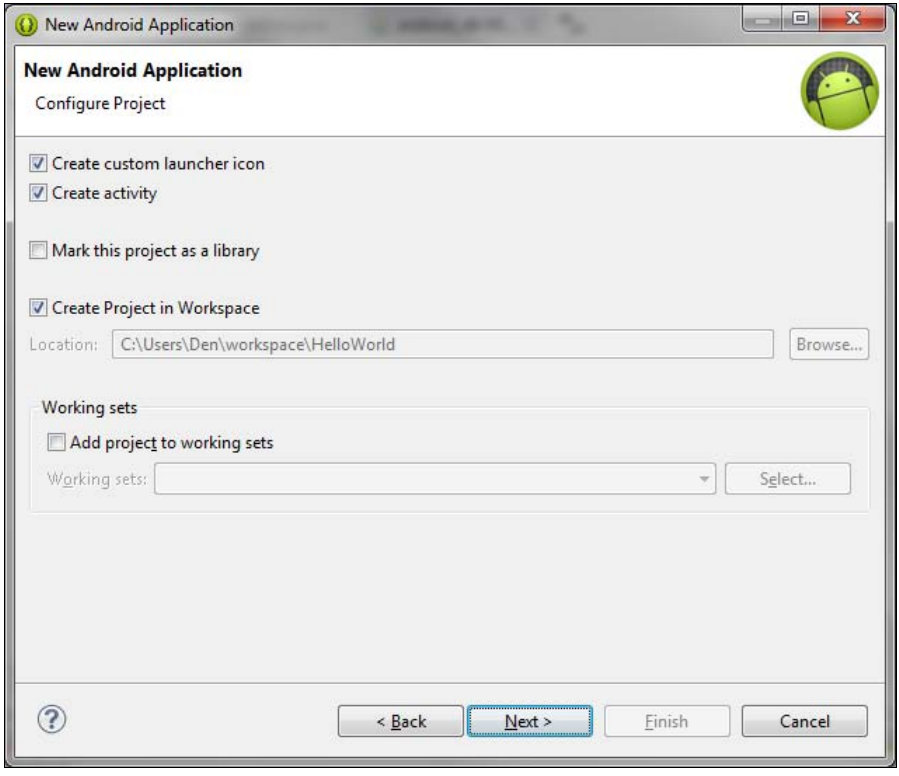


Рис. 3.3. Настройка проекта

Следующий шаг — настройка пиктограммы приложения. Мы его пропустим — пока нас устраивает и стандартная пиктограмма (рис. 3.4), но, как уже было отмечено, для реального продукта вам придется потратить время и, возможно, средства (на привлечение профессионального дизайнера) на создание приемлемой пиктограммы.

Далее нужно выбрать шаблон деятельности (рис. 3.5). Нас устроит **Blank Activity**, а для ваших будущих проектов вы сможете выбрать другой шаблон (есть возможность предварительного просмотра шаблона, позволяющая понять, что означает тот или иной его вариант).

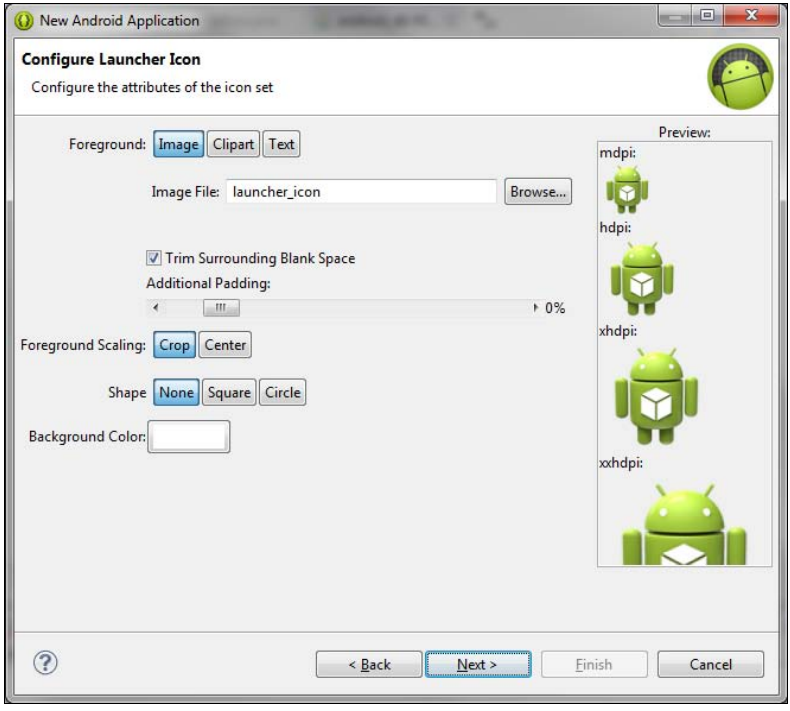


Рис. 3.4. Создание пиктограммы приложения

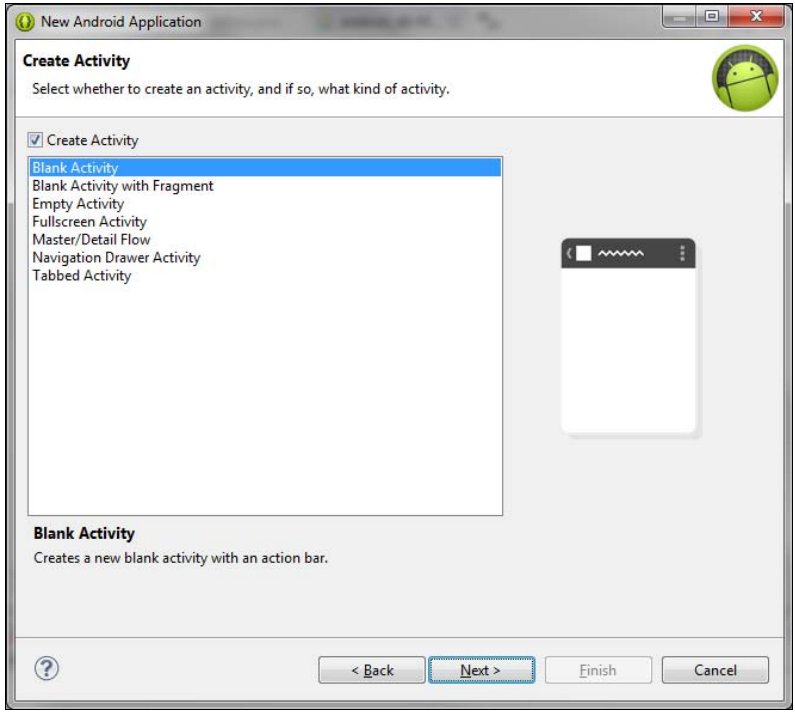


Рис. 3.5. Шаблон деятельности

На следующем шаге требуется ввести имя деятельности. По умолчанию среда предлагает имя: **MainActivity** и имя разметки **(Layout Name)** — **activity_main**. Оставьте эти значения без изменений и нажмите кнопку **Finish** для создания проекта.

Поработав несколько мгновений, компьютер откроет основное окно Eclipse со сгенерированным проектом (рис. 3.6). Как можно видеть, плагин ADT генерирует простейший проект с одним окном и текстовым полем, содержащим строку: **Hello world!**

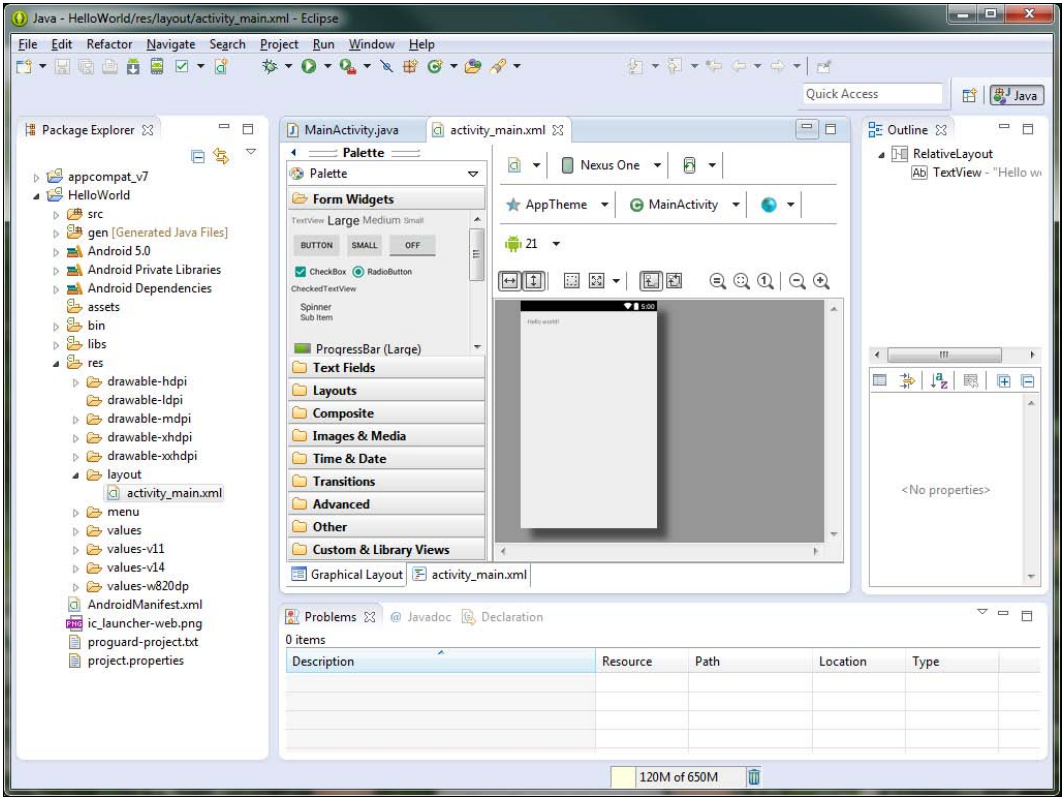


Рис. 3.6. Созданный Android-проект

Вы можете изменить строку по умолчанию, отредактировав файл strings.xml. Для этого в левой части окна Eclipse в области **Package Explorer** перейдите в каталог res\values — в нем и находится этот файл (рис. 3.7). Щелкните по нему двойным щелчком, и откроется редактор ресурсов: **Android Resources** (рис. 3.8). Щелкните по ресурсу **hello_world** (это и есть наша строка) — здесь вы сможете указать для нее произвольное значение. Укажите все, что угодно, — чтобы ваше приложение отличалось от стандартного проекта. Я когда-то добавлял после **World** свое имя — чтобы убедиться, что компилируется и загружается в эмулятор действительно созданная мной программа, а не уже готовые файлы, проект для которых ADT сгенерировал по умолчанию.

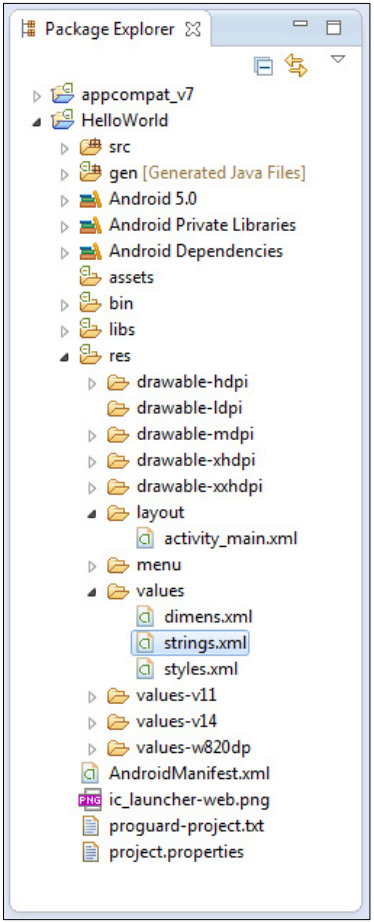


Рис. 3.7. Область **Package Explorer**

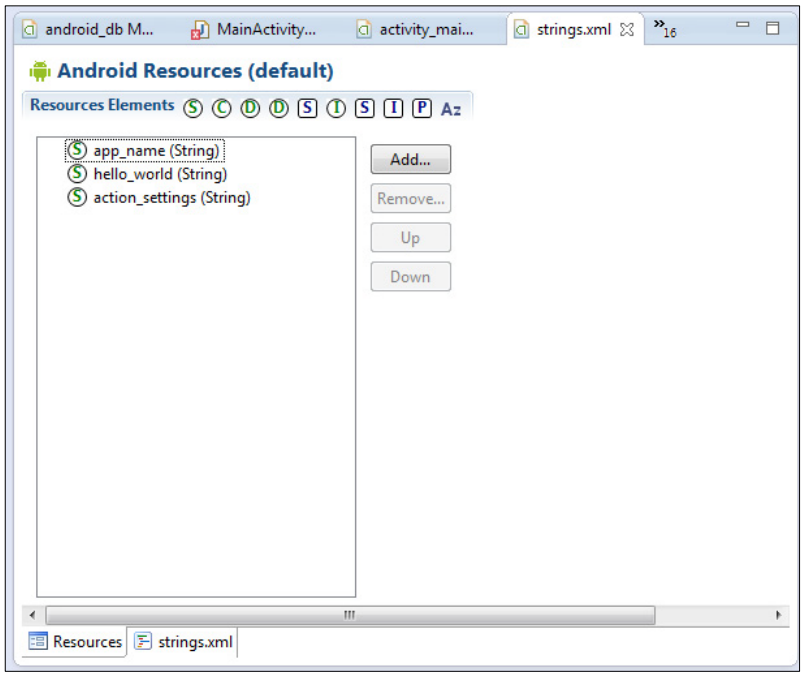


Рис. 3.8. Редактор ресурсов

Если вам удобнее редактировать XML-файл (что характерно для больших проектов), перейдите в окне редактора ресурсов на вкладку **strings.xml** — вы переключитесь в режим редактирования XML-файла (рис. 3.9).

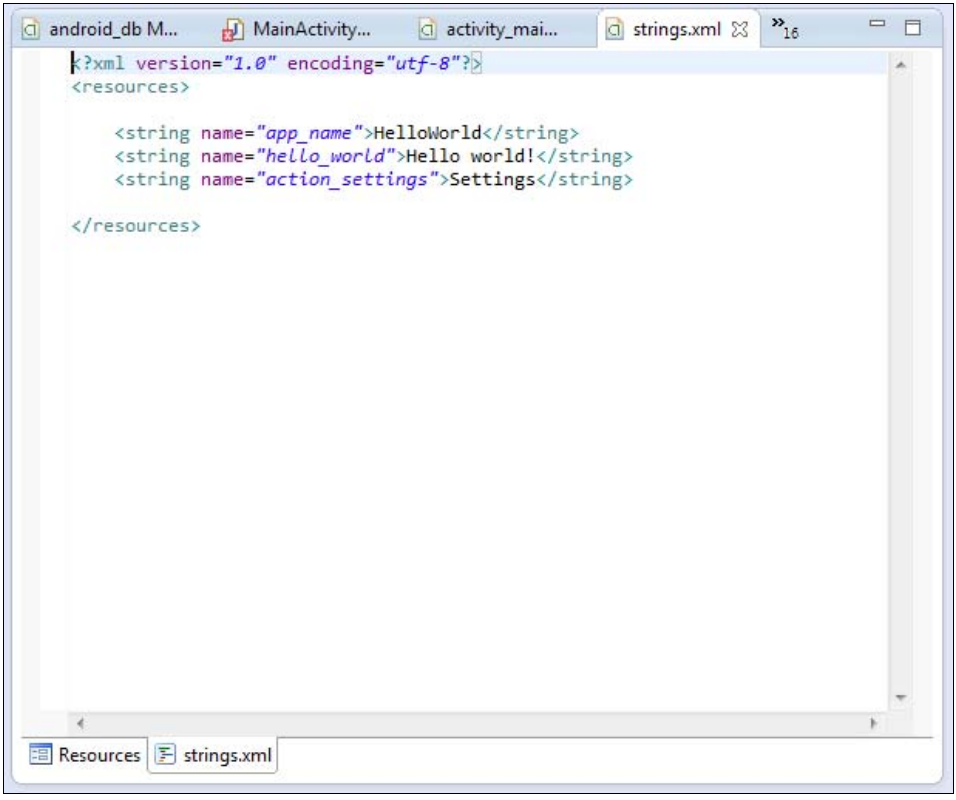


Рис. 3.9. Режим редактирования XML-файла

Теперь запустим проект. Убедитесь, что в области **Package Explorer** выбран именно тот проект, который вы хотите запустить. Нажмите кнопку **Run** на панели инструментов (напоминает кнопку **Play** на пульте дистанционного управления). Откроется окно, в котором нужно выбрать, как следует запустить приложение (рис. 3.10). Выберите **Android Application** и нажмите кнопку **OK**.

Откроется окно эмулятора Android, и нужно будет немного подождать, пока эмулятор загрузится (рис. 3.11). Разблокируйте эмулятор, как если бы это было обычное физическое устройство.

В результате всех ваших «мучений» вы увидите заветную строку в эмуляторе (рис. 3.12).

Поздравляю! Вы только что создали свое первое Android-приложение и запустили его в эмуляторе. Скомпилированный файл HelloWorld.apk будет помещен в каталог workspace\HelloWorld\bin\. Можете поделиться им с вашими друзьями, чтобы они увидели, какой вы мегакрутой программист ☺.

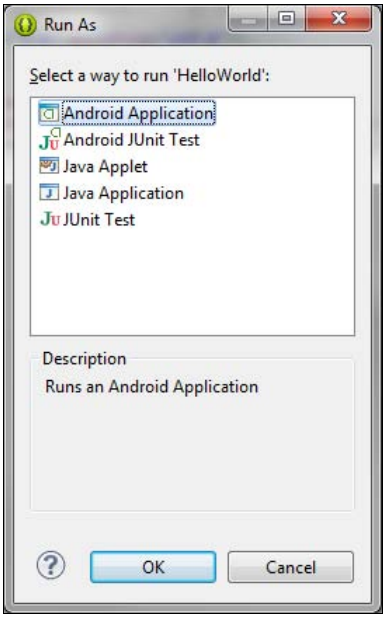


Рис. 3.10. Как запустить приложение?

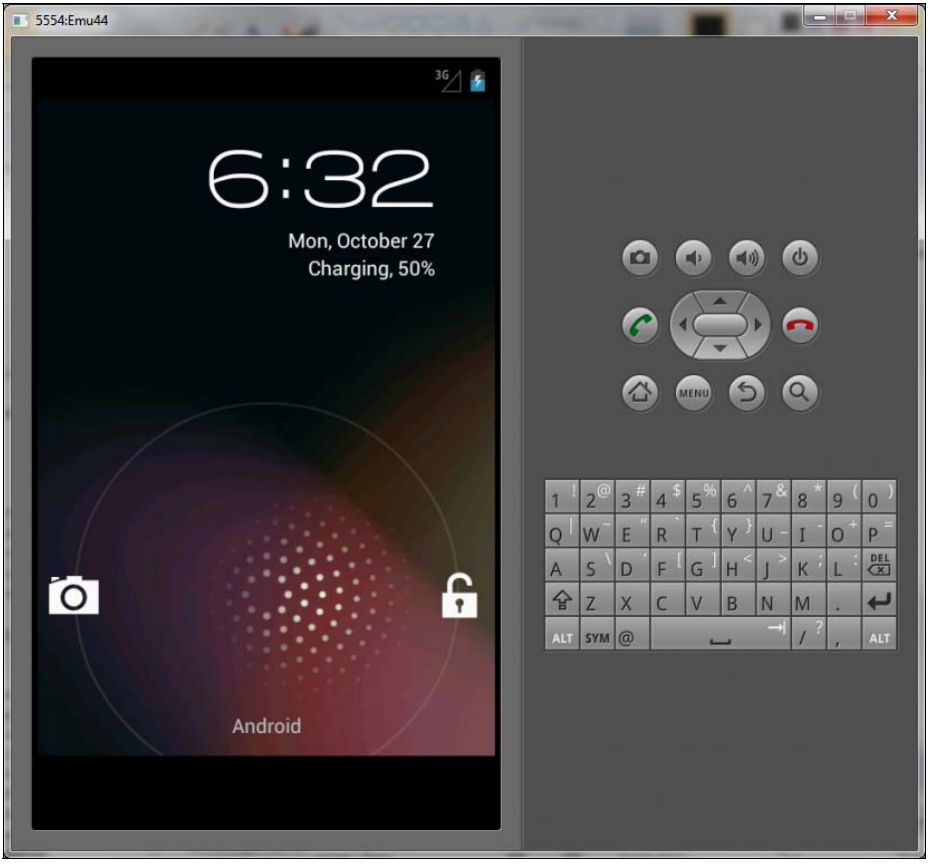


Рис. 3.11. Разблокировка эмулятора

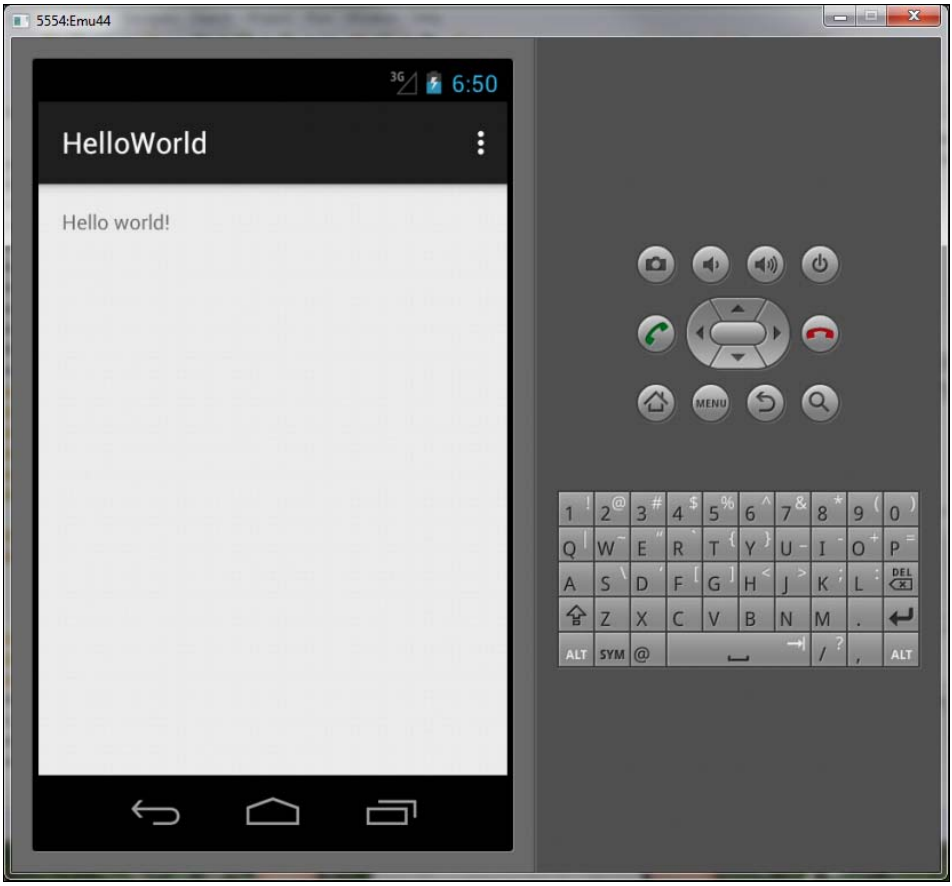


Рис. 3.12. Приложение запущено

При запуске приложения (даже такого простого) нужно помнить о нескольких нюансах:

- запуск эмулятора может длиться около 3–5 минут, на слабых компьютерах — еще дольше. Не нужно думать, что эмулятор завис, и завершать процесс. Позже мы решим проблему медлительности, а пока просто подождите;
- сначала в эмулятор производится загрузка операционной системы Android, а потом уже приложения. Опять-таки: если вы увидели синий фон, подождите немного, и ваше приложение будет запущено;
- в зависимости от версии платформы эмулятор может быть заблокирован. Для его разблокировки нажмите кнопку **Menu**, и вы сможете получить доступ к своему приложению;
- если после успешного запуска приложения вы внесли изменения в проект (например, изменили тестовую строку), но, нажав кнопку **Run**, видите старое приложение (без изменений), закройте окно эмулятора, а в Eclipse выполните команду **Project | Clean**. После этого можно опять нажимать кнопку **Run** для запуска проекта.

3.2. Подробнее о запуске приложения в Android 5.0

Нажать кнопку **Run** и ждать, пока запустится приложение, конечно, просто, но это не наш путь, поскольку мы легких путей не ищем. А что, если нужно запустить приложение на физическом устройстве или на другом виртуальном устройстве? Сейчас мы со всем этим и разберемся. И параллельно поговорим о запуске приложения именно в Android 5.0.

Мне потребовался примерно час времени (а, может, даже больше), чтобы запустить наше простейшее приложение в эмуляторе Android 5.0. И чтобы вы не тратили время на пробы и ошибки, я расскажу вам, что нужно делать.

Во-первых, следует создать само виртуальное устройство. Основное время у меня ушло на эксперименты с его различными параметрами, чтобы подобрать пол-

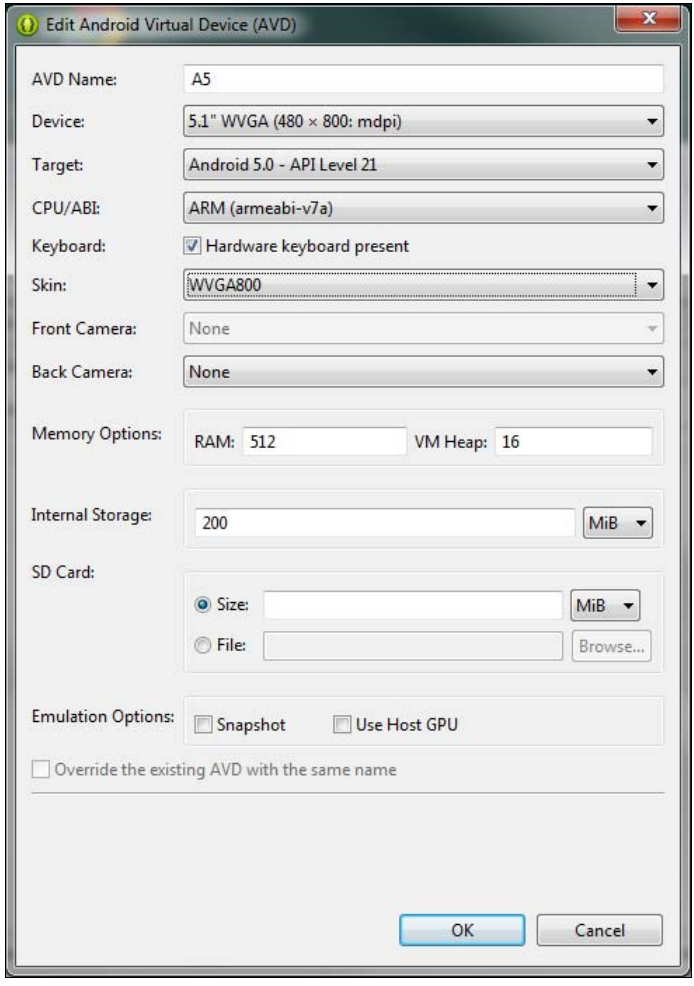


Рис. 3.13. Эмулятор для Android 5.0

ностью рабочие. В некоторых случаях я получал сообщение об ошибке, в некоторых — так и не дождался загрузки эмулятора. Эксперименты показали, что на запуск эмулятора требуется много времени — так, если эмулятор для Android 4, как было сказано ранее, запускается примерно 3–5 минут, то запуск эмулятора Android 5.0 длится около 10 минут. Пример рабочей конфигурации эмулятора приведен на рис. 3.13.

Установив указанные параметры, выполните команду **Project | Properties** и убедитесь, что в разделе **Android** для **Android 5.0** выбран уровень **API 21** (рис. 3.14).

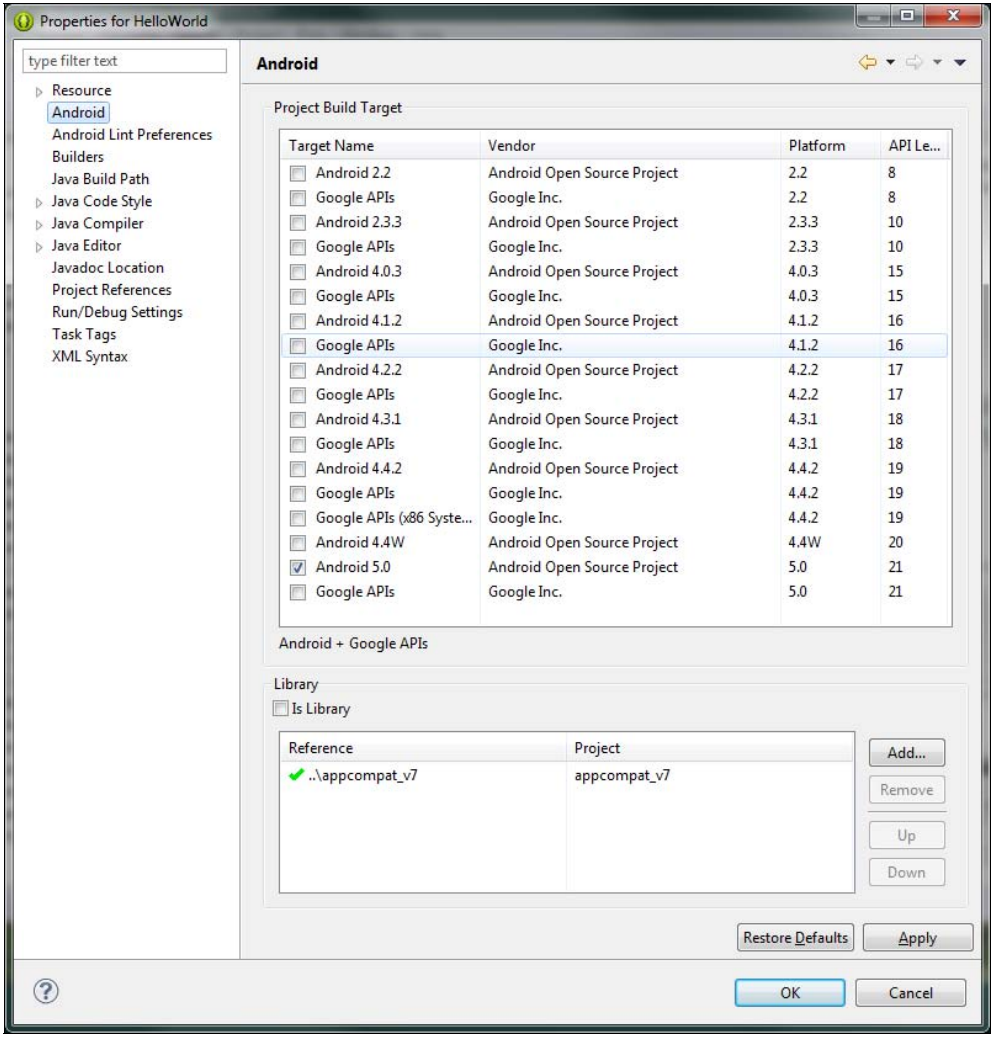


Рис. 3.14. Создаем приложение для Android 5.0

Теперь выполните команду меню **Run | Run Configurations**. Перейдите на вкладку **Target**. Здесь можно задать эмулятор, в котором будет запущено приложение (рис. 3.15). Как можно видеть, я явно указал, что приложение должно быть запуще-

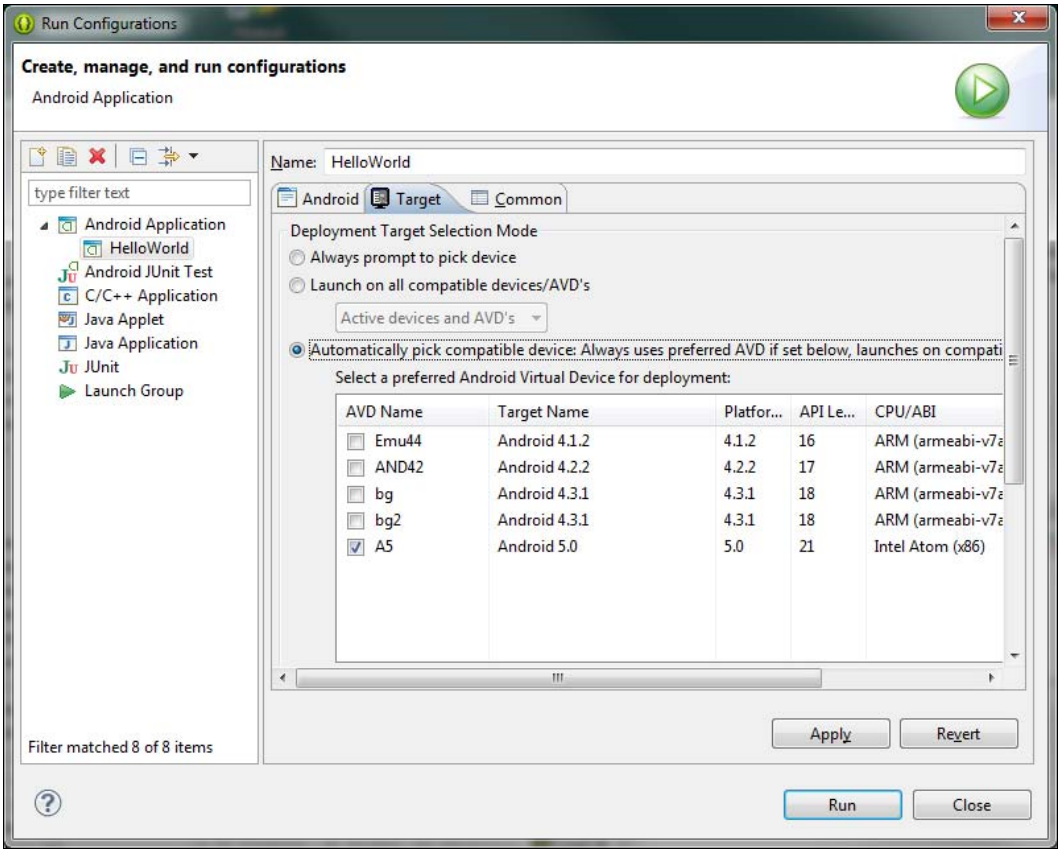


Рис. 3.15. Конфигурации запуска

но в эмуляторе **A5**, который мной создан специально для Android 5.0. Для запуска приложения нажмите кнопку **Run**.

Если выбран вариант **Always prompt to pick device**, Eclipse будет перед каждым запуском приложения спрашивать, на каком эмуляторе или физическом устройстве следует запустить приложение, открывая окно, изображенное на рис. 3.16.

В верхней части этого окна приводится список подключенных физических устройств (в настоящий момент ничего не подключено), а в нижней — список созданных эмуляторов. Выберите устройство и нажмите кнопку **OK**.

Спустя минут 10 вы увидите экран блокировки Android 5.0 (рис. 3.17). «Ухватитесь» за замок и потяните его вверх для разблокировки эмулятора.

Теперь можно «побродить» по эмулятору, наслаждаясь его обновленным интерфейсом (рис. 3.18).

Чтобы убедиться, что у нас именно версия 5.0, нужно запустить в эмуляторе приложение **Dev Settings** и перейти в раздел **About phone** (рис. 3.19). На рис. 3.20 приведено наше приложение, запущенное в Android 5.0.

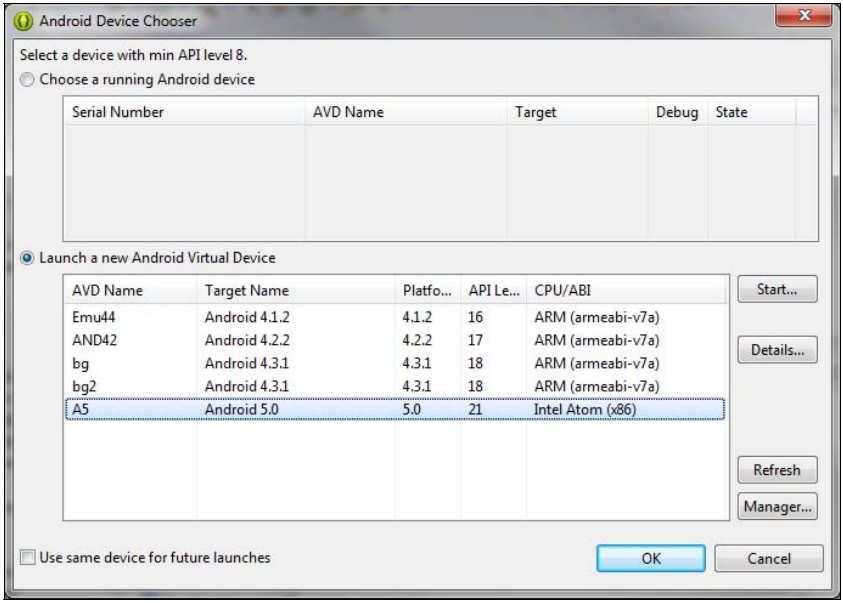


Рис. 3.16. Выбор эмулятора или физического устройства для запуска приложения

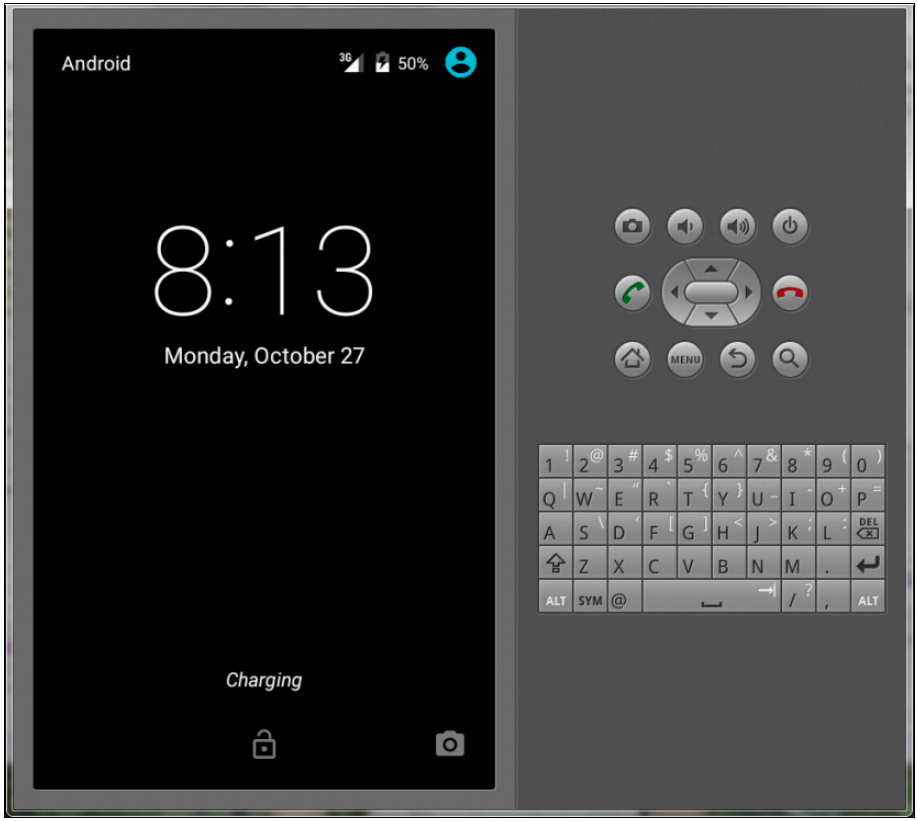


Рис. 3.17. Эмулятор Android 5.0

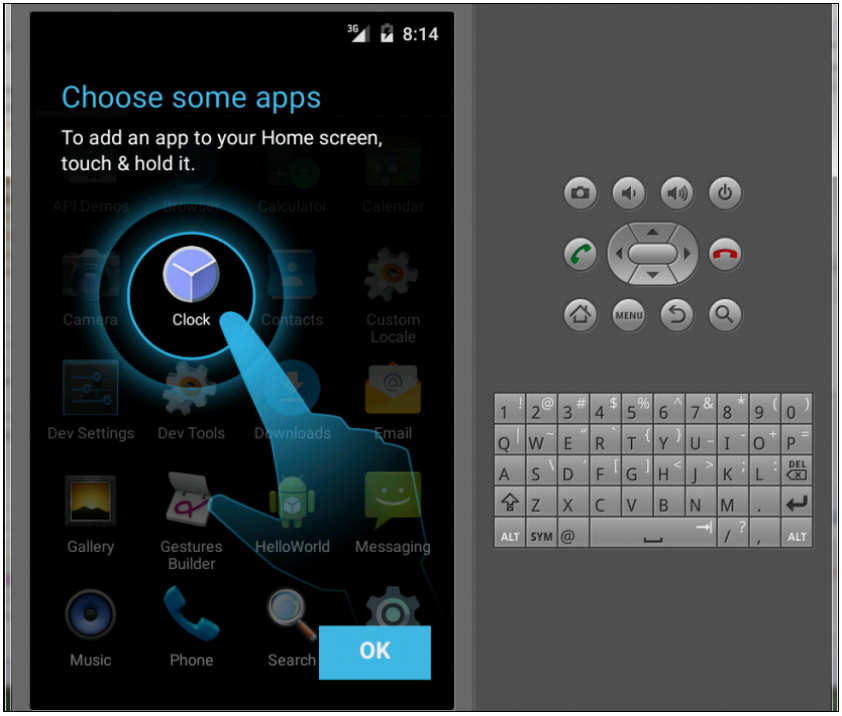


Рис. 3.18. Интерфейс Android 5.0

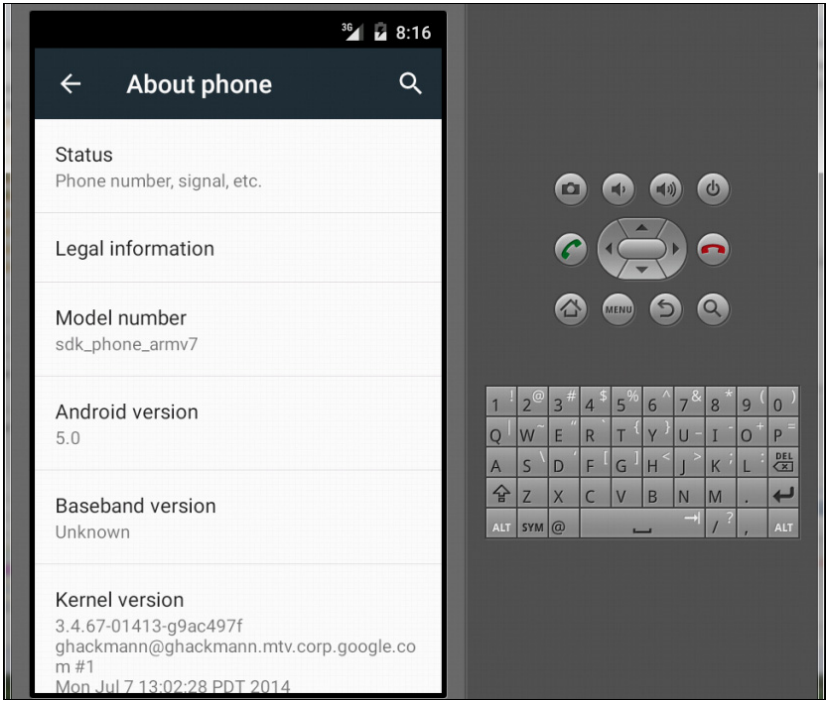


Рис. 3.19. Действительно запущен эмулятор Android 5.0

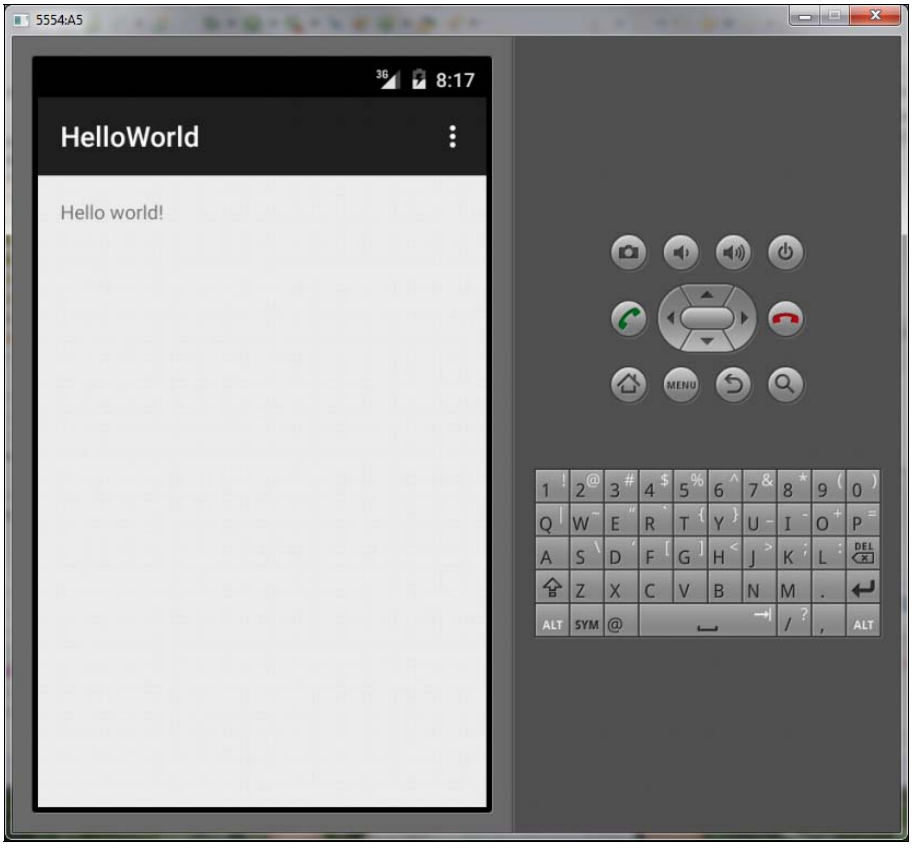


Рис. 3.20. Приложение **HelloWorld** в Android 5.0

3.3. Боремся с медленным запуском эмулятора

Очень сильно раздражает медленный запуск эмулятора — ждать 3–5 минут для запуска приложения надоедает. Как оказалось, решить проблему достаточно просто. Для этого при создании эмулятора нужно включить параметр **Snapshot**, что значительно ускорит загрузку эмулятора.

Первый запуск виртуального устройства Android (AVD) все равно останется медленным — компьютеру нужно некоторое время на создание снапшота («снимка» файловой системы эмулятора), зато все последующие станут практически мгновенными — в среднем запуск эмулятора будет длиться 5–8 секунд, что на фоне 3–5 минут вполне приемлемо.

Если же вы уже создали AVD, перейдите в окно менеджера SDK и AVD (рис. 3.21), выделите ваш AVD и нажмите кнопку **Edit**. В открывшемся окне включите параметр **Snapshot** (рис. 3.22) и нажмите кнопку **OK**.

Выделите в окне менеджера (см. рис. 3.21) устройство, нажмите кнопку **Start**, в открывшемся окне (рис. 3.23) включите параметры **Launch from snapshot** и **Save to**

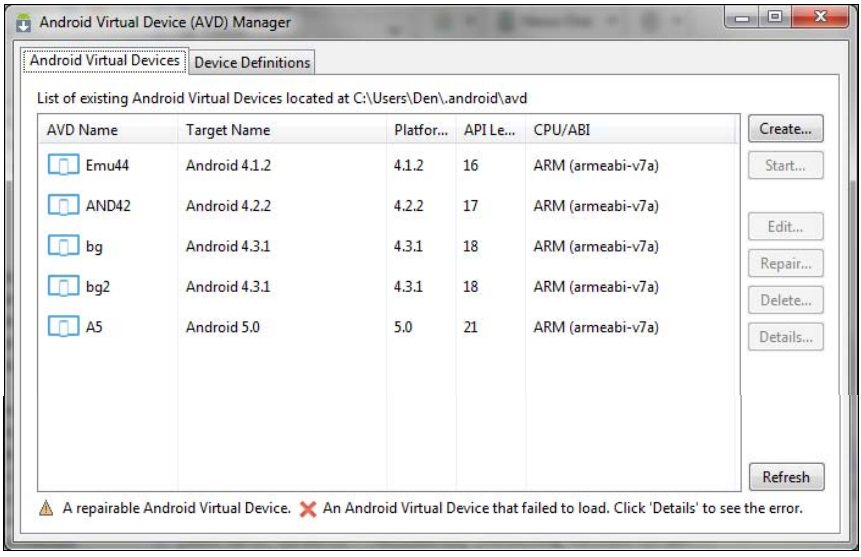


Рис. 3.21. Android SDK and AVD Manager

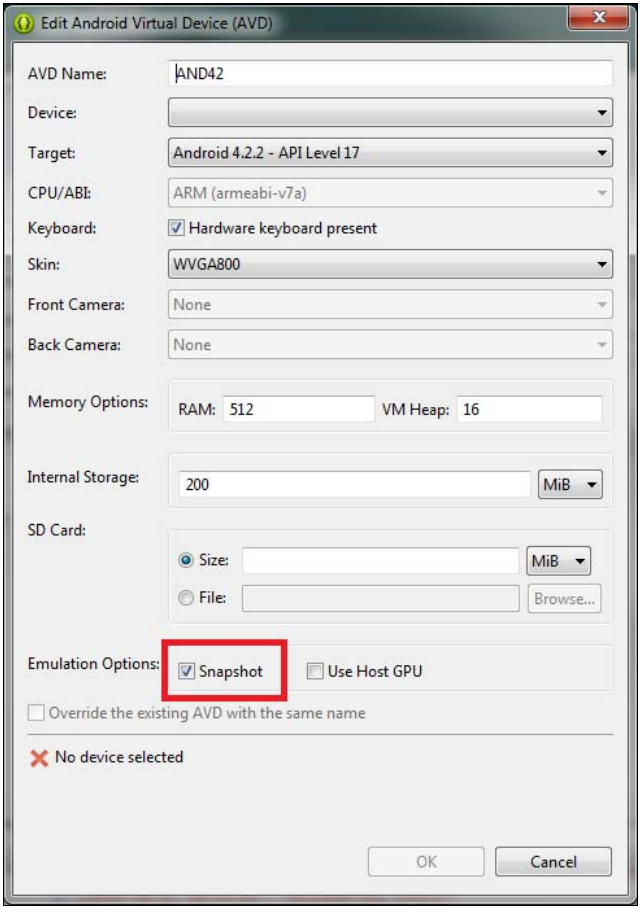


Рис. 3.22. Редактирование виртуального устройства

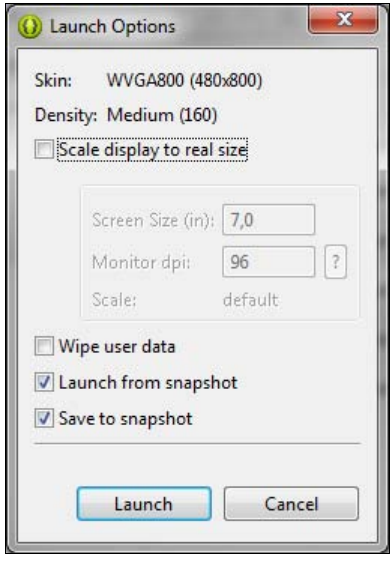


Рис. 3.23. Параметры запуска виртуального устройства

snapshot. Нажмите кнопку **Launch**. Теперь эмулятор будет запускаться значительно быстрее.

3.4. Создание снимка экрана виртуального устройства

Иногда нужно создать снимок экрана своей программы — например, когда вы публикуете программу на Play Маркет или просто хотите похвастаться перед друзьями. Но средствами операционной системы (нажав клавишу <PrintScreen> или даже <Alt>+<PrintScreen>) это не всегда удобно, поскольку если вы эмулируете планшет, то окно эмулятора может не поместиться на вашем экране. Однако в Eclipse есть средства, позволяющие сделать снимок экрана, даже если окно слишком большое.

Итак, для создания снимка экрана виртуального устройства следует в Eclipse выполнить команду **Window | Show View | Other**. Откроется окно (рис. 3.24), в котором нужно выбрать **Devices** и нажать кнопку **OK**.

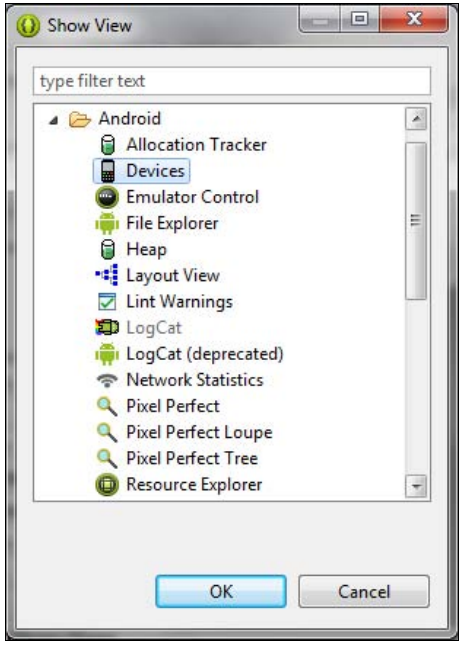


Рис. 3.24. Окно **Show View**

После этого в нижней части окна Eclipse появится список Android-устройств: в нем будут перечислены все виртуальные и реальные (если таковые подключены к компьютеру) устройства. Выберите устройство, снимок экрана которого вы желаете получить (разумеется, устройство должно быть активным), и нажмите кнопку создания снимка экрана — **Screen Capture** (на рис. 3.25 эта кнопка отмечена стрелкой). Откроется окно, в котором нужно нажать кнопку **Save** — для сохранения снимка экрана на жесткий диск, для закрытия окна нажмите кнопку **Done**.

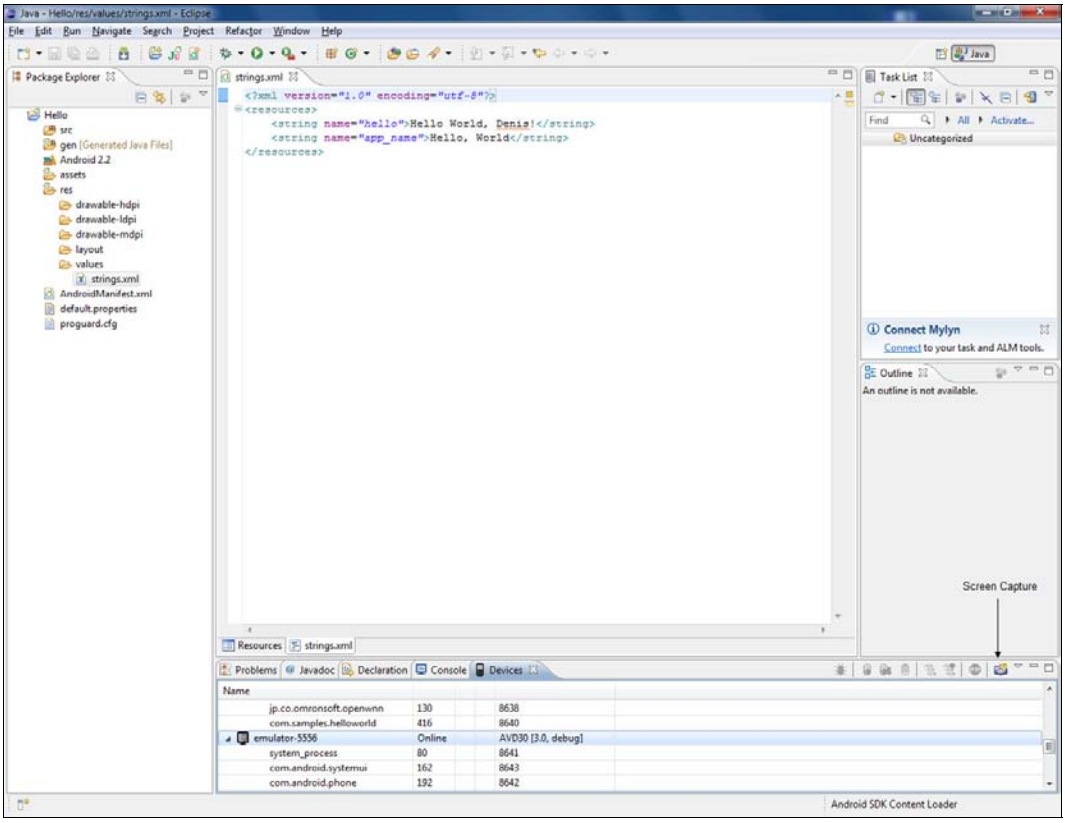


Рис. 3.25. Список устройств и кнопка **Screen Capture**

ЧАСТЬ II

Базовое программирование для Android

ГЛАВА 4

Основы построения приложений

4.1. Структура Android-проекта

В прошлой главе мы создали и запустили первое Android-приложение. Ничего, что оно было слишком простым — главное, мы разобрались, как его можно запустить и как преодолеть некоторые трудности, с которыми вы можете столкнуться при запуске эмулятора Android.

Просмотреть структуру Android-проекта можно в окне **Package Explorer** основного окна Eclipse (рис. 4.1). Структура проекта меняется в зависимости от уровня API — фактически, от версии платформы Android, для которой ведется разработка. На рис. 4.1 изображена структура проекта для платформы 5.0. Обратите внимание: если вы еще не компилировали (не запускали[1]) приложение, то у вас не будет каталога bin.

Дело в том, что в этом каталоге содержится откомпилированный код Java-классов вместе с файлами ресурсов и данными. Код, данные и ресурсы — все это упаковывается в файл с расширением apk. Именно этот файл используется для распространения вашей программы и установки ее на мобильном устройстве. Да, и именно его после создания вашей программы нужно отправить вашим друзьям, выложить на своем сайте, опубликовать на Play Маркет и т. д. Полный путь к APK-файлу выглядит так:

```
рабочее_пространство\имя_проекта\bin
```

По умолчанию для рабочего пространства используется каталог C:\Users\ <*пользователь*>\workspace\ (или /home/<пользователь>/workspace/ в Linux).

Самый важный каталог — это каталог res, представляющий собой каталог ресурсов приложения. В этом каталоге находятся следующие подкаталоги:

---

[1] Компиляция и запуск приложения — это не одно и то же. При запуске приложения производится его компиляция (если приложение не было откомпилировано или с момента последней компиляции в проект были внесены изменения), загрузка его в эмулятор или в физическое устройство и непосредственно сам запуск приложения.

- *res*/drawable* — содержат изображения, адаптированные для различных разрешений экрана. По умолчанию в этих каталогах будут находиться файлы icon.png разных разрешений;
- res/layout — содержит разметку элементов пользовательского интерфейса приложения;
- res/menu — этот каталог не создается плагином ADT, его нужно создавать самостоятельно, если в этом есть необходимость. В этом каталоге содержатся XML-файлы меню;
- res/values* — различные значения, например, строковые значения, массивы и т. д. В предыдущей главе мы уже познакомились с файлом strings.xml, содержащим строковые значения;
- res/xml — другие XML-файлы (этот каталог также не создается плагином ADT и его нужно создать вручную в случае необходимости).

Внутри указанных каталогов могут быть только файлы, нельзя создать подкаталоги внутри подкаталогов каталога res.

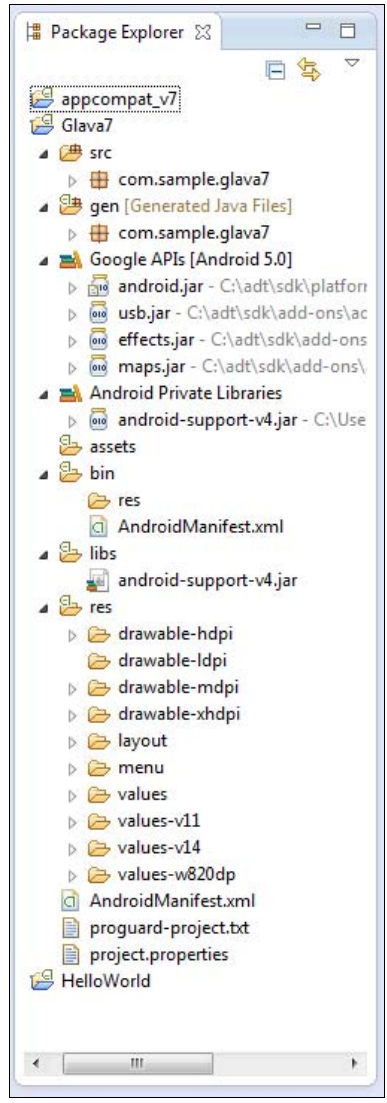


Рис. 4.1. Структура Android-проекта

Примечание

Крайне нежелательно вручную (с помощью файлового менеджера) изменять структуру и содержимое каталога проекта. Единственное, что вы можете сделать — это скопировать APK-файл из каталога *bin* проекта, добавить новые файлы в подкаталоги каталога *res* (потом, чтобы «увидеть» эти файлы в проекте, нужно нажать клавишу <F5> для его обновления или перезапустить Eclipse). Изменять структуру и содержимое файлов вне среды Eclipse можно, если вы действительно понимаете, что делаете, и это невозможно сделать из Eclipse по каким-либо причинам.

В следующей главе мы подробно поговорим о создании пользовательского интерфейса. А пока только загляните в каталог res/layout.

Разметку интерфейса пользователя можно создать как визуально (рис. 4.2), так и путем редактирования файла activity_main.xml вручную (листинг 4.1).

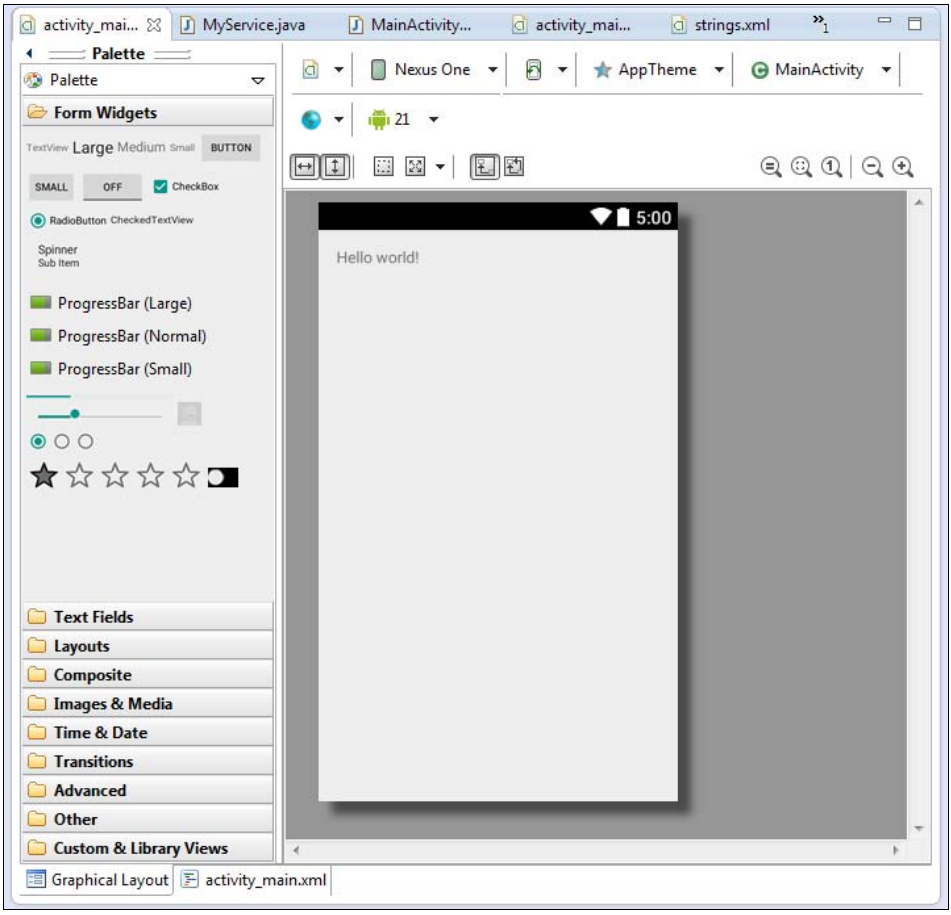


Рис. 4.2. Графическая разметка интерфейса пользователя (платформа 5.0)

Листинг 4.1. Файл разметки activity_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.sample.helloworld.MainActivity" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

</RelativeLayout>
```

На рис. 4.2 представлен редактор среды Eclipse, который используется как для визуального редактирования разметки, так и для редактирования обычных файлов в текстовом режиме. В верхней области редактора имеются вкладки, служащие для переключения между редактируемыми объектами. В текущий момент редактируются: файл разметки **activity_main.xml**, файл класса **MyService.java**, Java-код активности **MainActivity.java**, файл со строковыми константами **strings.xml** и другие объекты. Открыта вкладка **activity_main.xml** (название соответствует названию редактируемого объекта). Разметку можно редактировать как визуально (нижняя вкладка редактора: **Graphical Layout**), так и в текстовом режиме — путем редактирования XML-файла разметки (нижняя вкладка редактора: **activity_main.xml**).

Оба способа (графический и текстовый) равносильны. Когда вы редактируете разметку графически, Eclipse автоматически создает содержимое XML-файла разметки. Поначалу вам будет больше нравиться графический способ, но затем вы придете к редактированию именно в текстовом режиме. Пользуйтесь тем режимом, который вам больше подходит.

В графическом режиме в левой области окна редактора выводится палитра компонентов, которая существенно облегчает выбор и размещение их на форме приложения.

Обратите также внимание на кнопки масштаба в верхнем правом углу окна редактора и на кнопку с изображением Android и числом (в настоящий момент там число **21**). Эта кнопка, открывающая список, позволяет выбрать платформу, для которой создается приложение.

Кроме того, левее кнопки списка с типом эмулируемого устройства (сейчас это **Nexus One**) находится кнопка списка, позволяющая выбрать ориентацию экрана (книжная/альбомная).

Текстовая разметка задается элементом `TextView` (см. листинг 4.1). Обратите внимание — в нем выводится текстовое значение `@string/hello_world`. Значение `hello_world` определено в файле res/values/strings.xml (листинг 4.2).

Листинг 4.2. Файл res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">HelloWorld</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>

</resources>
```

После компиляции проекта в каталоге gen/<*имя пакета*> создается файл R.java (листинг 4.3), который используется для обращения программы к своему каталогу ре-

сурсов. В Android 5.0 файл R.java очень длинный, поэтому в листинге 4.3 приведено только его начало.

Листинг 4.3. Файл R.java

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found.  It
 * should not be modified by hand.
 */

package com.sample.helloworld;

public final class R {
    public static final class anim {
        public static final int abc_fade_in=0x7f040000;
        public static final int abc_fade_out=0x7f040001;
        public static final int abc_slide_in_bottom=0x7f040002;
        public static final int abc_slide_in_top=0x7f040003;
        public static final int abc_slide_out_bottom=0x7f040004;
        public static final int abc_slide_out_top=0x7f040005;
    }
    public static final class attr {
```

Редактировать или изменять этот файл смысла нет, поскольку при следующей компиляции проекта среда его снова перезапишет. При добавлении нового ресурса среда также должным образом автоматически изменит файл R.java.

Среда также автоматически создает Java-файл для главного окна приложения (в частности, для приложения helloworld он находится в каталоге src\com\samples\helloworld\). Как можно догадаться, конкретное имя каталога, в котором находится этот файл, зависит от имени пакета. Если бы вы указали при создании проекта имя пакета com.samples.hello, то файл основного окна приложения нужно было искать в каталоге src\com\samples\hello\. Имя Java-файла зависит от названия проекта, в нашем случае файл называется MainActivity.java (листинг 4.4). Если в вашем приложении предусмотрено несколько окон, будет создан отдельный Java-файл для каждого окна.

Листинг 4.4. Файл MainActivity.java

```
package com.sample.helloworld;

import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
```

```
import android.view.Menu;
import android.view.MenuItem;


public class MainActivity extends ActionBarActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }


    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is
present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();
        if (id == R.id.action_settings) {
            return true;
        }
        return super.onOptionsItemSelected(item);
    }
}
```

Нам осталось рассмотреть только один файл, который называется одинаково для любого Android-приложения: AndroidManifest.xml. В этом файле задается конфигурация приложения: описываются его компоненты, подключаются библиотеки и т. д. Указываются в этом файле и название приложения и версия API. Код файла AndroidManifest.xml представлен в листинге 4.5.

Листинг 4.5. Файл AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.sample.helloworld"
    android:versionCode="1"
    android:versionName="1.0" >
```

```
    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="21" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Позже мы этот файл рассмотрим подробнее, а пока вам достаточно знать, для чего он используется.

4.2. Компоненты Android-приложения

Если вы программировали на языке C, то знаете, что у C-приложения есть одна точка входа (entry point) — функция `main()`. У Android-приложения нет единственной точки входа. Android-приложение состоит из компонентов, которые система может запускать, когда ей это необходимо. Одно приложение может также вызывать компоненты другого приложения, если, конечно, приложение разрешило использование своих компонентов.

Всего существует четыре типа компонентов:

- `Activity` — деятельность;
- `Service` — служба;
- `ContentProvider` — контент-провайдер;
- `BroadcastReceiver` — приемник широковещательных намерений.

> ***Примечание***
>
> Намерения — это асинхронные сообщения, активирующие деятельности, службы и приемники широковещательных намерений. Намерение — это объект класса `Intent`, представляющий собой содержание сообщения.

Рассмотрим эти компоненты подробнее.

- Начнем с первого компонента — `Activity`. Деятельность (активность) представляет собой визуальный интерфейс пользователя, т. е., попросту говоря, окно. Окно обычно заполняет весь экран мобильного устройства, но может быть меньших размеров, чем экран устройства.

  Если у приложения несколько окон, то у него будет несколько деятельностей. Каждая деятельность независима от других, при открытии новой деятельности (нового окна) работа предыдущей деятельности приостанавливается.

- У следующего компонента приложения — службы (`Service`) — нет графического интерфейса пользователя. Служба[1] выполняется в фоновом режиме до тех пор, пока не завершит свою работу.

  Другие приложения могут подключаться к службе. После подключения к службе вы можете использовать функции, предоставляемые службой. Приложение может также запустить или остановить службу.

- Для получения информации о внешних событиях и реакции на них служит компонент `BroadcastReceiver`. Источником события могут быть службы и другие приложения. Приложение может реагировать на любые события, которые оно посчитает важным.

  У приемников широковещательных намерений нет пользовательского интерфейса, но в ответ на событие они могут запустить деятельность, т. е. отобразить окно.

- Следующий компонент приложения — контент-провайдер (`ContentProvider`). Он служит для предоставления доступа к данным программы другим приложениям. Данные программы могут храниться как в файловой системе, так и в базе данных SQLite (могут быть и другие способы хранения данных).

4.3. Процессы в ОС Android

Если хотя бы один компонент приложения будет востребован, ОС Android запустит процесс, содержащий единственный основной поток для выполнения. Все компоненты приложения работают в этом процессе и потоке.

Однако это поведение по умолчанию. Вы можете сделать так, чтобы компоненты работали в других процессах и порождали дополнительные потоки, т. е. создать многопотоковое приложение. В случае нехватки памяти операционная система может завершить процесс, если нужно выделить память более важным процессам. Компоненты, выполняющиеся в этом процессе, будут уничтожены.

Но какой процесс можно уничтожить, а какой нельзя? Согласитесь, будет некрасиво, если Android уничтожит процесс, с которым пользователь работает в данный момент. Поэтому Android оценивает важность процесса с точки зрения пользовате-

[1] Подробно о службах мы поговорим в *главе 11*.

ля. Операционная система никогда не уничтожит видимый пользователем компонент процесса, но может завершить процесс с невидимыми деятельностями, которые более не отображаются на экране. Также может быть завершена служба, выполняющаяся в другом процессе.

Какой процесс будет завершен в первую очередь, зависит от его приоритета. У активного процесса критический приоритет, поэтому он не может быть завершен, у видимого и сервисного процесса высокий приоритет, поэтому система, скорее всего, его тоже не завершит. Главными кандидатами на завершение являются процессы с низким приоритетом — фоновые и пустые процессы. Сначала система удалит пустые процессы, затем — фоновые.

Думаю, с приоритетами все ясно, но нужно разобраться с терминологией. *Активным процессом* (Foreground Process) называется процесс, с которым в данный момент работает пользователь. Система считает процесс активным, если процесс выполняет деятельность, с которой работает пользователь, или же процесс выполняет службу, которую использует активная деятельность. Заметьте, что процесс А может выполнять службу, которая используется деятельностью, выполняющейся процессом Б. С этой деятельностью работает пользователь, поэтому процессы А и Б считаются активными.

Также процесс считается активным, если он имеет объект Service и выполняется один из методов обратного вызова, который определен в этом объекте. Если процесс имеет объект Broadcast Receiver и выполняется его метод обратного вызова для приема намерения, то процесс тоже считается активным.

Активные процессы система может удалить только в том случае, если памяти осталось так мало, что все активные процессы уже не могут выполняться одновременно.

Видимым (Visible Process) называется процесс, не имеющий никаких приоритетных компонентов. Например, процесс А запустил деятельность не на весь экран. Пользователь работает с процессом Б, который также отображает активность. Активность процесса А не имеет фокуса, но все еще видна пользователю, поэтому процесс А называется видимым. Также видимым считается процесс, выполняющий службу, которая связана с деятельностью, находящейся на переднем плане, но не активна или частично закрыта другой деятельностью.

Сервисным (Service Process) называется процесс, в котором выполняется служба и который не относится к активным и видимым. Обычно сервисные процессы не имеют интерфейса, но они выполняют задания, необходимые пользователю, — например, сервисным процессом может считаться фоновая работа проигрывателя. Поэтому система старается сохранить сервисные процессы.

Фоновым (Background Process) считается процесс, в котором выполняется деятельность, которая в данный момент не видна пользователю. Фоновый процесс может быть уничтожен в любой момент, если понадобилась память активному, сервисному или видимому процессу.

Пустой (Empty Process) процесс вообще не содержит активных компонентов приложения. Система использует такие процессы в качестве кэша — для более быст-

рой последующей инициализации компонентов приложения. Однако если понадобится память, то такие процессы будут удалены в первую очередь.

4.4. Подробнее о файле *AndroidManifest.xml*

Перед запуском компонента операционная система должна убедиться, что он существует. Для описания компонентов используется файл AndroidManifest.xml. У каждого Android-приложения есть свой файл манифеста — даже у самых простых приложений.

В файле манифеста указываются имя Java-пакета приложения (имя пакета используется в качестве уникального идентификатора приложения), разрешения, которыми должно обладать приложение для обращения к защищенным частям API (нужно для взаимодействия с другими приложениями), библиотеки, необходимые для выполнения этого приложения, и т. д.

Но самое главное — это описание компонентов приложения, а именно деятельностей, служб, контент-провайдеров и т. д. Объявления в файле манифеста позволяют операционной системе узнать, чем является тот или иной компонент и при каких условиях он должен быть запущен.

В среду Eclipse встроен редактор файла манифеста. Для его запуска щелкните двойным щелчком на файле AndroidManifest.xml в окне **Package Explorer** (рис. 4.3).

Основными элементами файла AndroidManifest.xml являются `<manifest>`, `<uses-dsk>` и `<application>`. Остальные элементы этого файла используются при необходимости. При желании вы можете редактировать элемент `<application>` с помощью встроенного редактора Eclipse (рис. 4.4).

Элемент `<manifest>` является корневым элементом файла AndroidManifest.xml. По умолчанию плагин ADT генерирует элемент `<manifest>` так:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.samples.helloworld"
      android:versionCode="1"
      android:versionName="1.0">
```

- Первый атрибут (`xmlns:android`) определяет пространство имен Android, этот атрибут всегда остается неизменным для всех Android-приложений.
- Второй атрибут (`package`) задает имя пакета приложения, которое вы указали при создании проекта.
- Третий атрибут (`android:versionCode`) задает внутренний номер версии.
- Атрибут `android:versionName` указывает пользовательскую версию. Вы можете указать как строку, так и указатель на строковый ресурс.

Элемент `<uses-dsk>` определяет минимальный уровень API, необходимый для запуска приложения, — т. е., по сути, определяет версию Android:

```
<uses-sdk android:minSdkVersion="21" />
```

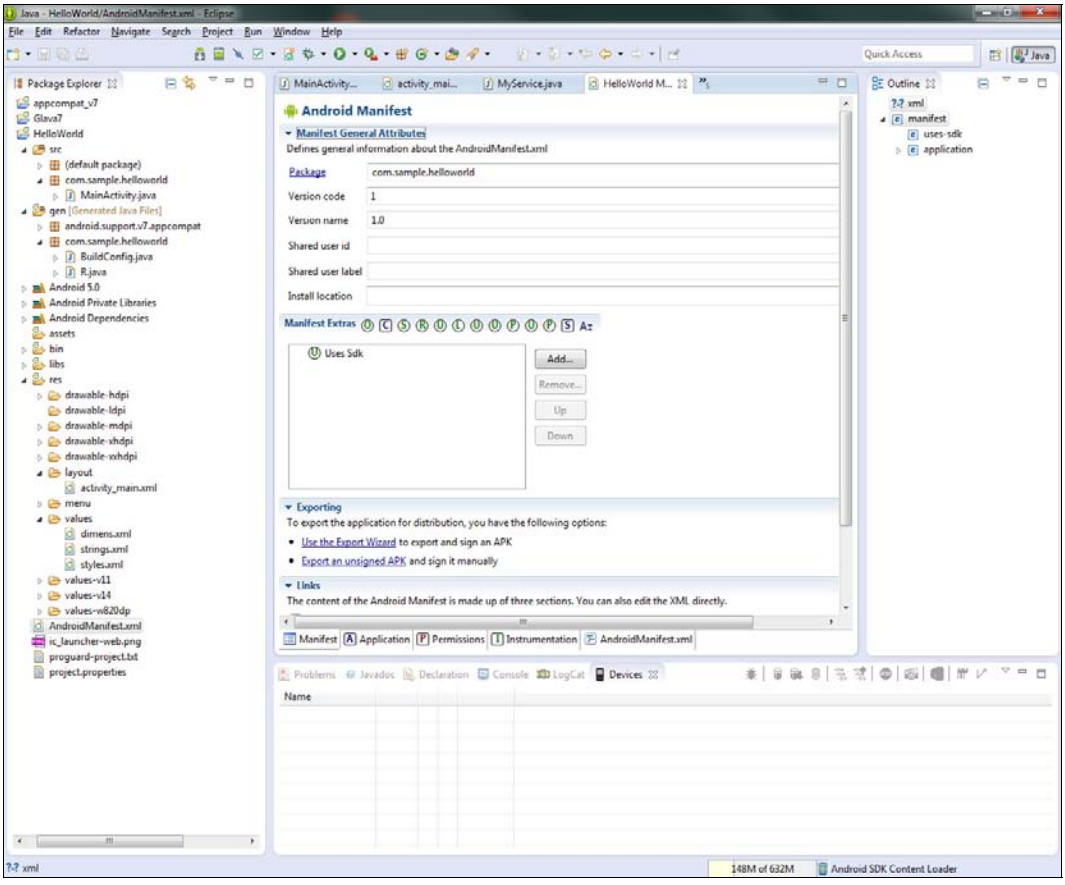


Рис. 4.3. Редактор файла манифеста

Обычно задается один параметр этого элемента — `android:minSdkVersion`. Можно также задать элемент `targetSdkVersion`, задающий целевую платформу приложения.

Самый главный элемент файла манифеста — `<application>`. В этом элементе описываются:

- `<activity>` — деятельность;
- `<activity-alias>` — псевдоним для деятельности;
- `<service>` — служба;
- `<receiver>` — приемник намерений;
- `<provider>` — контент-провайдер;
- `<uses-library>` — подключаемые библиотеки.

В простом приложении, которое мы создали в предыдущей главе, элемент `<application>` представлен так:

```
<application
    android:allowBackup="true"
```

```
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
```

В нашем случае указаны атрибуты `android.icon` (пиктограмма приложения), `android.label` (название приложения) и `android.theme` (тема оформления приложения).

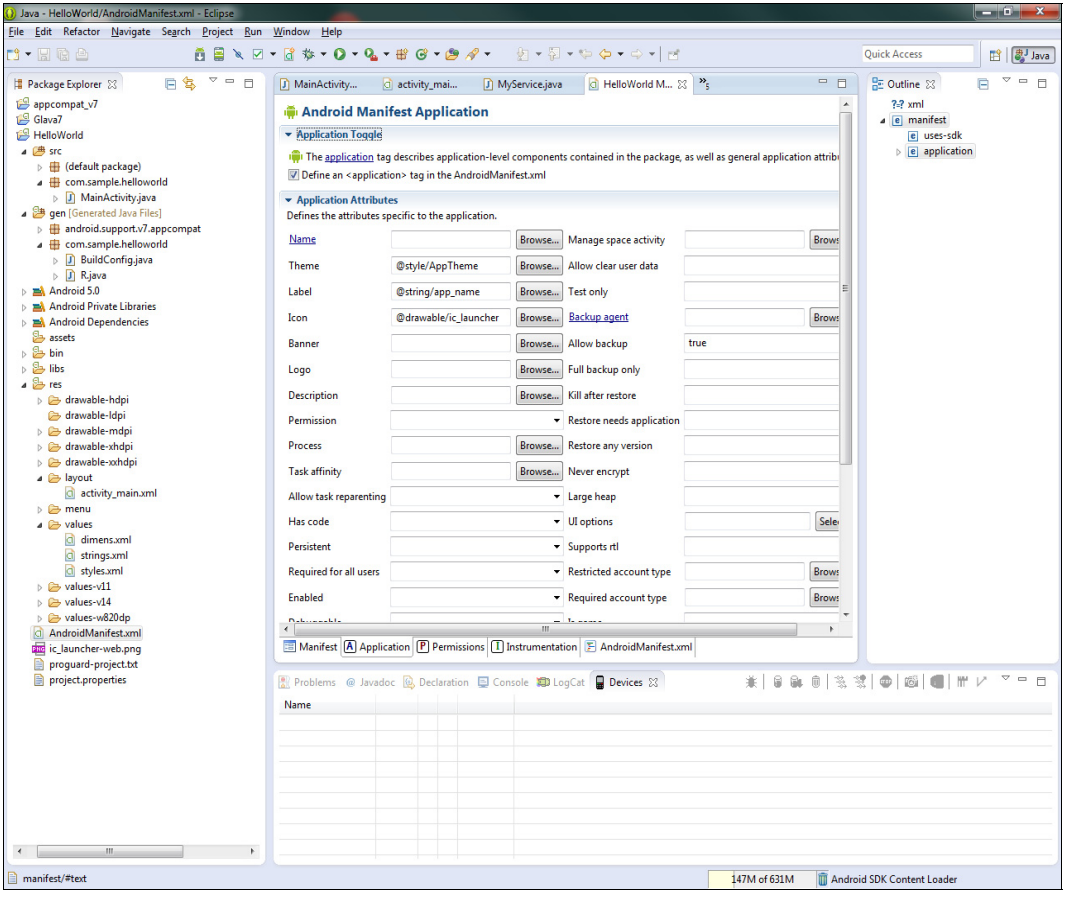


Рис. 4.4. Редактор элемента `<application>` в Eclipse

Элемент <activity> описывает деятельность так:

- атрибут android:name задает имя класса, которое должно включать полное обозначение пакета, но поскольку имя пакета уже определено в <manifest>, то имя класса можно записывать в сокращенном виде, что и делает плагин ADT;
- атрибут android:label задает текстовую метку, которая будет показана пользователю.

Кроме этих атрибутов элемент <activity> может содержать множество других, определяющих разрешение экрана, его ориентацию и т. д.

Объявляя деятельности вручную, помните, что если деятельность не описана в элементе <manifest>, то пользователь не сможет ее увидеть.

Следующий атрибут — <intent-filter>. Он определяет типы намерений, на которые будут отвечать деятельность, сервис или приемник намерений. Фильтр намерений (как раз это и есть атрибут <intent-filter>) определяет, что может сделать деятельность или служба. В этом элементе могут быть дочерние элементы <action>, <category>, <data>:

- элемент <action> добавляет действие к фильтру намерений. В фильтре намерений могут быть несколько элементов <action>, но, как минимум, должен быть один такой элемент;
- в элементе <category> определяется категория компонента, которую должно обработать намерение. Как правило, задаются строковые константы (в качестве имени намерения), определенные в классе Intent, например:

```
<category android:name="android.intent.category.LAUNCHER" />
```

- элемент <data> используется, чтобы добавить спецификацию данных к фильтру намерений. Спецификация может быть URL, типом данных или и URL, и типом данных.

Мы рассмотрели основные элементы файла манифеста. Остальные элементы будут описаны далее по мере необходимости.

ГЛАВА 5

Разработка интерфейса пользователя

5.1. Разметка интерфейса

5.1.1. Файл разметки и редактор разметки

Графический интерфейс пользователя формируется с помощью объектов `View` (представление) и `ViewGroup` (группа представлений). При этом класс `ViewGroup` является дочерним классом для `View`.

Класс `View` служит основой для подклассов, которые называются *виджетами*. Виджеты — это элементы пользовательского интерфейса: текстовые поля, кнопки и т. п.

Разметка интерфейса пользователя — это процесс размещения элементов интерфейса (виджетов) для конкретного окна приложения (для конкретной деятельности). Разметку можно выполнить двумя способами. Первый заключается в редактировании XML-файла окна приложения. Для главного окна приложения этот файл называется res/layout/main.xml. Такой способ поначалу вам покажется более суровым. Начинающим программистам больше понравится второй способ — графический. Плагин ADT предлагает очень удобный редактор разметки интерфейса пользователя (рис. 5.1).

Какой способ будет рассмотрен в книге? Конечно же, первый — ручная разметка путем редактирования XML-файла. Ведь проблем с освоением редактора интерфейса, думаю, у вас не возникнет. На практике же вы можете использовать любой способ, а структуру файла разметки мы сейчас рассмотрим, чтобы правильно вас сориентировать.

Каждый элемент файла разметки является объектом класса `View` или `ViewGroup`. Если представить все элементы интерфейса пользователя в виде иерархии, то объекты класса `ViewGroup` будут ветвями дерева, а объекты класса `View` — листьями. Иерархия созданного интерфейса отображается на вкладке **Outline** редактора интерфейса. На рис. 5.1 видно, что добавлены разметка `RelativeLayout` (это и будет объект класса `ViewGroup`) и три кнопки (это объекты класса `View`, точнее их потомки).

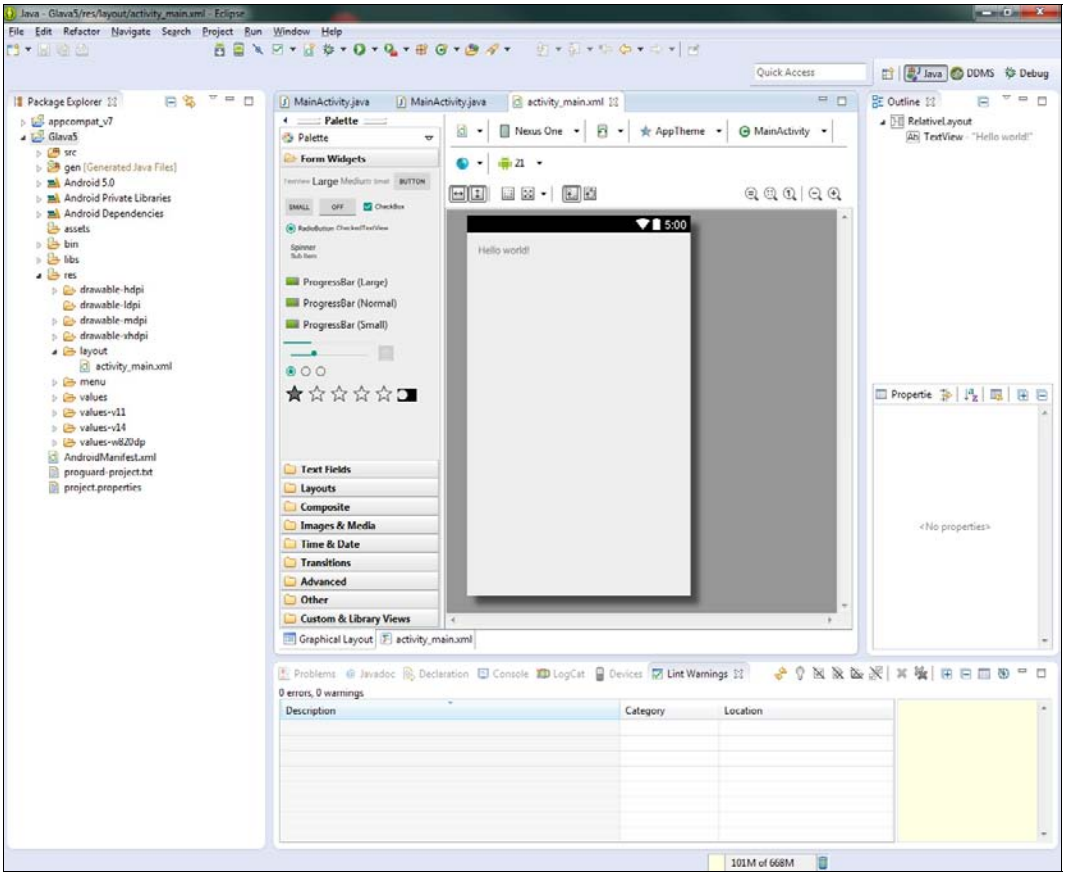


Рис. 5.1. Графическая разметка интерфейса пользователя

Рассмотрим файл разметки (res/layout/activity_main.xml) нашего первого проекта. С этим файлом мы уже знакомы, но, чтобы не листать лишний раз книгу, для наглядности еще раз приведу его в листинге 5.1.

Листинг 5.1. Файл разметки activity_ main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.sample.glava5.MainActivity" >

    <TextView
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

</RelativeLayout>
```

В каждом файле разметки должен быть только один корневой элемент. В нашем случае таким элементом является `RelativeLayout` (относительная разметка). После определения корневого элемента вы можете добавить в него дополнительные объекты разметки или виджеты в качестве дочерних элементов.

Рассмотрим некоторые атрибуты элементов `LinearLayout` и `TextView`:

- `xmlns:android` — объявление пространства имен Android. Стандартный атрибут и стандартное значение для Android-приложения;
- `android:layout_width` — ширина объекта `View` или `ViewGroup`. В нашем случае объект занимает весь экран, поэтому используется значение `match_parent` (это то же самое, что и значение `fill_parent`, которое применялось ранее. Вы можете использовать как `match_parent`, так и `fill_parent`). Можно задать и значение `wrap_content` — тогда ширина или высота определяются по содержимому элемента;
- `android:layout_height` — высота объекта. Для этого атрибута также можно использовать значения `match_parent` и `wrap_content`;
- `android:text` — текст, который должен отобразить объект `TextView`. В нашем примере вместо строковой константы используется значение из файла strings.xml — строка `hello_world`. Благодаря такому приему очень легко выполнить локализацию приложения (перевод на другой язык).

У каждого объекта `View` или `ViewGroup` — свой набор атрибутов. С ними мы познакомимся чуть позже — когда будем рассматривать базовые виджеты.

Установить атрибуты объектов можно как с помощью файла разметки, так и в Java-коде. В большинстве случаев имя атрибута XML соответствует названию метода в классе Java. Например, для установки атрибута `android:height` (задает высоту объекта) используется метод `setHeight(int)`, для установки атрибута `android:text` — метод `setText(int)` и т. д. Но бывают исключения. Метод `setHighlightColor(int)` служит для установки значения атрибута `android:textColorHighlight`. Как видите, название метода похоже на название атрибута, но не соответствует ему полностью. Все это говорит о том, что перед написанием Java-кода желательно обратиться к документации и не полагаться только на одну интуицию. Также при написании Java-кода можно воспользоваться помощью среды Eclipse. Если вы забыли (или вообще не знаете!), как называется то или иное свойство или метод объекта, введите его имя, точку и через мгновение раскроется выпадающий список со свойствами и методами этого объекта, а также с подсказкой по выделенному методу/свойству (рис. 5.2). Аналогичный функционал есть и в других IDE.

О Java-коде мы поговорим позже, пока же рассмотрим типы разметки.

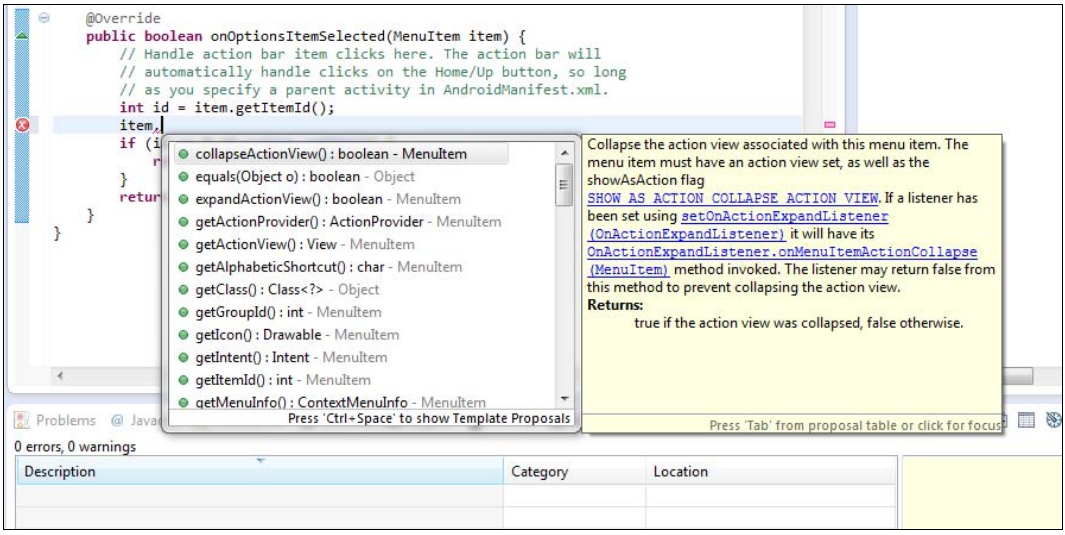


Рис. 5.2. Помощь в написании кода

5.1.2. Типы разметки

Вы можете использовать один из четырех типов разметки:

- `FrameLayout` — разметка фрейма;
- `LinearLayout` — линейная разметка;
- `TableLayout` — табличная разметка;
- `GridLayout` — разметка в виде сетки;
- `AbsoluteLayout` — абсолютная разметка, когда указывается явная позиция на экране в системе координат (x,y);
- `RelativeLayout` — относительная разметка.

FrameLayout

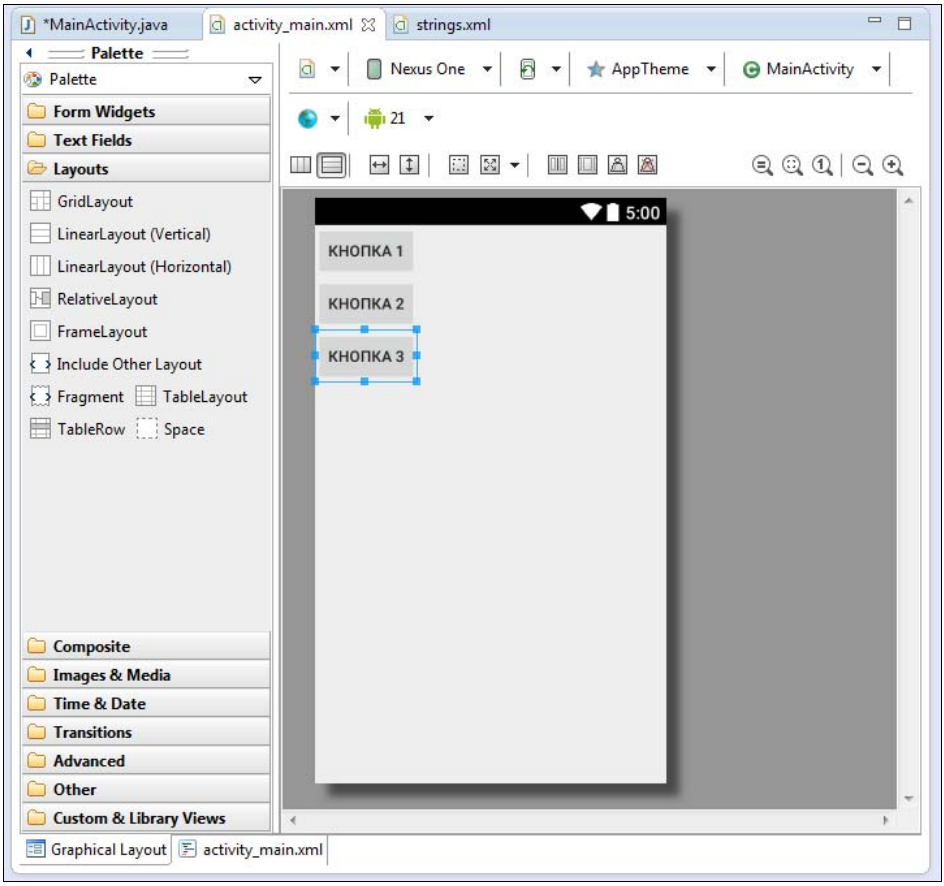
Начнем с первой разметки. `FrameLayout` — самый простой тип разметки. Все дочерние элементы `FrameLayout` будут прикреплены к верхнему левому углу экрана.

Этот тип разметки настолько примитивен, что даже наше простейшее приложение (которое мы создали в *главе 3*) использует тип разметки `LinearLayout`. Единственное применение для `FrameLayout` — это ее использование внутри ячейки таблицы (см. *далее*), а для создания полноценной разметки приложения этот вариант не годится.

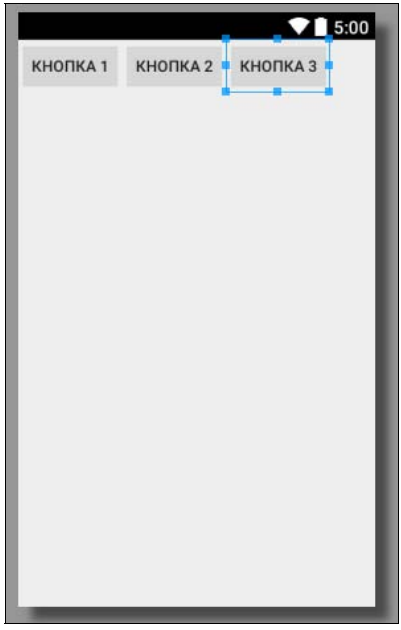
LinearLayout

Линейная разметка (`LinearLayout`) размещает виджеты горизонтально или вертикально — в зависимости от атрибута `android:orientation`:

```
android:orientation="vertical"
```



а



б

Рис. 5.3. *а* — кнопки выровнены вертикально;
б — кнопки выровнены горизонтально

Посмотрите на рис. 5.3, *а* — на нем задана линейная вертикальная разметка, поэтому кнопки размещаются друг под другом — вертикально. Чтобы задать горизонтальную разметку, нужно изменить атрибут так:

```
android:orientation="horizontal"
```

Измените атрибут `android:orientation`, затем перейдите на вкладку **Graphical Layout** и вы увидите, что наши кнопки выстроились горизонтально (рис. 5.3, *б*).

В листинге 5.2 приведен файл разметки для интерфейса с тремя кнопками, изображенного на рис. 5.3, *а*.

Листинг 5.2. Файл разметки main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <Button android:text="@string/button1" android:id="@+id/button1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"></Button>
    <Button android:text="@string/button2" android:id="@+id/button2"
android:layout_width="wrap_content"
android:layout_height="wrap_content"></Button>
    <Button android:text="@string/button3" android:id="@+id/button3"
android:layout_width="wrap_content"
android:layout_height="wrap_content"></Button>
</LinearLayout>
```

Рис. 5.4. У последней кнопки максимальный «вес»

У разметки LinearLayout есть атрибут android:layout_weight, задающий «вес» отдельного дочернего элемента. Чем выше «вес», тем важнее элемент. Элемент с максимальным «весом» займет всю оставшуюся часть родительского элемента. По умолчанию «вес» элемента равен 0. Установите для третьей кнопки «вес», отличный от 0:

```
<Button android:text="Button" android:id="@+id/button3" android:layout_weight="1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"></Button>
```

В результате последняя кнопка будет растянута на всю ширину экрана (рис. 5.4).

TableLayout

Разметка TableLayout размещает виджеты в строки и столбцы — в виде таблицы. Границы таблицы не отображаются, таблица служит не для представления данных, а сугубо для размещения виджетов.

Таблица может иметь строки с разным количеством ячеек. Для формирования строк таблицы разметки используются объекты TableRow. Каждый такой объект — это одна строка таблицы. В строке может вообще не быть ячеек, может быть одна или несколько ячеек. В свою очередь ячейка может быть объектом класса ViewGroup — другими словами, ячейка допускает разметку. То есть, в ячейку таблицы вы можете поместить объект LinearLayout и с его помощью разместить элементы внутри ячейки. Как элемент ячейки можно использовать и другой элемент TableLayout — получится вложенная таблица.

Сейчас мы продемонстрируем использование табличной разметки на примере создания клавиатуры для простейшего калькулятора. Создайте новый проект или откройте наш простой проект, разработанный в *главе 3*. Перейдите к редактору разметки (для этого откройте в структуре проекта файл /res/layout/main.xml и щелкните на нем двойным щелчком). Затем перейдите на вкладку **activity_main.xml** и удалите из XML-файла все, кроме элемента <?xml>.

Затем создайте разметку — для этого в группе **Layouts** выберите **TableLayout**. В эту разметку нужно добавить четыре элемента TableRow. Для каждого элемента TableRow нужно установить следующие атрибуты:

- Gravity = center;
- Layout width = fill_parent;
- Layout height = wrap_content.

Для этого щелкните на элементе правой кнопкой, выберите необходимый атрибут и установите нужное значение (рис. 5.5). Свойство (атрибут) **Gravity**, установка которого показана на рис. 5.5, выравнивает элемент в контейнере — мы выбираем выравнивание по центру. В контекстном меню приведены не все свойства. Ранее (в старой версии IDE) все свойства были собраны в подменю **Properties**. Сейчас часто используемые свойства выделены в контекстное меню, а некоторые другие свойства (которые используются реже) собраны в подменю **Other Properties**.

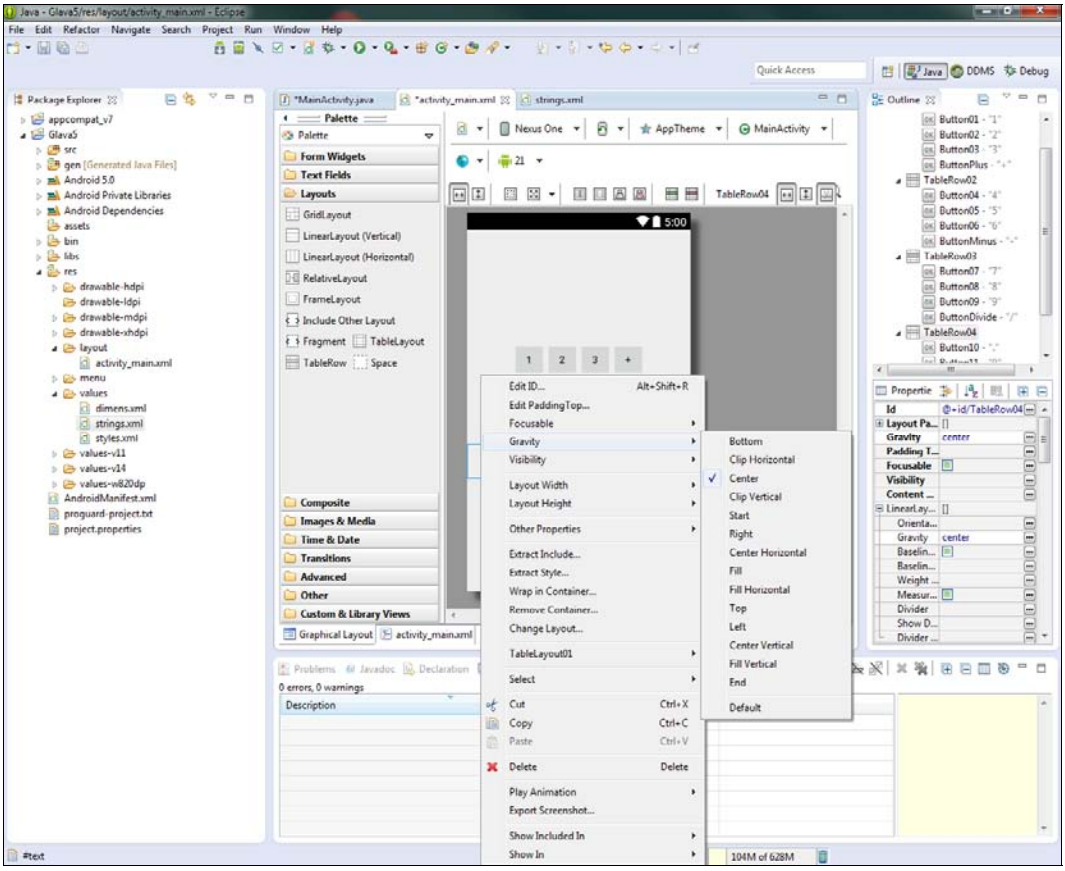


Рис. 5.5. Установка значения атрибута

Далее в каждую строку (в каждый контейнер `TableRow`) нужно добавить по четыре кнопки (кнопки находятся в группе **Form Widgets**). Ширину каждой кнопки нужно установить в `20pt`, но это значение зависит от типа платформы Android и от размера экрана. На рис. 5.6 представлена наша клавиатура, версия платформы — 5.0. Обратите внимание на иерархию графического интерфейса (вкладка **Outline**).

В XML-файле разметка `TableLayout` будет объявлена так:

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/TableLayout01"
android:layout_width="fill_parent"
android:layout_height="fill_parent">
```

Наша клавиатура смотрится некрасиво только потому, что она расположена не по центру экрана. Чтобы она была размещена по центру (по горизонтали и по вертикали), установите атрибут `Gravity` для `TableLayout`. В итоге в файле разметки `TableLayout` будет объявлена так:

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/TableLayout01"
```

```
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:gravity="center">
```

На экране это будет выглядеть, как показано на рис. 5.7.

Полный код XML-файла разметки клавиатуры калькулятора представлен в листинге 5.3.

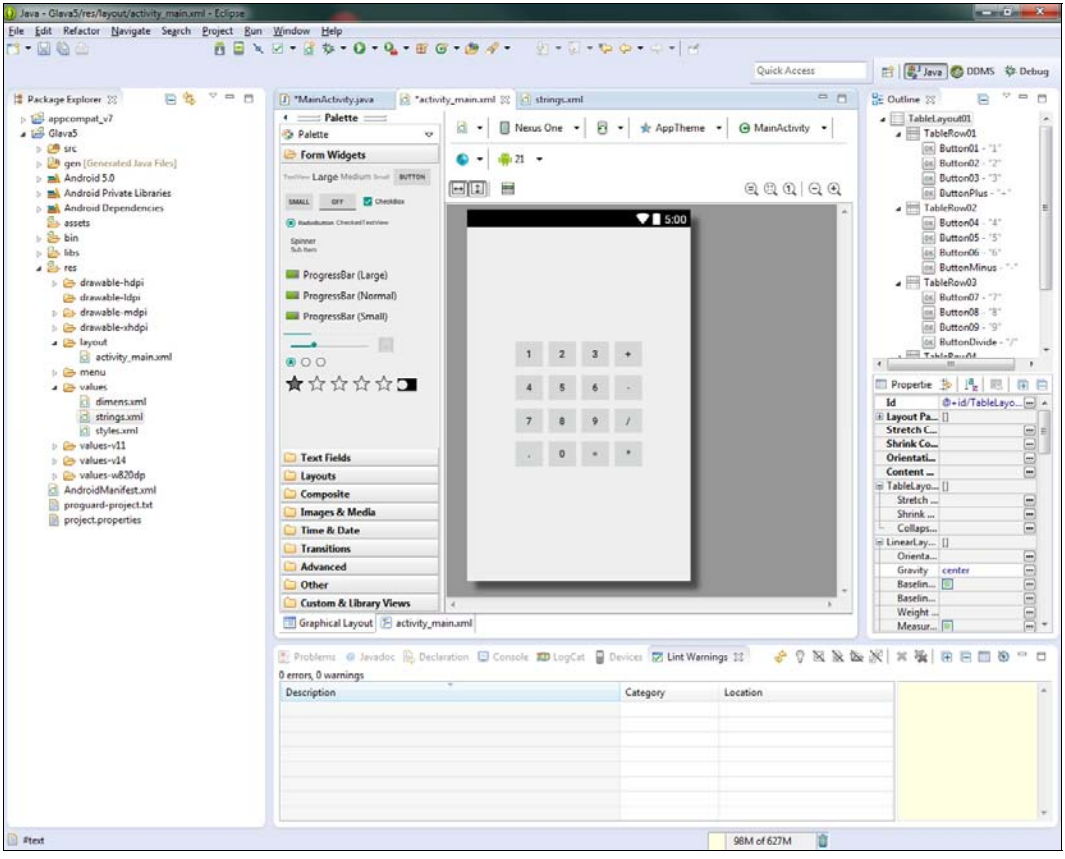


Рис. 5.6. Созданная клавиатура калькулятора (Android 5.0)

Листинг 5.3. Файл activity_main.xml для клавиатуры калькулятора

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/TableLayout01"
        android:layout_width="fill_parent"
        android:gravity="center"
        android:layout_height="fill_parent">
<TableRow
        android:id="@+id/TableRow01"
        android:layout_height="wrap_content"
```

```
        android:layout_width="fill_parent"
        android:gravity="center">
<Button
        android:id="@+id/Button01"
        android:layout_height="wrap_content"
        android:text="1"
        android:layout_width="20pt"/>
<Button
        android:id="@+id/Button02"
        android:layout_height="wrap_content"
        android:text="2"
        android:layout_width="20pt"/>
<Button
        android:id="@+id/Button03"
        android:layout_height="wrap_content"
        android:text="3"
        android:layout_width="20pt"/>
<Button
        android:id="@+id/ButtonPlus"
        android:layout_height="wrap_content"
        android:text="+"
        android:layout_width="20pt"/>
</TableRow>
<TableRow
        android:id="@+id/TableRow02"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:gravity="center">
<Button
        android:id="@+id/Button04"
        android:layout_height="wrap_content"
        android:layout_width="20pt"
        android:text="4"/>
<Button
        android:id="@+id/Button05"
        android:layout_height="wrap_content"
        android:layout_width="20pt"
        android:text="5"/>
<Button
        android:id="@+id/Button06"
        android:layout_height="wrap_content"
        android:layout_width="20pt"
        android:text="6"/>
<Button
        android:id="@+id/ButtonMinus"
        android:layout_height="wrap_content"
```

Рис. 5.7. Клавиатура размещена по центру экрана мобильного устройства

```
        android:text="-"
        android:layout_width="20pt"/>
</TableRow>
<TableRow
        android:id="@+id/TableRow03"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:gravity="center">
<Button
        android:id="@+id/Button07"
        android:layout_height="wrap_content"
        android:layout_width="20pt"
        android:text="7"/>
<Button
        android:id="@+id/Button08"
        android:layout_height="wrap_content"
        android:layout_width="20pt"
        android:text="8"/>
<Button
        android:id="@+id/Button09"
        android:layout_height="wrap_content"
        android:layout_width="20pt"
        android:text="9"/>
<Button
        android:id="@+id/ButtonDivide"
        android:layout_height="wrap_content"
        android:text="/"
        android:layout_width="20pt"/>
</TableRow>
<TableRow
        android:id="@+id/TableRow04"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:gravity="center">
<Button
        android:id="@+id/Button10"
        android:layout_height="wrap_content"
        android:layout_width="20pt"
        android:text="."/>
<Button
        android:id="@+id/Button11"
        android:layout_height="wrap_content"
        android:layout_width="20pt"
        android:text="0"/>
<Button
        android:id="@+id/Button12"
        android:layout_height="wrap_content"
```

```
        android:layout_width="20pt"
        android:text="="/>
<Button
        android:id="@+id/ButtonMul"
        android:layout_height="wrap_content"
        android:text="*"
        android:layout_width="20pt"/>
</TableRow>
</TableLayout>
```

Когда вы вникнете в структуру XML-файла, то обнаружите, что редактировать разметку интерфейса вручную даже проще, чем использовать интерфейс Eclipse. Во всяком случае, создать разметку для Android-приложения вручную не сложнее, чем создать HTML-страницу. Я предпочитаю создавать разметку вручную, а устанавливать некоторые свойства — с помощью редактора разметки — не всегда удается запомнить все необходимые свойства и их значения.

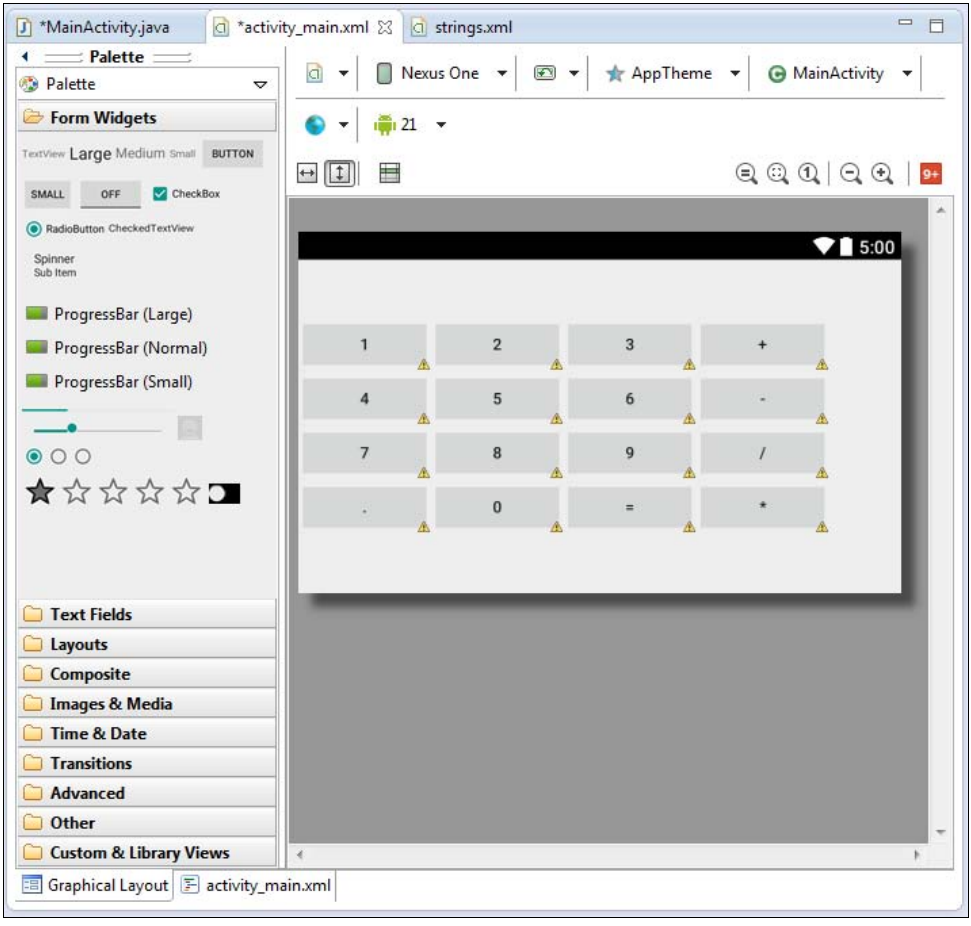


Рис. 5.8. Клавиатура калькулятора для Android 5.0

Вернемся к ширине кнопок. На рис. 5.7 наша клавиатура выглядит практически идеально. Увеличивать размер кнопок не нужно — ведь если вы будете создавать полноценный калькулятор, вам понадобится место для дополнительных кнопок: MR, MC, M+, MS, CE, CE и т. д. На рис. 5.8 показано, что будет, если увеличить размер кнопок до 50 pt. Они даже не поместятся на экране при книжной ориентации, поэтому на рис. 5.8 ориентация экрана — альбомная.

RelativeLayout

При относительной разметке (`RelativeLayout`) позиция дочерних виджетов определяется относительно родительского элемента. При использовании этого типа разметки элементы графического интерфейса размещены так, что если первый элемент расположен по центру, другие элементы, выравнивание которых задается относительно первого элемента, будут также выровнены по центру экрана.

Лучше всего продемонстрировать относительную разметку на примере. На рис. 5.9 изображен интерфейс пользователя: надпись, поле ввода и две кнопки. На рис. 5.10 — иерархия интерфейса. Файл разметки интерфейса представлен в листинге 5.4.

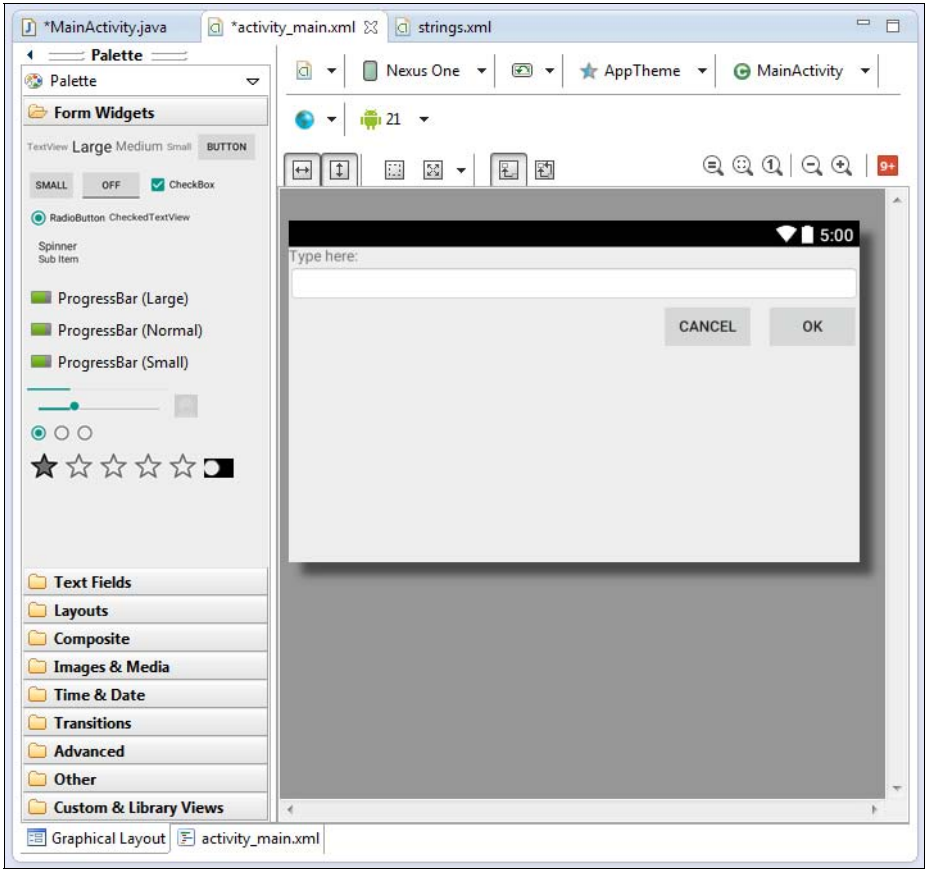


Рис. 5.9. Относительная разметка

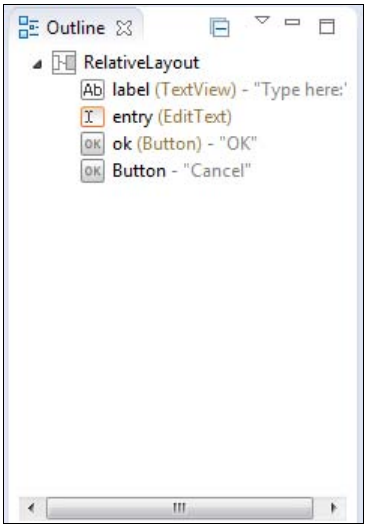


Рис. 5.10. Иерархия интерфейса

Листинг 5.4. Файл разметки интерфейса, изображенного на рис. 5.9

```
<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="Type here:"/>
    <EditText
        android:id="@+id/entry"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="@android:drawable/editbox_background"
        android:layout_below="@id/label"/>
    <Button
        android:id="@+id/ok"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/entry"
        android:layout_alignParentRight="true"
        android:layout_marginLeft="10dip"
        android:text="OK" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toLeftOf="@id/ok"
        android:layout_alignTop="@id/ok"
        android:text="Cancel" />
</RelativeLayout>
```

Более сложный пример использования относительной разметки можно найти на моем форуме: **http://www.dkws.org.ua/phpbb2/viewtopic.php?p=34603#34603**.

GridLayout

Этот тип разметки впервые появился в Android 4 и очень похож на `TableLayout`. В проектах, адаптируемых под новые платформы (4.*x* и 5.*x*), рекомендуется использовать именно этот тип разметки. Разметка `GridLayout` относится к классу `android.widget.GridLayout` и позволяет создавать колонки, ряды и ячейки как в `TableLayout`, но при этом элементы можно гибко настраивать. Получается, что новый тип разметки гораздо удобнее `TableLayout`.

Чтобы попробовать `GridLayout` на практике, давайте воссоздадим с его помощью клавиатуру нашего калькулятора (рис. 5.11). Посмотрите, насколько элегантнее стал код (листинг 5.5). Сравните его с кодом из листинга 5.3.

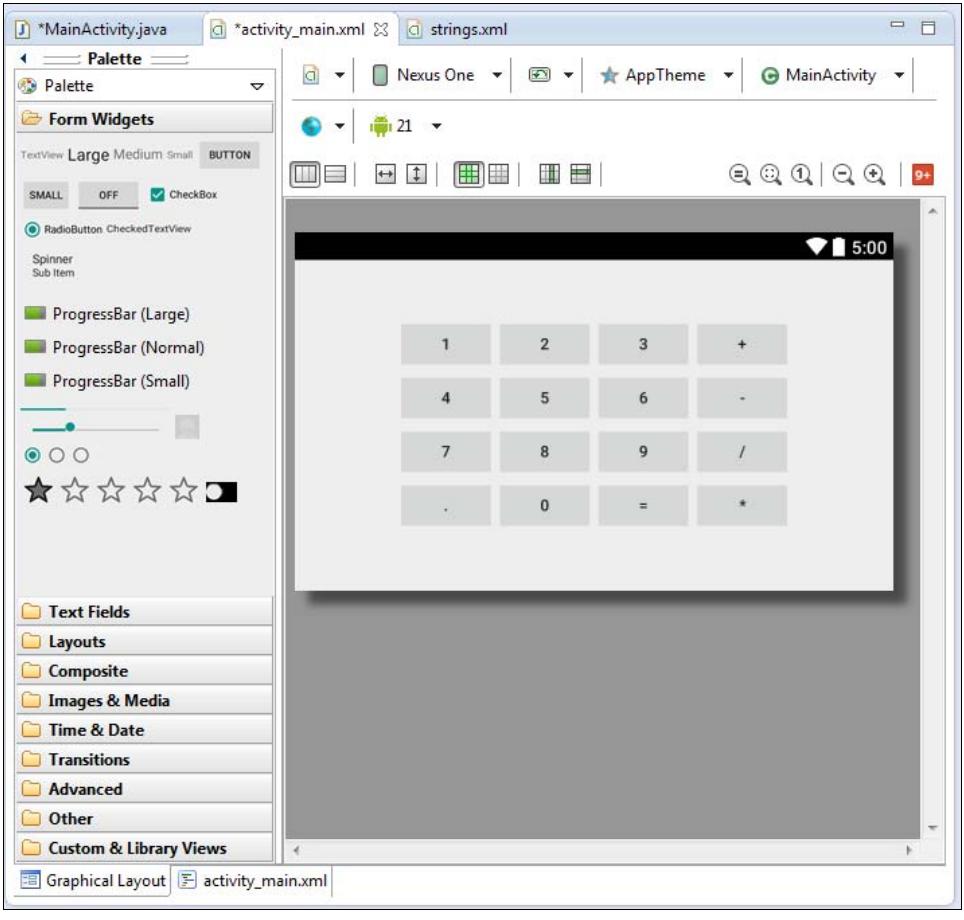


Рис. 5.11. Клавиатура калькулятора с использованием `GridLayout`

Листинг 5.5. Клавиатура калькулятора с использованием `GridLayout`

```
<?xml version="1.0" encoding="utf-8"?>
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:columnCount="4"
    android:orientation="horizontal" >
    <Button android:text="1" />
    <Button android:text="2" />
    <Button android:text="3" />
    <Button android:text="+" />
    <Button android:text="4" />
    <Button android:text="5" />
    <Button android:text="6" />
    <Button android:text="-" />
    <Button android:text="7" />
    <Button android:text="8" />
    <Button android:text="9" />
    <Button android:text="/" />
    <Button android:text="." />
    <Button android:text="0" />
    <Button android:text="=" />
    <Button android:text="*" />
```

Обратите внимание: мы сначала указали, сколько у нас будет столбцов (`android:columnCount="4"`), а потом просто перечислили элементы, которые нужно поместить в столбцы.

Но это слишком тривиальный пример. С помощью `GridLayout` можно создавать и более сложные варианты разметки (листинг 5.6, рис. 5.12).

Листинг 5.6. Более сложная клавиатура калькулятора

```
<?xml version="1.0" encoding="utf-8"?>
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:columnCount="4"
    android:orientation="horizontal">

    <Button
        android:layout_column="3"
        android:text="*" />
    <Button android:text="1" />
    <Button android:text="2" />
    <Button android:text="3" />
```

```
    <Button android:text="/" />
    <Button android:text="4" />
    <Button android:text="5" />
    <Button android:text="6" />
    <Button android:text="-" />
    <Button android:text="7" />
    <Button android:text="8" />
    <Button android:text="9" />
    <Button
        android:layout_rowSpan="3"
        android:text="+" />
    <Button
        android:layout_columnSpan="2"
        android:text="0" />
    <Button android:text="^2" />
    <Button
        android:layout_columnSpan="3"
        android:text="=" />
</GridLayout>
```

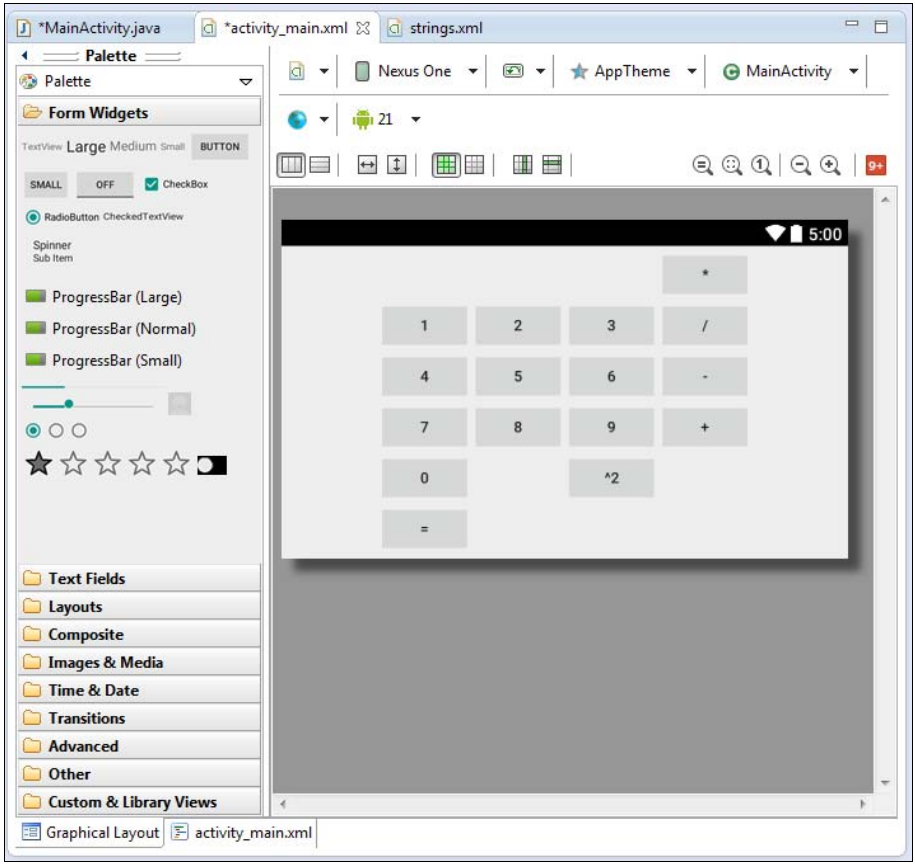


Рис. 5.12. Сложная клавиатура калькулятора

С помощью TableLayout организовать что-либо подобное тоже можно, но код будет более громоздким.

Absolute Layout

Абсолютная разметка не очень удобна, поскольку нужно указывать координаты элементов явно. Может, когда-то она вам пригодится, а сейчас лишь приведу пример с двумя кнопками: **OK** и **Отмена** (листинг 5.7), расположение которых задается явно.

Листинг 5.7. Пример абсолютной разметки

```
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent">

   <Button
      android:layout_width="100dp"
      android:layout_height="wrap_content"
      android:text="OK"
      android:layout_x="50px"
      android:layout_y="361px" />
   <Button
      android:layout_width="100dp"
      android:layout_height="wrap_content"
      android:text="Отмена"
      android:layout_x="225px"
      android:layout_y="361px" />

</AbsoluteLayout>
```

5.1.3. Исследование разметки с помощью Hierarchy Viewer

Иногда элементы интерфейса почему-то выстраиваются не так, как нам того хочется. Помочь разобраться в этом позволяет утилита Hierarchy Viewer.

Для запуска Hierarchy Viewer нужно выполнить следующие действия:

1. Запустить эмулятор мобильного устройства. Лучше всего это сделать с помощью команды **Run** текущего проекта — чтобы было загружено интересующее вас приложение. При этом не спешите и дождитесь полной загрузки эмулятора и приложения. На рис. 5.13 показан запущенный эмулятор, в который загружено приложение, отображающее клавиатуру эмулятора.
2. Запустить файл hierarchyviewer.bat из каталога tools (находится в каталоге, в который вы установили Android SDK). В открывшемся окне Hierarchy Viewer

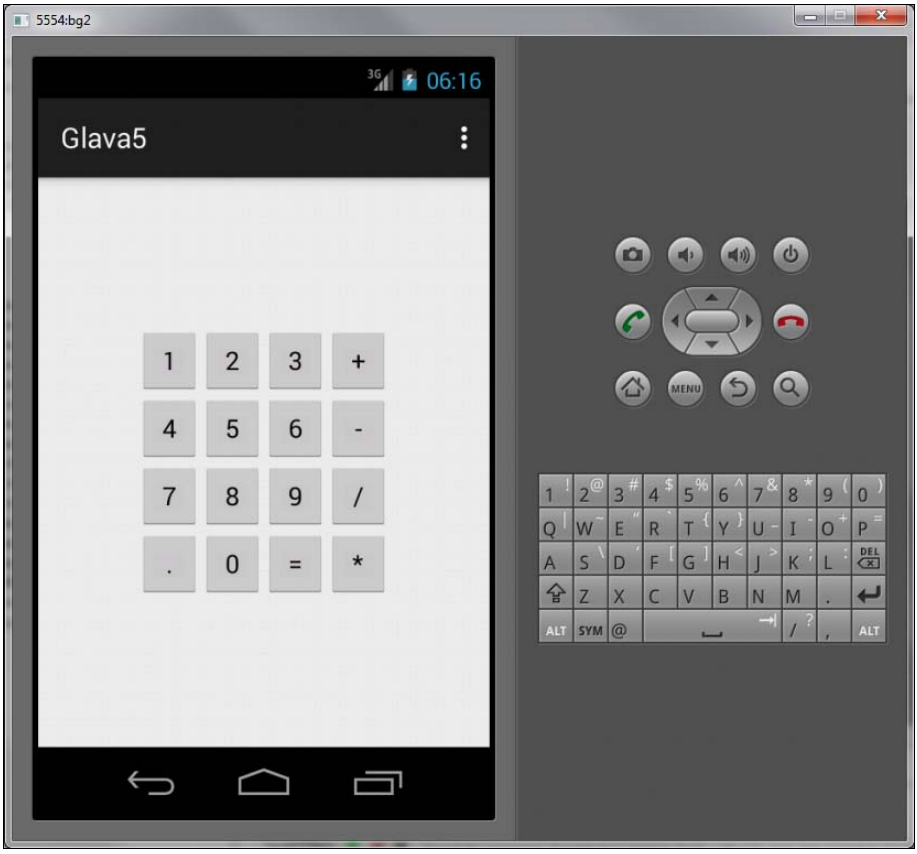


Рис. 5.13. Эмулятор с загруженным приложением

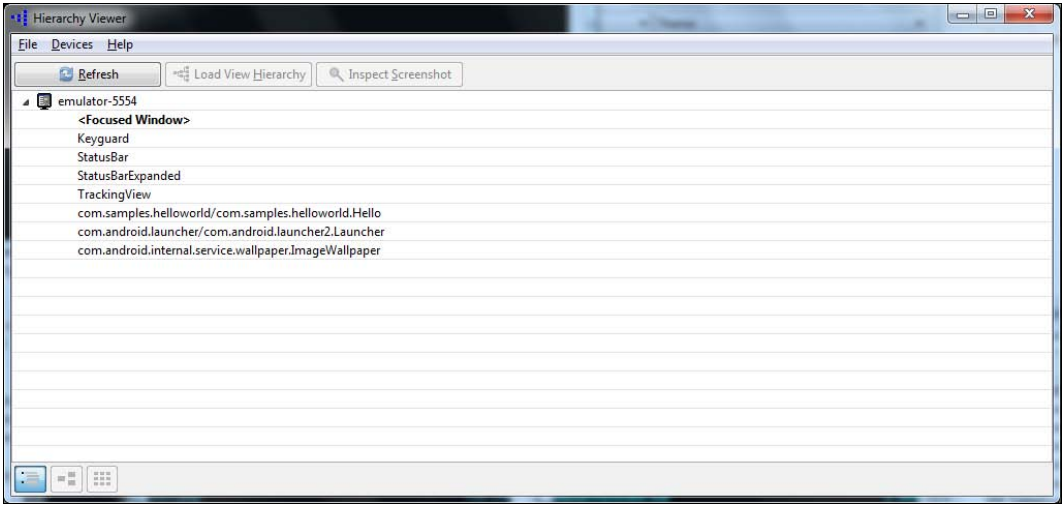


Рис. 5.14. Утилита Hierarchy Viewer

(рис. 5.14) щелкните по ссылке **<Focused Window>**, а затем нажмите кнопку **Load View Hierarchy**.

3. Немного подождите и вы увидите иерархию интерфейса пользователя запущенного приложения (рис. 5.15). Изменить масштаб иерархии можно с помощью колесика мыши.

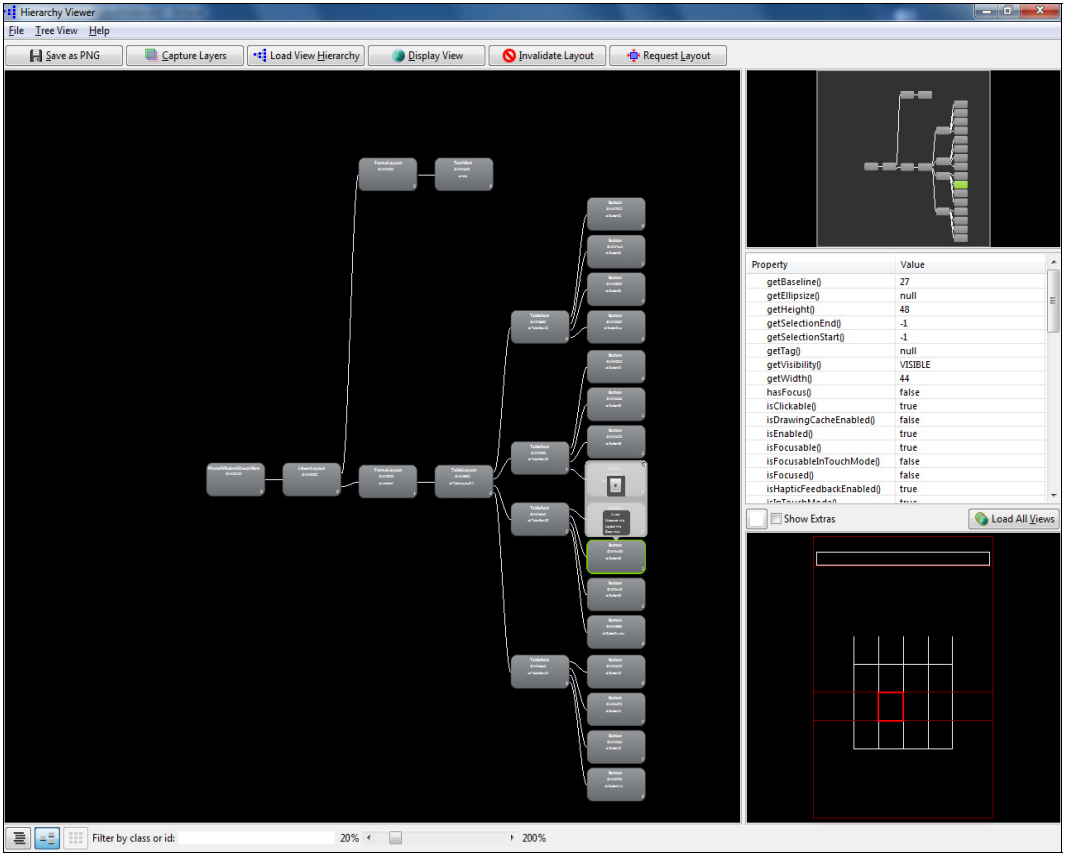


Рис. 5.15. Иерархия интерфейса пользователя

5.2. Основные виджеты графического интерфейса

Виджеты — это элементы графического интерфейса. Ранее мы уже познакомились с двумя виджетами: текстовым полем и кнопкой. Сейчас мы наше знакомство с виджетами продолжим.

Прежде чем приступить, создайте новый проект с именем `Test5`, деятельность этого проекта назовем `Test5Activity`, а пакет: `com.samples.test5` — так вам будет понятнее, откуда взялись некоторые идентификаторы.

5.2.1. Текстовые поля

В Android вы можете использовать два текстовых поля: `TextView` и `EditText`. Первое служит для отображения текста без возможности его редактирования, а второе — это классическое текстовое поле с возможностью ввода и редактирования текста.

Несмотря на свою простоту, виджет `TextView` довольно часто применяется в Android-программах для вывода инструкций по работе с программой и другого текста.

В файле разметки значение `TextView` можно установить так:

```
android:text="Text";
```

В Java-коде значение виджета устанавливается так:

```
TextView text = (TextView)findViewById(R.id.textView1);
text.setText("Sample text");
```

Совсем другое дело, если вы планируете создать приложения с многоязыковой поддержкой пользовательского интерфейса. Тогда непосредственно в файле разметки значение (текстовую строку) указывать не нужно. Вместо этого создается ссылка на текстовый XML-ресурс:

```
android:text="@string/str_value"
```

Здесь `str_value` — это имя строкового ресурса, описанного в файле strings.xml. В Java-коде установить имя ресурса можно тем же методом `setText()`:

```
TextView text = (TextView)findViewById(R.id.textView1);
text.setText(R.string.str_value);
```

У элемента `TextView` есть много методов и свойств. Мы рассмотрим только основные свойства, относящиеся к отображению текста.

Размер шрифта задается свойством `android:textSize` в пикселах (`px`), независимых от плотности пикселах (`dp`), независимых от масштабирования пикселах (`sp`), пунктах (`pt`), дюймах (`in`) и миллиметрах (`mm`):

```
android:textSize="14pt";
```

Стиль текста задается свойством `android:textStyle`:

- `normal` — обычное начертание символов;
- `bold` — полужирное начертание символов;
- `italic` — курсив.

Например:

```
android:textStyle="bold"
```

Цвет шрифта задается в свойстве `android:textColor` указанием его значений в шестнадцатеричной кодировке в формате #RGB или #ARGB. Во втором случае значение A — это прозрачность. Если A = 0, то прозрачность 100 % (элемент практически не будет виден).

Теперь немного практики. Создайте новый проект. По умолчанию будет создан проект, выводящий строку «Hello World!» с помощью TextView — как раз то, что нам и нужно.

Полосы прокрутки

По умолчанию, если текст не помещается в виджете TextView, появляются полосы прокрутки. Другими словами, вам ничего не нужно делать для того, чтобы они появились. Но если есть желание отображать полосы прокрутки, даже когда они не нужны, вы можете использовать виджеты ScrollView (вертикальная полоса прокрутки) и HorizontalScrollView (горизонтальная полоса прокрутки). Прочитать об этих виджетах можно в руководстве по Android:

- **http://developer.android.com/reference/android/widget/ScrollView.html**
- **http://developer.android.com/reference/android/widget/HorizontalScrollView.html**

Перейдите к файлу разметки проекта — res/layout/activity_main.xml. Попробуем изменить свойства TextView. Если вы забыли, как называется то или иное свойство, — Eclipse его вам подскажет. Наберите: android: и немного подождите — откроется выпадающий список, из которого нужно будет выбрать искомое свойство (рис. 5.16). Как видите, Eclipse помогает писать не только Java-код, но и файлы разметки.

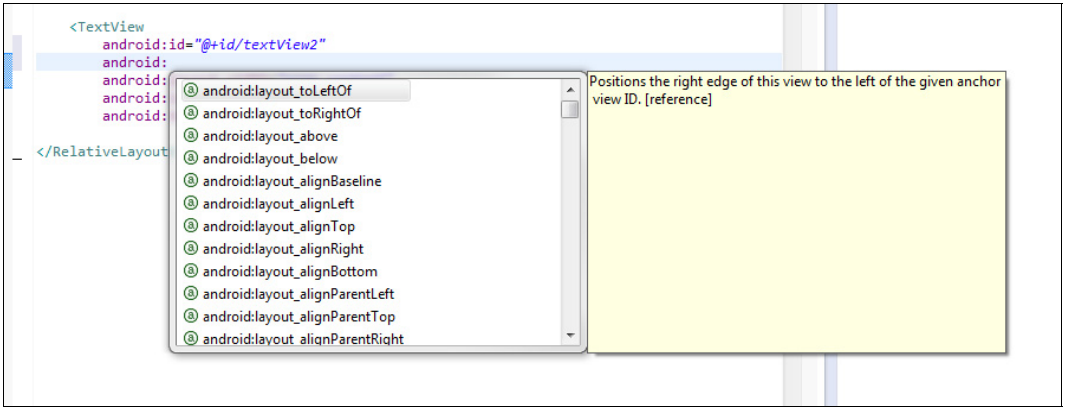


Рис. 5.16. Подсказка Eclipse при создании файла разметки

Установим свойства textSize и textStyle, а также присвоим имя нашему текстовому полю (txt1), чтобы к нему можно было обратиться из Java-кода (листинг 5.8 и рис. 5.17).

Листинг 5.8. Файл activity_main.xml, установка атрибутов TextView

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    >
```

```
<TextView
    android:id="@+id/txt1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="20pt"
    android:textStyle="bold"
    android:text="@string/hello_world"
    />
</LinearLayout>
```

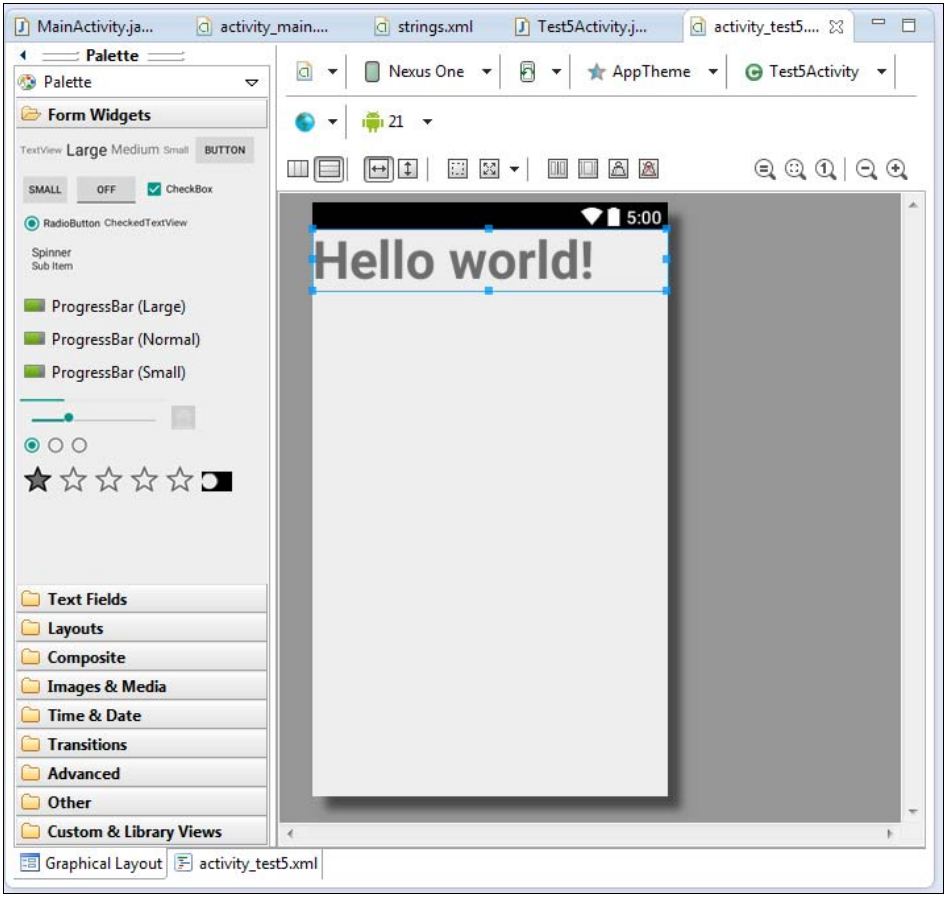


Рис. 5.17. Изменение размера и стиля шрифта

Теперь попробуем усложнить задачу и изменить текст нашего виджета из Java-кода. Для этого откройте файл, содержащий Java-код нашего приложения: src/<название пакета>/<название_проекта>Activity.java. В нашем случае этот файл называется src/com.samples.test5/Test5Activity.java. На рис. 5.18 показано содержимое этого файла по умолчанию, а также область **Package Explorer**, объясняющая, как «добраться» до этого файла.

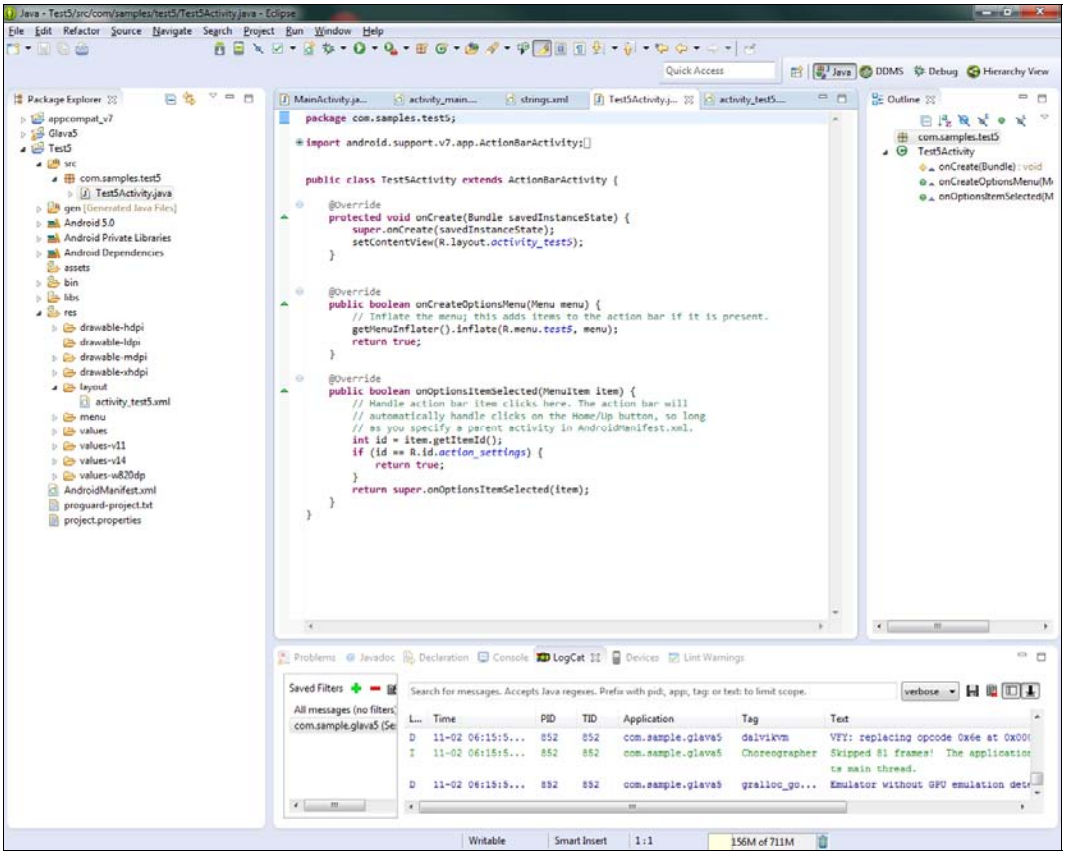


Рис. 5.18. Файл Test5Activity.java

Измените Java-файл так, чтобы он выглядел, как показано в листинге 5.9. Чтобы работать с виджетом `TextView`, нужно импортировать пакет `android.widget.TextView`. Самые важные строки выделены полужирным шрифтом, а лишние — удалены для компактности кода.

Листинг 5.9. Файл Test5Activity.java

```
/* первая строка зависит от названия вашего пакета, у вас будет другой */
package com.samples.test5;

import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.widget.TextView;

/* название класса (Test5Activity) зависит от параметров, указанных
при создании проекта */

public class Test5Activity extends ActionBarActivity {
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_test5);

        TextView txt1 = (TextView)findViewById(R.id.txt1);
        txt1.setText("Hi!");

    }

}
```

Теперь запустите приложение. В качестве значения текстового поля установлена строка "Hi!" (рис. 5.19).

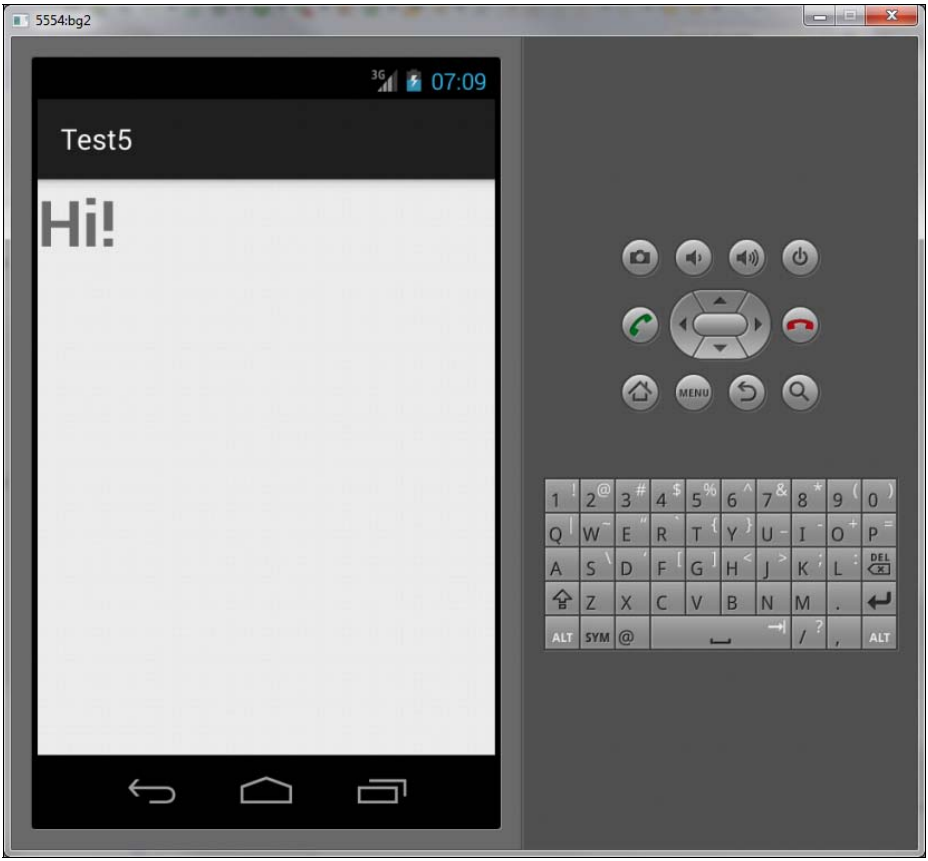


Рис. 5.19. Результат работы нашей программы

Текстовое поле EditText, как уже было отмечено, позволяет вводить и редактировать текст. Основной метод этого поля — getText(), позволяющий получить введенный пользователем текст. Значение, возвращаемое методом getText(), имеет

тип Editable. По сути, Editable — это надстройка над String, но, в отличие от него, значение типа Editable может быть изменено в процессе выполнения программы (String является неизменяемым типом, и при изменении типа String попросту создается другой экземпляр String, содержащий новое значение).

Кроме метода getText() вам может понадобиться метод selectAll(), выделяющий весь текст в окне EditText. Если весь текст выделять не нужно, можно использовать метод setSelection():

```
setSelection(int start, int stop)
```

Этот метод выделяет участок текста, начиная с позиции start до позиции stop.

Установить тип начертания шрифта можно с помощью метода setTypeface — например:

```
txt1.setTypeface(null, Typeface.NORMAL);
```

Вместо NORMAL можно указать BOLD и/или ITALIC.

Для добавления поля EditText в разметку окна нужно поместить в файл разметки следующий код:

```
<EditText
     android:id="@+id/entry1"
     android:layout_width="match_parent"
     android:layout_height="wrap_content"
     android:text="Simple text"/>
```

Далее мы научимся работать с этим полем — займемся получением и установкой текста поля. А пока ограничимся только тем, что добавим поле в файл разметки. Как выглядит это поле, вы уже видели на рис. 5.19.

5.2.2. Кнопки

Кнопки — очень важный элемент пользовательского интерфейса. К кнопкам относятся не только непосредственно сами кнопки, но и переключатели (независимые и зависимые). Для определения кнопок имеется пять классов:

- Button;
- CheckButton;
- ToggleButton;
- RadioButton;
- ImageButton.

Классы CheckButton, ToggleButton и RadioButton являются потомками класса CompoundButton, который, в свою очередь, является потомком класса Button. А вот класс ImageButton является потомком класса ImageView, поэтому ImageButton представляет собой более изображение, нежели кнопку — в прямом понимании этого слова. Имеется также кнопка SmallButton, которая выглядит практически так же, как и Button, только ее размеры более компактные. Кнопки Button и SmallButton

отличаются только атрибутом `style`. Для обычной кнопки он не указывается, а для `SmallButton` выглядит так:

```
style="?android:attr/buttonStyleSmall"
```

Button — обычная кнопка

Начнем с обычной кнопки и продемонстрируем создание относительно простого приложения. Java-кода в этом приложении будет чуть больше, чем обычно, поэтому читателям, не знакомым с Java, нужно быть внимательными, чтобы не допустить ошибку.

Добавьте в наш проект, демонстрирующий изменение значения текстового поля с помощью Java-кода, одну кнопку. Теперь изменение значения текстового поля будет происходить не при запуске приложения, а при нажатии кнопки (рис. 5.20).

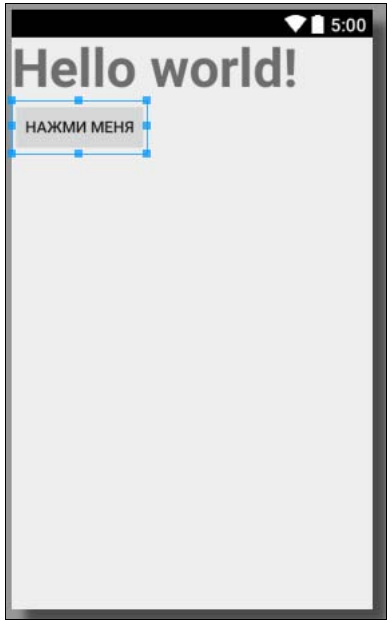


Рис. 5.20. Разметка проекта с текстовым полем и кнопкой

XML-разметка этого проекта представлена в листинге 5.10.

Листинг 5.10. Файл разметки проекта с текстовым полем и кнопкой

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    >
<TextView
    android:id="@+id/txt1"
```

```
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textSize="20pt"
    android:textStyle="bold"
    android:text="@string/hello_world"
    />

<Button
    android:text="@string/press_me"
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
</Button>
</LinearLayout>
```

Обратите внимание на идентификаторы текстового поля и кнопки. Текстовое поле называется `txt1`, а кнопка — `button1`. Этими идентификаторами мы воспользуемся в Java-коде. Также не забудьте указать в файле strings.xml константу `press_me` со значением `"Нажми меня"`.

Кстати, приятно, когда Eclipse пишет код за вас — автодополнение кода очень удачно реализовано в этой IDE. Java-код нашего приложения приведен в листинге 5.11.

Листинг 5.11. Java-код второго проекта

```
/* первая строка зависит от названия вашего пакета, у вас будет другой */
package com.samples.test5;

import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.Button;
import android.view.View;


/* название класса (Test5Activity) зависит от параметров, указанных при
создании проекта */

public class Test5Activity extends ActionBarActivity {

       private TextView txt1;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_test5);
```

```
        /* находим текстовое поле */
        txt1 = (TextView)findViewById(R.id.txt1);
        /* находим кнопку */
        final Button button1 = (Button)findViewById(R.id.button1);
        button1.setOnClickListener(new View.OnClickListener() {

                    public void onClick(View v) {
                        // TODO Auto-generated method stub
                        txt1.setText("Hi!");
                    }
                });

    }

}
```

Как обычно, самые важные моменты, на которые нужно обратить внимание, выделены полужирным шрифтом. Сначала мы объявляем переменную `txt1` — для текстового поля. Затем находим текстовое поле и кнопку. После этого с помощью метода `setOnClickListener()` устанавливаем обработчик события — в нашем случае нажатия кнопки. Обработчик события очень прост: мы устанавливаем новое значение текстового поля `txt1`.

Теперь запустите приложение. Сначала будет выведена строка **Hello World!** (рис. 5.21, *а*), а после нажатия кнопки строка будет изменена на **Hi!** (рис. 5.21, *б*).

Думаю, как установить обработчик нажатия кнопки, понятно. Однако код получился довольно громоздкий. Скажем, когда у вас одна или две кнопки, то такой код — это не проблема. А вот когда у вас 5 кнопок (или больше), то очень легко запутаться — уж больно много скобок. Хотя Eclipse делает все возможное, чтобы вам помочь, есть один способ уменьшить код и существенно упростить его для восприятия.

Представим, что вы в файле разметки объявили пять кнопок с именами от `button1` до `button5`. Сначала находим кнопки:

```
final Button button1 = (Button)findViewById(R.id.button1);
final Button button2 = (Button)findViewById(R.id.button2);
...
final Button button5 = (Button)findViewById(R.id.button5);
```

Потом устанавливаем один обработчик для всех кнопок:

```
button1.setOnClickListener(this);
button2.setOnClickListener(this);
...
button5.setOnClickListener(this);
```

Затем анализируем, какая кнопка была нажата, и выполняем соответствующее действие:

```
@Override
public void onClick(View v) {
switch (v.getId()) {
        case R.id.button1: txt1.setText("Button 1"); break;
        case R.id.button2: txt1.setText("Button 2"); break;
        case R.id.button3: txt1.setText("Button 3"); break;
        case R.id.button4: txt1.setText("Button 4"); break;
        case R.id.button5: txt1.setText("Button 5"); break;
        }
}
```

Есть еще один способ описания обработчика кнопки. В файле разметки нужно установить свойство `android:onClick`, содержащее имя обработчика, который будет вызван. Например:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/pick_date"
    android:onClick="Button1Click" />
```

А в код класса главной деятельности добавить описание этого обработчика:

```
public void Button1Click(View v) {
    // сделать что-то
}
```

а

б

Рис. 5.21. Второй проект в работе: *а* — до нажатия кнопки; *б* — после нажатия кнопки

RadioButton — зависимые переключатели

Зависимые переключатели (виджеты класса `RadioButton`) используются внутри контейнера `RadioGroup` — группы зависимых переключателей. Зависимый переключатель позволяет выбрать только одну из опций в одной из групп переключателей. В каждой из групп может быть активным только один переключатель.

Вы в основном будете пользоваться тремя методами:

- `toggle()` — инвертирует состояние зависимого переключателя;
- `isChecked()` — возвращает значение зависимого переключателя (`true` — активен);
- `setChecked()` — изменяет значение переключателя в зависимости от переданного параметра.

Для демонстрации работы с зависимыми переключателями добавьте в наш проект с кнопкой и текстовым полем два зависимых переключателя с именами `r1` и `r2`. При нажатии кнопки значение текстового поля будет изменено в зависимости от выбранного переключателя. Если ни один из переключателей не выбран, значение `TextView` изменено не будет. Разметка проекта приведена в листинге 5.12.

Листинг 5.12. Файл разметки проекта (зависимые переключатели)

```
<?xml version="1.0" encoding="utf-8"?>
<RadioGroup xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
<RadioButton
       android:text="Value 1"
       android:id="@+id/r1"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content">
</RadioButton>
<RadioButton android:text="Value 2"
       android:id="@+id/r2"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content">
</RadioButton>
<TextView
    android:id="@+id/txt1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textSize="20pt"
    android:textStyle="bold"
    android:text="@string/hello_world"
    />
```

```
<Button
        android:text="OK"
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
</Button>
</RadioGroup>
```

Определением, какой переключатель активен, занимается обработчик нажатия кнопки. Для этого он использует метод `isChecked()` — листинг 5.13.

Листинг 5.13. Java-код проекта (зависимые переключатели)

```
package com.samples.test5;

import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.Button;
import android.widget.RadioButton;
import android.view.View;

public class Test5Activity extends ActionBarActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_test5);

        final TextView txt1 = (TextView)findViewById(R.id.txt1);
        final RadioButton r1 = (RadioButton)findViewById(R.id.r1);
        final RadioButton r2 = (RadioButton)findViewById(R.id.r2);
        final Button button1 = (Button)findViewById(R.id.button1);

        button1.setOnClickListener(new View.OnClickListener() {

                    public void onClick(View v) {
                           // TODO Auto-generated method stub
                           if (r1.isChecked()) txt1.setText("Value 1");
                           if (r2.isChecked()) txt1.setText("Value 2");
                    }
              });

    }

}
```

Как видите, все довольно просто. Готовое приложение показано на рис. 5.22.

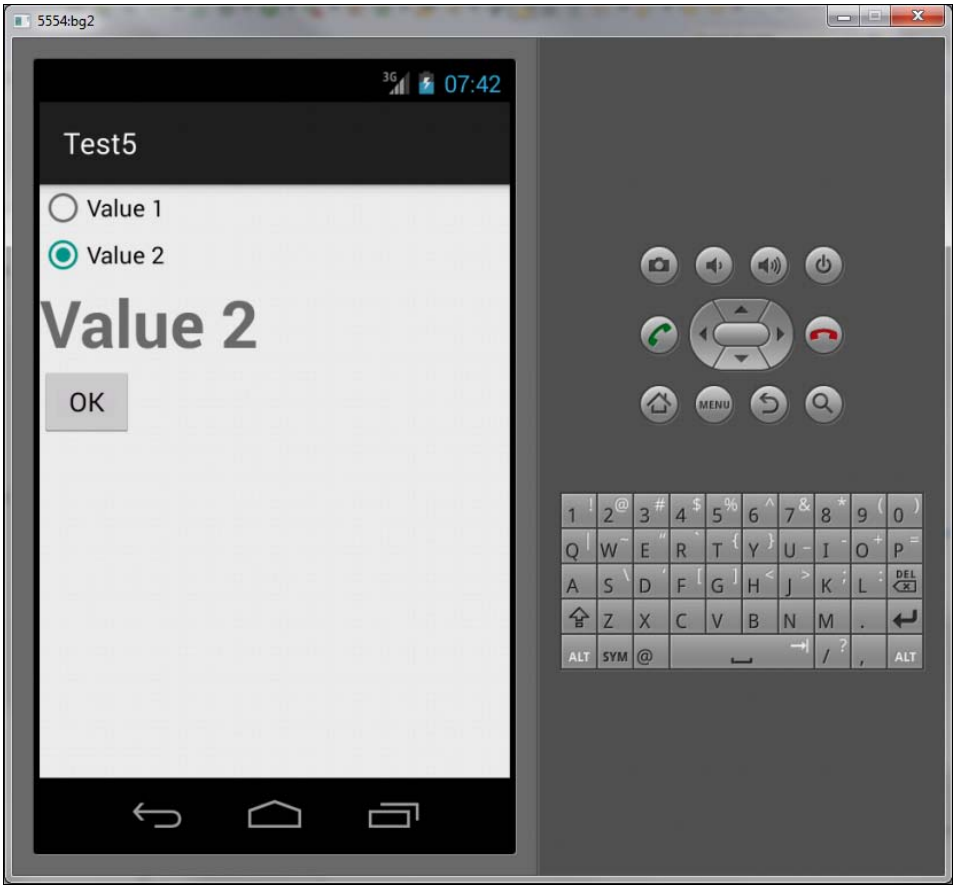


Рис. 5.22. Демонстрация использования зависимых переключателей

CheckBox — независимые переключатели

Переключатели `CheckBox` не привязываются к какому-либо контейнеру (вроде `RadioGroup`), значение независимого переключателя не зависит от состояния других переключателей, поэтому такие переключатели и называются независимыми.

Для работы с `CheckBox` вы можете использовать те же методы (`isChecked()`, `toggle()`, `setChecked()`), что и в случае с `RadioButton`. В файле разметки независимые переключатели определяются так:

```
<CheckBox
android:id="@+id/checkbox"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="I agree"/>
```

Найти `Checkbox` в Java-коде можно так:

```
final CheckBox check = (CheckBox)findViewById(R.id.checkbox);
```

Далее можно проверить, включен ли переключатель:

```
if (check.isChecked()) txt1.setText("OK");
```

ToggleButton — кнопка включено/выключено

Кнопка `ToggleButton` может находиться в одном из положений: включено или выключено. Когда кнопка нажата, горит зеленый индикатор, как бы «встроенный» в кнопку, а также видно, что кнопка немного вдавлена.

По большому счету от `ToggleButton` можно было отказаться и использовать обычный виджет `CheckBox`, однако кнопка `ToggleButton` выглядит более привлекательно и больше подходит для включения/выключения режимов программы, чем переключатель `CheckBox`.

У кнопки есть два основных свойства: `android:textOff` и `android:textOn`. Первое устанавливает текст кнопки, когда она выключена, а второе — когда включена. В программном коде этим свойствам соответствуют методы `setTextOff()` и `setTextOn()`.

С помощью метода `setChecked(boolean checked)` вы можете программно изменить состояние кнопки. При изменении состояния генерируется событие `onCheckedChanged()`.

Добавьте в наш проект кнопку `ToggleButton` и удалите все, что нам не нужно. Пусть в проекте останутся только текстовое поле `TextView`, кнопка `Button` (она нам еще пригодится) и новая кнопка `ToggleButton`. Разметка проекта представлена в листинге 5.14.

Листинг 5.14. Разметка проекта (демонстрация использования `ToggleButton`)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
    >
<TextView
        android:id="@+id/txt1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="20pt"
        android:textStyle="bold"
        android:text="@string/hello_world"
    />
<Button
        android:text="Button"
        android:id="@+id/button1"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content">
</Button>
<ToggleButton
        android:textOff="Turbo off"
        android:textOn="Turbo on"
        android:id="@+id/tb1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
</ToggleButton>
</LinearLayout>
```

При нажатии кнопки `ToggleButton` будет изменяться значение текстового поля в зависимости от состояния кнопки. Такой эффект достигается благодаря обработчику события `OnCheckedChange`. В коде программы есть еще и обработчик нажатия обычной кнопки — он просто устанавливает значение текстового поля. Вы можете от него отказаться, как и от самой кнопки `Button`. Код Java представлен в листинге 5.15, а результат показан на рис. 5.23 — кнопка `ToggleButton` нажата и текст заменен на **Turbo mode on**.

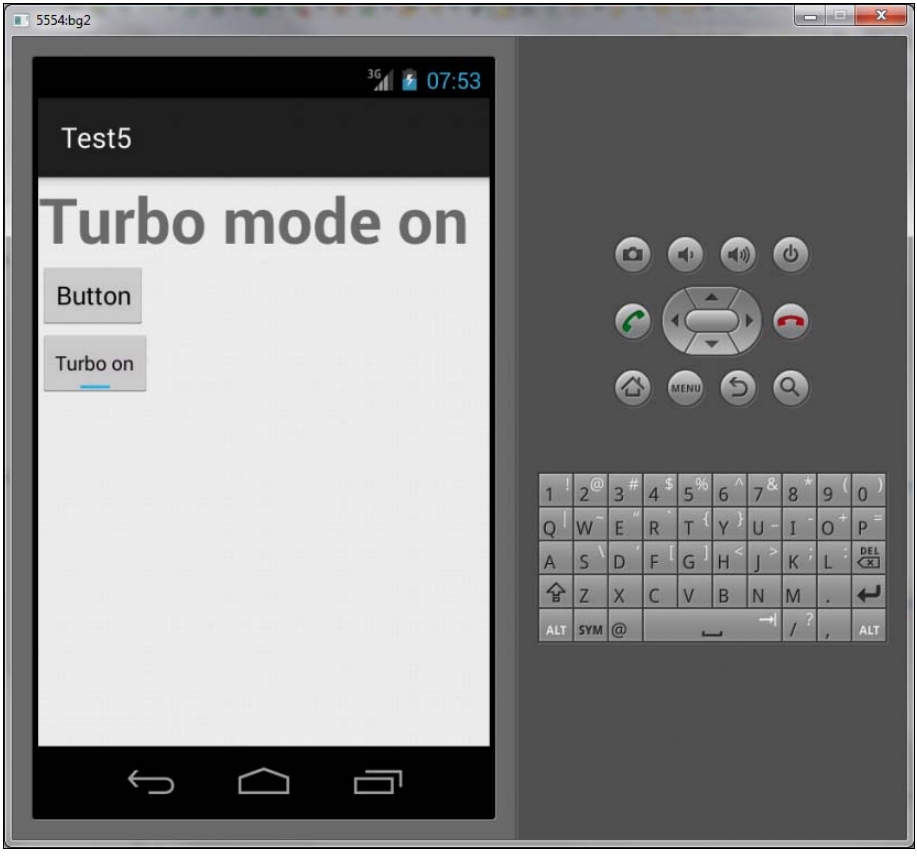


Рис. 5.23. `ToggleButton` в действии

Листинг 5.15. Java-код проекта (использование `ToggleButton`)

```
package com.samples.test5;

import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.Button;
import android.widget.CompoundButton;
import android.widget.ToggleButton;
import android.widget.CompoundButton.OnCheckedChangeListener;
import android.view.View;

public class Test5Activity extends ActionBarActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_test5);

        final TextView txt1 = (TextView)findViewById(R.id.txt1);

        final Button button1 = (Button)findViewById(R.id.button1);

        final ToggleButton tButton = (ToggleButton)findViewById(R.id.tb1);
        tButton.setOnCheckedChangeListener(new OnCheckedChangeListener() {

       public void onCheckedChanged(
               CompoundButton buttonView, boolean isChecked) {
              // TODO Auto-generated method stub
                     if (isChecked)
                       txt1.setText("Turbo mode on");
                     else
                       txt1.setText("Turbo mode off");
                     }
              });
            button1.setOnClickListener(new View.OnClickListener() {
                     public void onClick(View v) {
                            // TODO Auto-generated method stub
                            txt1.setText("Default mode");
                     }
              });

    }

}
```

ImageButton — кнопка с изображением

Виджет `ImageButton` — нечто среднее между изображением и кнопкой. Вместо текста на эту кнопку будет выведено изображение, что позволяет создавать более привлекательные кнопки.

Установить изображение кнопки можно или атрибутом `android:src` элемента `<ImageButton>`, или методом `setImageResource(int)`.

В файле разметки элемент `<ImageButton>` описывается так:

```
<ImageButton
android:id="@+id/button"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:src="@drawable/icon"/>
```

Работа с такой кнопкой осуществляется так же, как и с обычной кнопкой. Первым делом нужно подключить необходимый пакет, объявить переменную и найти сам элемент в файле разметки:

```
import android.widget.ImageButton;
...
ImageButton button;
...
button = (ImageButton)findViewById(R.id.button);
button.setImageResource(R.drawable.play);
```

Последние два оператора находят кнопку `ImageButton` и устанавливают для нее изображение с именем `play`, которое находится в папке res/drawable или res/drawable-* (имя зависит от разрешения экрана — для более новых платформ Android). Лучше всего использовать файлы в формате PNG. В нашем случае (ресурс `R.drawable.play`) в каталоге res/drawable должен быть файл play.png.

Обработчик нажатия кнопки устанавливается так же, как и для обычной кнопки `Button`.

5.2.3. Индикатор *ProgressBar*

Сейчас мы рассмотрим довольно интересный виджет — `ProgressBar`, представляющий собой индикатор (шкалу) процесса.

Начнем с разметки (листинг 5.16). Наша деятельность будет содержать два поля ввода, кнопку и, конечно же, `ProgressBar`. Редактор разметки для нашего приложения показан на рис. 5.24.

Листинг 5.16. Разметка для `ProgressBar`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
```

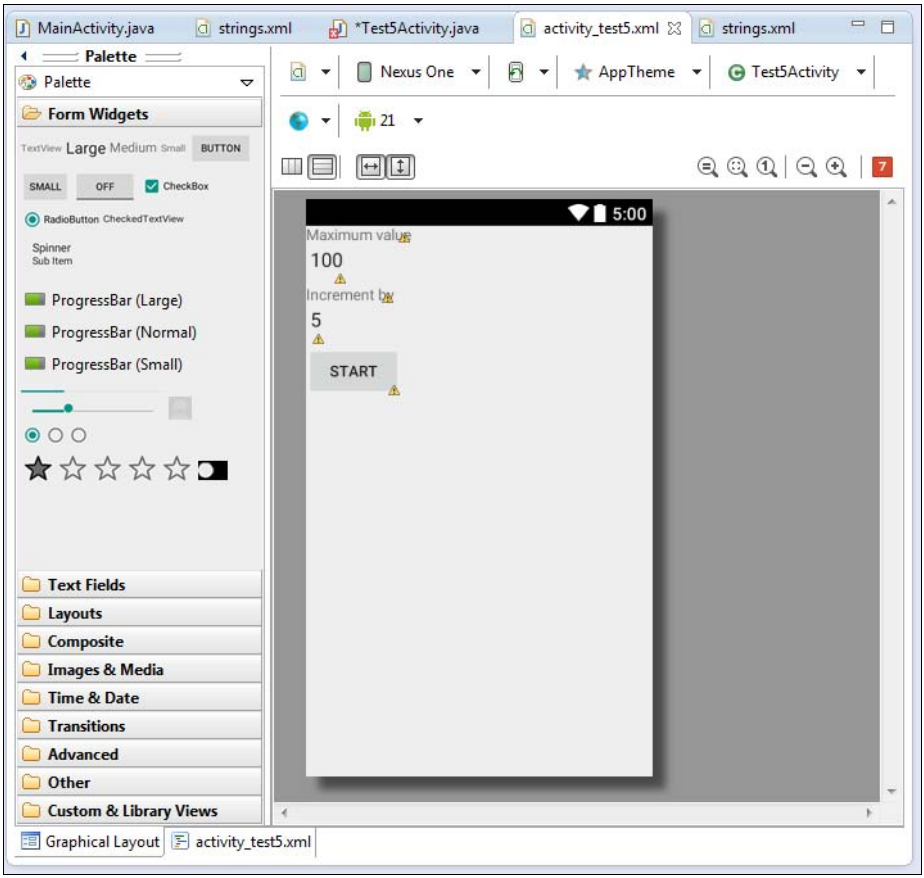


Рис. 5.24. Редактор разметки для деятельности настройки `ProgressBar`

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    >

<TextView android:id="@+id/TextView02"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="Maximum value"></TextView>

<EditText android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="100"
       android:id="@+id/maximum"></EditText>

<TextView android:id="@+id/TextView01"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="Increment by"></TextView>
```

```
<EditText android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="5"
       android:id="@+id/increment"></EditText>

<Button android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="Start"
       android:id="@+id/startbtn"></Button>

</LinearLayout>
```

Прежде чем приступать к написанию Java-кода, нужно разобраться, как станет работать наше приложение. Максимальное значение для `ProgressBar` устанавливается в поле **Maximum Value**, шаг (величина, на которую изменяется значение `ProgressBar` каждые полсекунды, или 500 мс) задается значением, указанным в поле **Increment by**. При нажатии кнопки **Start** появится диалоговое окно, в котором и будет отображен наш индикатор `ProgressBar`. Увеличение значения `ProgressBar` производится в потоке каждые 500 мс. Понимаю, что с диалоговыми окнами мы еще пока не работали, но они будут рассмотрены уже в следующей главе.

Полный исходный Java-код представлен в листинге 5.17.

Листинг 5.17. Полный Java-код приложения

```
package com.samples.test5;

import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.app.ProgressDialog;

public class Test5Activity extends Activity implements OnClickListener {
    ProgressDialog dialog;
    int increment;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_test5);

        Button startbtn = (Button) findViewById(R.id.startbtn);
        startbtn.setOnClickListener(this);
    }
```

```
public void onClick(View view) {

    // получаем шаг инкремента из текстового поля
    EditText et = (EditText) findViewById(R.id.increment);
    // конвертируем строку в число
    increment = Integer.parseInt(et.getText().toString());

    // создаем новое диалоговое окно
    dialog = new ProgressDialog(this);
    dialog.setCancelable(true);
    dialog.setMessage("Loading...");
    // шкала должна быть горизонтальной
    dialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
    // значение шкалы по умолчанию — 0
    dialog.setProgress(0);

    // получаем максимальное значение
    EditText max = (EditText) findViewById(R.id.maximum);
    // конвертируем строку в число
    int maximum = Integer.parseInt(max.getText().toString());
    // устанавливаем максимальное значение
    dialog.setMax(maximum);
    // отображаем диалоговое окно
    dialog.show();

    // создаем поток для обновления шкалы
    Thread background = new Thread (new Runnable() {
       public void run() {
           try {
               // увеличиваем значение шкалы каждые 500 мс,
               // пока не будет достигнуто максимальное значение
               while (dialog.getProgress()<= dialog.getMax()) {
                   // ждем 500 мс
                   Thread.sleep(500);

                   // активируем обработчик обновления
   progressHandler.sendMessage(progressHandler.obtainMessage());
               }
           } catch (java.lang.InterruptedException e) {

           }
       }
    });

    // запускаем фоновый поток
    background.start();

}
```

```
    // обработчик для фонового обновления
    Handler progressHandler = new Handler() {
        public void handleMessage(Message msg) {
            // увеличиваем значение шкалы
            dialog.incrementProgressBy(increment);
        }
    };
}
```

Запустите приложение, установите параметры (рис. 5.25) и нажмите кнопку **Start** — вы увидите диалоговое окно и работающий в нем `ProgressBar` (рис. 5.26).

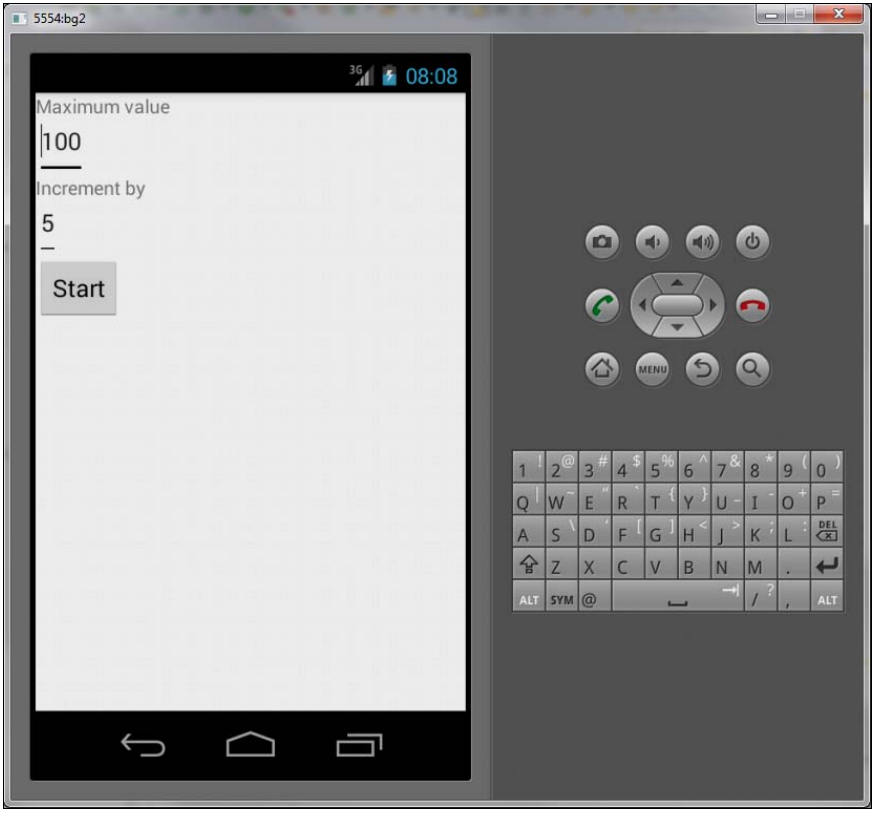


Рис. 5.25. Установка параметров `ProgressBar`

5.2.4. Средства отображения графики

Отображение графики осуществляется с помощью базового виджета `ImageView`. Для загрузки изображения в классе `ImageView` существует несколько методов:

- `setImageResource(int resId)` — загружает изображение из ресурса;
- `setImageURI(Uri uri)` — загружает изображение по его URI;
- `setImageBitmap(Bitmap bitmap)` — загружает растровое изображение.

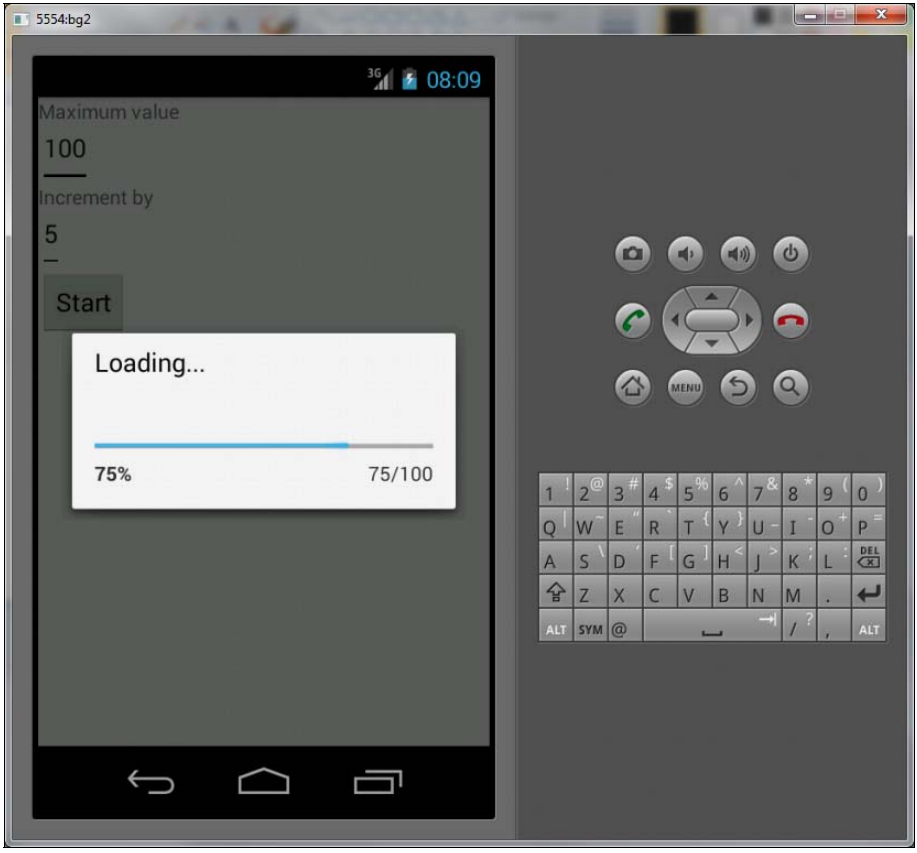


Рис. 5.26. Шкала в действии

Для изменения размера изображения служат следующие методы:

- `setMaxHeight()` — устанавливает максимальную высоту;
- `setMaxWidth()` — устанавливает максимальную ширину.

Если вы хотите загрузить изображение из файла разметки, то воспользуйтесь атрибутом `android:src`. При добавлении `ImageView` с помощью редактора разметки откроется окно, позволяющее выбрать изображение из списка ресурсов проекта или системных ресурсов (рис. 5.27).

В файле разметки виджет `ImageView` определяется так:

```
<ImageView
        android:id="@+id/imageView1"
        android:contentDescription="@string/desc"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/abc_btn_check_material" />
```

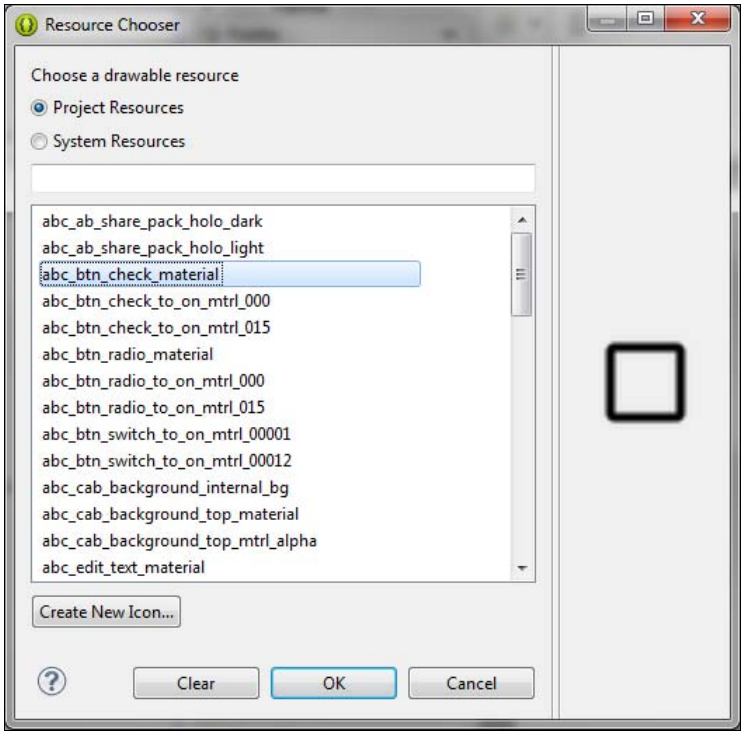


Рис. 5.27. Выбор ресурса

После этого в Java-коде работа с виджетом производится так:

```
final ImageView image = (ImageView)findViewById(R.id.imageView1);
// загружаем изображение из ресурса
image.setImageResource(R.drawable.icon);
```

5.2.5. Виджеты *AnalogClock* и *DigitalClock*

Виджеты `AnalogClock` (аналоговые часы) и `DigitalClock` (цифровые часы) используются для отображения системного времени.

Добавьте в проект оба виджета — они находятся на вкладке **Time & Date** редактора разметки, там же вы найдете средства не только отображения, но и выбора даты и времени.

Разметка приложения приведена в листинге 5.18. Сами виджеты в действии показаны на рис. 5.28.

Листинг 5.18. Разметка приложения с часами

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical" >
```

```
    <DigitalClock
        android:id="@+id/digitalClock1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="DigitalClock" />

    <AnalogClock
        android:id="@+id/analogClock1"
        android:layout_width="214dp"
        android:layout_height="wrap_content"
        android:layout_weight="0.29" />

</LinearLayout>
```

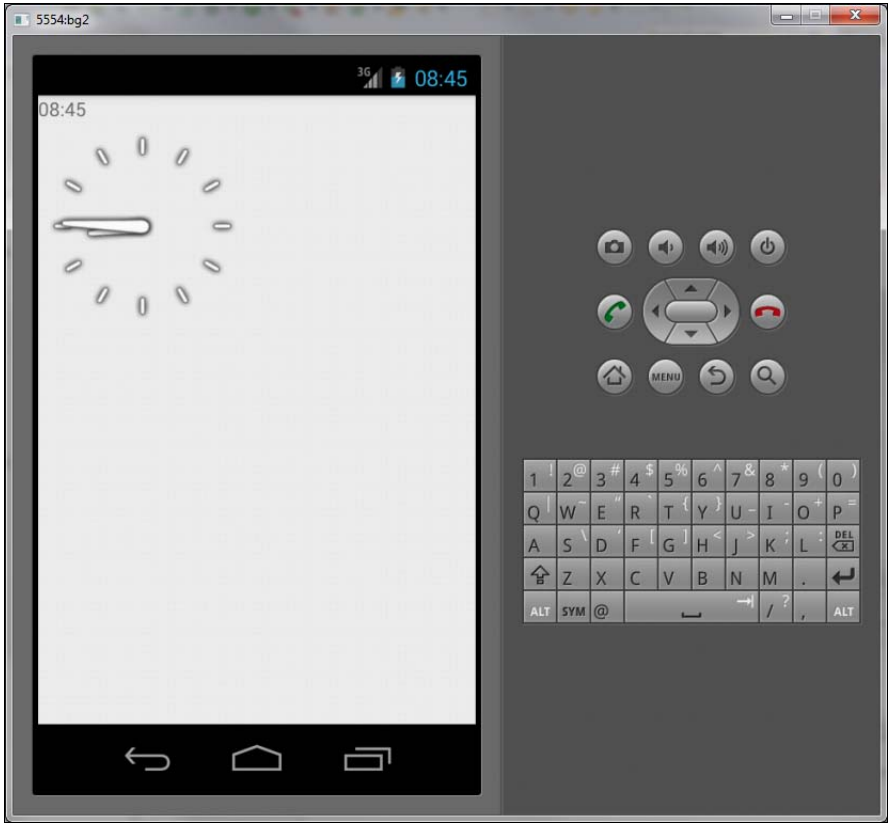


Рис. 5.28. Виджеты `AnalogClock` и `DigitalClock`

5.2.6. Использование компонента *DatePicker*

В секции **Time & Date** есть два очень важных компонента: **TimePicker** и **DatePicker**. Оба компонента используются аналогично, поэтому далее будет рассмотрен только **DatePicker**.

Стоит отметить, что этот компонент требует минимального уровня API 11 (Android 3.0), поэтому если в файле AndroidManifest.xml у вас указан минимальный уровень API 8, то это нужно исправить:

```
<uses-sdk
        android:minSdkVersion="11"
        android:targetSdkVersion="21" />
```

Далее надо расположить компоненты на форме, как показано на рис. 5.29. В листинге 5.19 приведена разметка этого приложения.

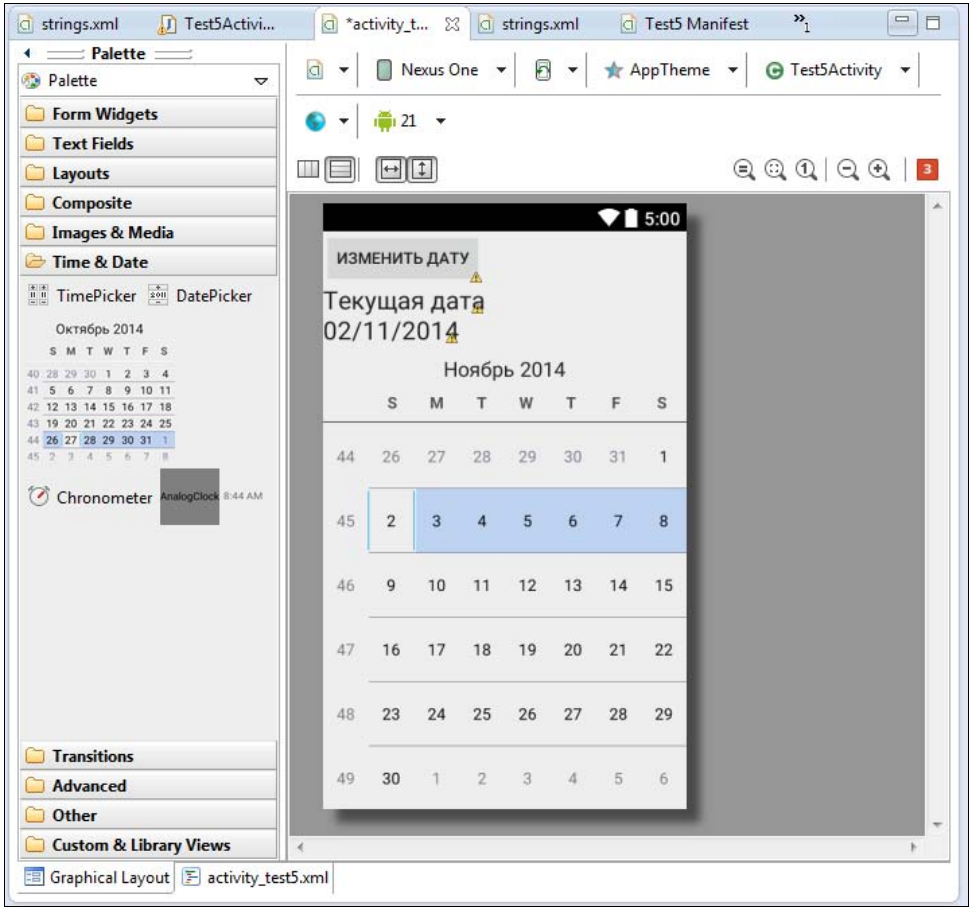


Рис. 5.29. Использование компонента **DatePicker**

Листинг 5.19. Файл разметки приложения выбора даты

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
```

```
    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Изменить дату" />

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Текущая дата"
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="02/11/2014"
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <CalendarView
        android:id="@+id/calendarView1"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

Java-код для **DataPicker** сводится к его инициализации и указанию обработчика события изменения даты. При инициализации указывается текущая дата и обработчик изменения даты. Вот пример такого кода:

```
import java.util.Calendar;
import android.widget.DatePicker.OnDateChangedListener;
import android.widget.Button;
import android.widget.DatePicker;
import android.widget.TextView;
...
CurDateTv2 = (TextView)findViewById(R.id.textView2);
Picker = (DatePicker)findViewById(R.id.calendarView1);
btnChangeDate = (Button)findViewById(R.id.button1);

final Calendar c = Calendar.getInstance();
year = c.get(Calendar.YEAR);
month = c.get(Calendar.MONTH);
day = c.get(Calendar.DAY_OF_MONTH);

Picker.init(year, month, day, new OnDateChangedListener() {
                    @Override
```

```
        public void onDateChanged(DatePicker view, int year, int monthOfYear,int
dayOfMonth) {
                CurDateTv2.setText(new StringBuilder()
                // нумерация месяцев начинается с 0, поэтому нужно увеличить
                // значение month на 1
                .append(dayOfMonth).append("/").append(monthOfYear + 1)
                 .append("/").append(year).append(" "));

        }
});
```

Обработчик события не делает ничего сверхъестественного, а просто устанавливает дату, которую выбрал пользователь, в качестве значения `textView`.

Компоненты **TimePicker** и **DatePicker** редко кто использует в качестве виджетов. Как правило, в этих целях создаются диалоговые окна выбора времени и даты, с которыми мы познакомимся в следующей главе.

Итак, здесь мы рассмотрели основные виджеты. С остальными виджетами вы можете познакомиться, прочитав руководство разработчика Android:

http://developer.android.com/reference/android/package-summary.html

ГЛАВА 6

Уведомления, диалоговые окна и меню

6.1. Уведомления

Приложения могут отображать два типа уведомлений: краткие всплывающие сообщения (Toast Notification) и постоянные напоминания (Status Bar Notification). Первые отображаются на экране мобильного устройства какое-то время и не требуют внимания пользователя. Как правило, это не критические информационные сообщения. Вторые постоянно отображаются в строке состояния и требуют реакции пользователя.

Например, приложение требует подключения к вашему серверу. Если соединение успешно установлено, можно отобразить краткое уведомление, а вот если подключиться не получилось, тогда отображается постоянное уведомление, чтобы пользователь сразу мог понять, почему приложение не работает.

Чтобы отобразить всплывающее сообщение, используйте класс `Toast` и его методы `makeText` (создает текст уведомления) и `show` (отображает уведомление):

```
Context context = getApplicationContext();
Toast toast = Toast.makeText(context, "This is notification",
                             Toast.LENGTH_LONG);
toast.show();
```

Первый параметр метода `makeText()` — это контекст приложения, который можно получить с помощью вызова `getApplicationContext()`. Второй параметр — текст уведомления. Третий — задает продолжительность отображения уведомления:

- `LENGTH_SHORT` — небольшая продолжительность (1–2 секунды) отображения текстового уведомления;
- `LENGTH_LONG` — показывает уведомление в течение более длительного периода времени (примерно 4 секунды).

По умолчанию всплывающее уведомление появится в нижней части экрана. Чтобы отобразить уведомление в другом месте, можно воспользоваться методом `setGravity()`, который нужно вызвать до метода `show()`:

```
toast.setGravity(Gravity.CENTER, 0, 0);
```

Первый параметр задает размещение в пределах большего контейнера — например, `GRAVITY.CENTER`, `GRAVITY.TOP` и т. д. Второй параметр — это смещение по оси *X*, третий — смещение по оси *Y*.

В нашем примере уведомление будет отображено по центру окна.

Теперь немного практики. Создайте новый проект (пусть он называется `Test6` — для совместимости с моим кодом), разметку можете не изменять, а можете вообще удалить все элементы деятельности. Наше приложение отобразит при запуске уведомление (рис. 6.1).

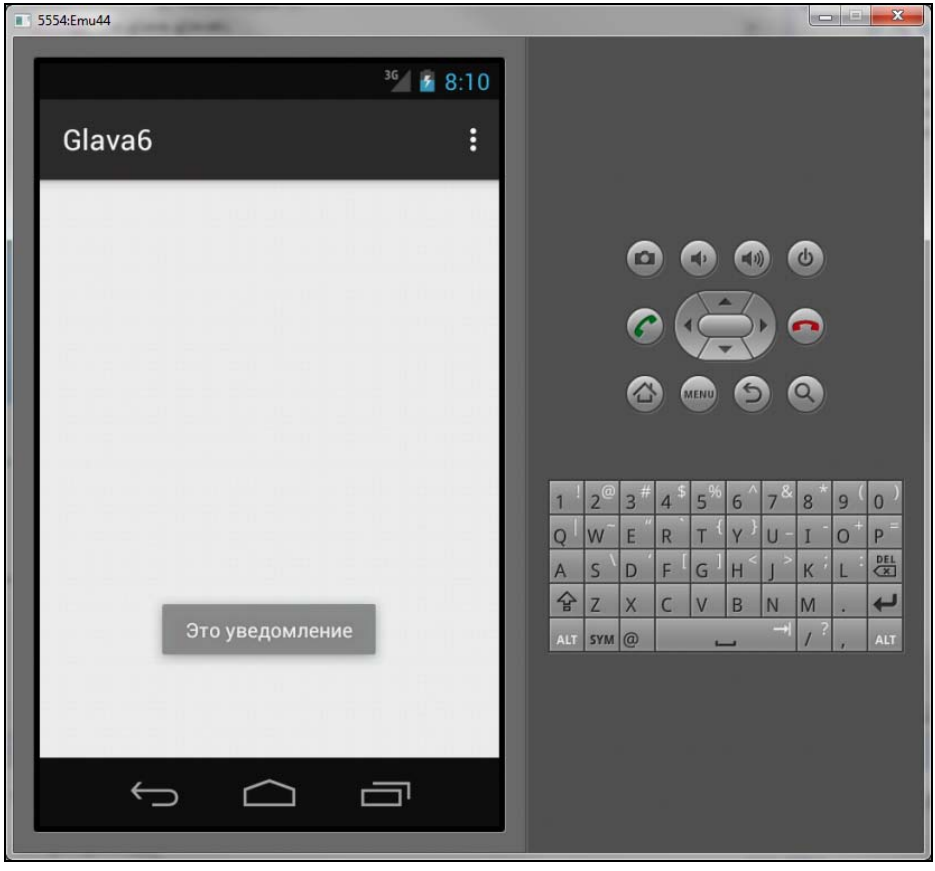


Рис. 6.1. Всплывающее уведомление

Java-код приложения представлен в листинге 6.1.

Листинг 6.1. Отображение всплывающего уведомления

```
package com.samples.test6;

import android.app.Activity;
import android.os.Bundle;
```

```
import android.widget.Toast;
import android.content.Context;
import android.view.Gravity;

public class Test6Activity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        Context context = getApplicationContext();
        Toast toast = Toast.makeText(context, "Это уведомление",
Toast.LENGTH_SHORT);
        toast.setGravity(Gravity.CENTER, 0, 0);
        toast.show();
    }
}
```

Создать уведомление в строке состояния немного сложнее. И тем более в современных версиях Android. Ведь раньше для создания таких уведомлений использовались классы `Notification` и `NotificationManager`. В Android 4 и 5 вместо класса `Notification` нужно использовать класс `Notification.Builder`. В листинге 6.2, *а* приводится старая версия кода, а в листинге 6.2, *б* — новая (чтобы вы могли сравнить).

Листинг 6.2, *а*. Старая версия кода (классы `Notification` и `NotificationManager`)

```
int NOTIFY_ID = 101;

Context context = getApplicationContext();
NotificationManager Mgr =
(NotificationManager)getSystemService(Context.NOTIFICATION_SERVICE);

int icon = R.drawable.icon;
CharSequence cText = "Ошибка!";
long t = System.currentTimeMillis();
Notification notify = new Notification(icon, cText, t);
CharSequence nTitle = "Ошибка";
CharSequence nText = "Не могу подключиться к серверу";
Intent intent = new Intent(this, Test6Activity.class);
PendingIntent cIntent = PendingIntent.getActivity(this, 0, intent, 0);
notify.setLatestEventInfo(context, nTitle, nText, cIntent);

Mgr.notify(NOTIFY_ID, notify);
```

Листинг 6.2, *б*. Новая версия кода для Android 4 и 5 (класс `Notification.Builder`)

```
package com.glava.glava6;

import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.content.Context;
import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Intent;
import android.content.res.*;

public class MainActivity extends ActionBarActivity {

        private static final int NOTIFY_ID = 101;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Context context = getApplicationContext();

       Intent notificationIntent = new Intent(context, MainActivity.class);
       PendingIntent contentIntent = PendingIntent.getActivity(context,
                                     0, notificationIntent,
                                     PendingIntent.FLAG_CANCEL_CURRENT);

       NotificationManager nm = (NotificationManager) context
                               .getSystemService(Context.NOTIFICATION_SERVICE);

       Resources res = context.getResources();
       Notification.Builder builder = new Notification.Builder(context);

       builder.setContentIntent(contentIntent)
              .setSmallIcon(R.drawable.ic_launcher)
               // большая картинка
               // .setLargeIcon(R.drawable.ic_launcher)
               //.setTicker(res.getString(R.string.warning)) // текст в строке
состояния
              .setTicker("Ошибка!")
              .setWhen(System.currentTimeMillis()
              .setAutoCancel(true)
```

```
                // Заголовок уведомления
                .setContentTitle("Ошибка!")
                // Текст уведомления
                .setContentText("Не могу подключиться к серверу");
                Notification n = builder.getNotification();
                //Notification n = builder.build();
                nm.notify(NOTIFY_ID, n);
    }

}
```

Как вы могли заметить, в листинге 6.2, *б* приводится полный код приложения, а не его фрагмент. Код хоть и хорошо закомментирован, но есть некоторые нюансы, которые вам следует знать. Метод `setSmallIcon()` формирует маленькую картинку уведомления, а `setLargeIcon()` — большую. Я большую не указал, поскольку мне для демо-проекта ее готовить не захотелось. Однако в реальном приложении вы этот метод будете использовать часто. Метод `setTicker()` устанавливает в строке состояния текст сообщения, которое исчезнет, как только вы его просмотрите, — останется только картинка, заданная с помощью `setSmallIcon()`. Методы `setContentTitle()` и `setContentText()` задают, соответственно, заголовок и текст уведомления.

Метод `getNotification()` считается уже устаревшим, и на смену ему пришел метод `build()`. Однако метод `build()` можно использовать, если только в настройках проекта установлен минимальный уровень API 16 (Android 4.1.2). Поэтому, если вы пишете приложение для самых современных версий Android, устанавливайте минимальный уровень API 16 или выше и используйте метод `build()`. Если же вам нужно написать приложение, которое должно работать пусть не в самых древних версиях (та же 4.0 — это API 15), используйте метод `getNotification()` и не обращайте внимания на предупреждение об устаревшем коде. Приложения с этим методом будут нормально выполняться и на более новых версиях Android.

Результат наших стараний приведен на рис. 6.2.

Но это еще не все. Вы можете отменить собственные уведомления, когда в них пропадает необходимость (например, ваша программа уже может установить соединение с сервером, а уведомление об ошибке все еще продолжает «висеть» в строке уведомления):

```
nm.cancel(NOTIFY_ID); // nm - экземпляр класса NotificationManager
```

Если нужно в уведомлении выводить индикатор процесса (вы видели такой при установке программ из Play Маркет), используйте метод `setProgress()`:

```
setProgress(100, 75, false);
```

При появлении уведомления можно заставить смартфон звучать (`SOUND`), вибрировать (`VIBRATE`), а также мерцать (`LIGHT`). Для всего этого используются следующие константы:

- `Notification.DEFAULT_SOUND`;
- `Notification.DEFAULT_VIBRATE`;
- `Notification.DEFAULT_LIGHTS`.

Нужное поведение задается так:

```
notification.defaults = Notification.DEFAULT_SOUND |
        Notification.DEFAULT_VIBRATE;
```

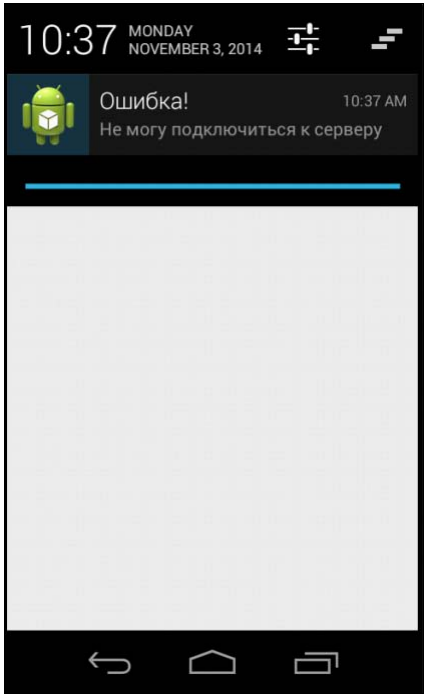


Рис. 6.2. Уведомление в строке состояния (Android 4.4)

Если вам нужно все и сразу, используйте константу `Notification.DEFAULT_ALL`. Только не забудьте для доступа к виброзвонку в файл манифеста добавить строчку:

```
<uses-permission android:name="android.permission.VIBRATE"/>
```

6.2. Диалоговые окна

В предыдущей главе мы познакомились с диалоговым окном `ProgressDialog`, отображающим индикатор `ProgressBar`, поэтому этот тип диалогового окна мы рассматривать не станем. Но кроме `ProgressDialog`, в Android предусмотрены диалоговые окна следующих типов:

- `AlertDialog` — диалоговое окно с кнопками;
- `DatePickerDialog` — диалоговое окно выбора даты;
- `TimePickerDialog` — диалоговое окно выбора времени.

6.2.1. *AlertDialog*

Самый частый вариант применения диалогового окна `AlertDialog` — это классическое диалоговое окно вопроса с кнопками **Да** и **Нет**. Вот сейчас мы и займемся его разработкой. После запуска наше приложение отобразит диалоговое окно (рис. 6.3), в котором при нажатии кнопки **Нет** не будет произведено никаких действий — будет вызван метод `cancel()`. А вот действие при нажатии кнопки **Да** вы запрограммируете сами — но позже, когда научитесь устанавливать интернет-соединения. Код приложения, отображающего это диалоговое окно, представлен в листинге 6.3.

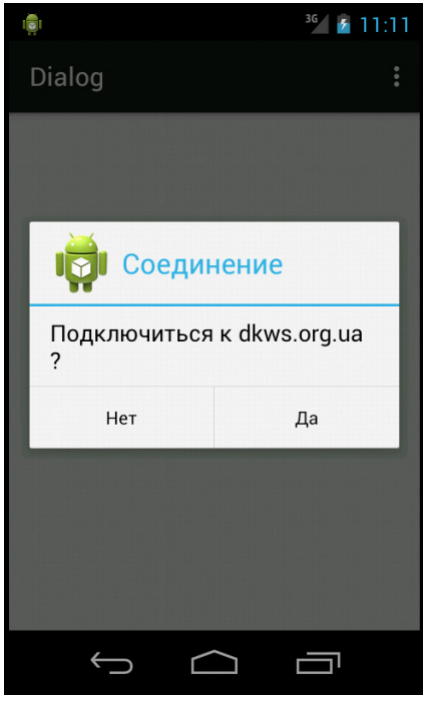


Рис. 6.3. Диалоговое окно типа `AlertDialog`

Листинг 6.3. Отображение диалогового окна типа `AlertDialog`

```
package com.sample.dialog;

import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.content.DialogInterface;
import android.app.AlertDialog;

public class MainActivity extends ActionBarActivity {

    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        AlertDialog.Builder alt_bld = new AlertDialog.Builder(this);
        // Сообщение диалога
           alt_bld.setMessage("Подключиться к dkws.org.ua ?")
           .setCancelable(false)
           .setPositiveButton("Да", new DialogInterface.OnClickListener() {
           public void onClick(DialogInterface dialog, int id) {
           // Действие для кнопки Да
           }
           })
           .setNegativeButton("Нет", new DialogInterface.OnClickListener() {
           public void onClick(DialogInterface dialog, int id) {
           //  Действие для кнопки Нет
           dialog.cancel();
           }
           });
           AlertDialog alert = alt_bld.create();
           // Title for AlertDialog
           alert.setTitle("Соединение");
           // Icon for AlertDialog
           alert.setIcon(R.drawable.ic_launcher);
           alert.show();

    }

}
```

Далее мы рассмотрим диалоговые окна выбора даты и времени.

6.2.2. *DatePickerDialog*

Диалоговые окна выбора даты и времени весьма «многострадальные». С ними вечно все не так. Мало того, что код вызова таких диалоговых окон довольно непростой, так еще и меняется из версии в версию API. Так, для показа диалогового окна ранее использовался метод `ShowDialog()`, который сейчас применять не рекомендуется. Вместо него нужно задействовать вот такую конструкцию, вовлекающую метод `show()`:

```
DatePickerDialog dpg =  new DatePickerDialog(this, mDateSetListener, mYear,
mMonth, mDay);
dpg.show();
```

А что, если нужно, чтобы программа была максимально совместима со всеми версиями? Тогда следует использовать `ShowDialog()` и в то же время обеспечить,

чтобы среда разрешила компиляцию. Для этого нужно добавить директиву `@SuppressWarnings("deprecation")`. Да, решение не очень хорошее, но зато работает от API 8 (2.*x*) до API 21 (5.0).

Я старался максимально упростить приводимый здесь код. Вы можете сравнить приведенный далее код с кодом (и решением) из руководства разработчика:

http://developer.android.com/guide/topics/ui/controls/pickers.html

Само приложение тоже очень простое. Оно содержит только надпись с призывом выбрать дату (рис. 6.4). При щелчке по надписи открывается диалоговое окно выбора даты (рис. 6.5), а после выбора даты и нажатия кнопки **Done** выбранная пользователем дата отображается в надписи (рис. 6.6).

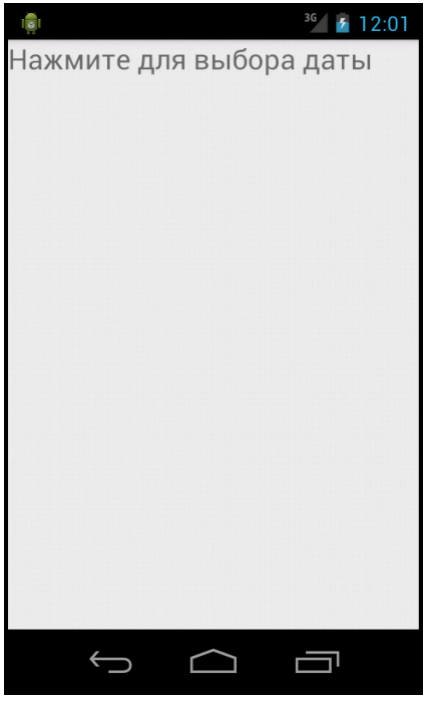


Рис. 6.4. Щелкните по надписи для выбора даты

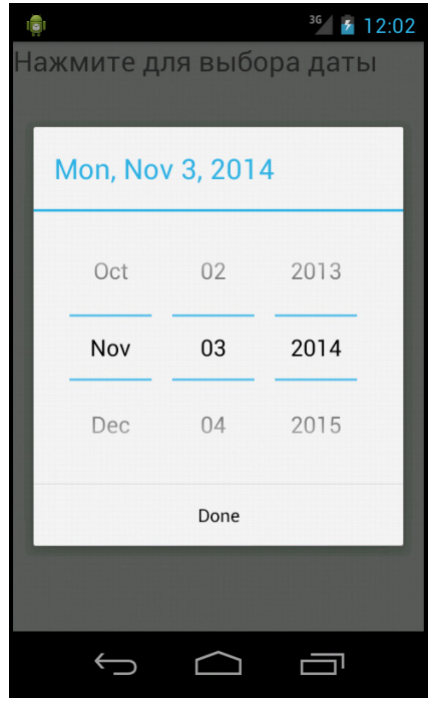


Рис. 6.5. Диалоговое окно выбора даты

Разметка приложения приведена в листинге 6.4, а Java-код приложения — в листинге 6.5.

Листинг 6.4. Разметка приложения

```
<?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">
```

```
<TextView
   android:id="@+id/tvDate"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:clickable="true"
   android:onClick="onclick"
   android:text="Нажмите для выбора даты"
   android:textSize="22sp">
</TextView>
</LinearLayout>
```

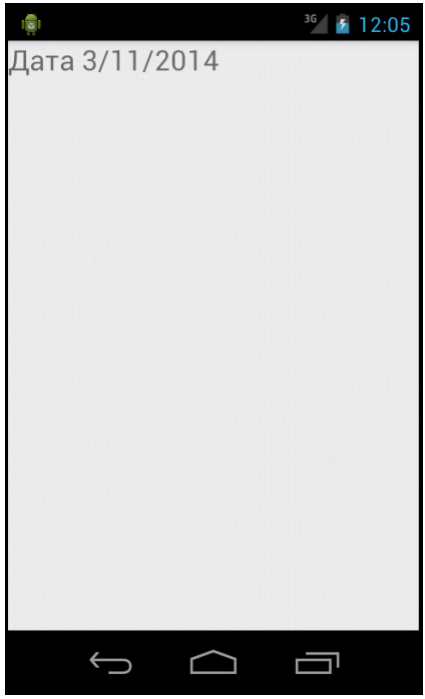


Рис. 6.6. Выбранная дата отображена в текстовой надписи

Листинг 6.5. Выбор даты

```
package com.sample.selectdatedlg;

import android.app.Activity;
import android.app.DatePickerDialog;
import android.app.DatePickerDialog.OnDateSetListener;
import android.app.Dialog;
import android.os.Bundle;
import android.view.View;
import android.widget.DatePicker;
import android.widget.TextView;
```

```
public class MainActivity extends Activity {

  int DIALOG_DATE = 1;
  // Дата для инициализации (можно получить через Calendar текущую, но
  // я старался сделать код как можно проще).
  int myYear = 2014;
  int myMonth = 10;   // ноябрь, нумерация месяцев начинается с 0!
  int myDay = 03;
  TextView tvDate;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        tvDate = (TextView) findViewById(R.id.tvDate);
    }

// Добавил директиву, чтобы среда не "ругалась" на старый код
    @SuppressWarnings("deprecation")
    public void onclick(View view) {
      showDialog(DIALOG_DATE);
    }



    @SuppressWarnings("deprecation")
    protected Dialog onCreateDialog(int id) {
      if (id == DIALOG_DATE) {
        DatePickerDialog tpd = new DatePickerDialog(this, myCallBack, myYear,
myMonth, myDay);
        return tpd;
      }
      return super.onCreateDialog(id);
    }

    OnDateSetListener myCallBack = new OnDateSetListener() {

    public void onDateSet(DatePicker view, int year, int monthOfYear,
        int dayOfMonth) {
      myYear = year;
      // увеличиваем полученный месяц для отображения
      myMonth = monthOfYear + 1;
      myDay = dayOfMonth;
      tvDate.setText("Дата " + myDay + "/" + myMonth + "/" + myYear);
    }
    };
}
```

6.2.3. *TimePickerDialog*

Сейчас мы напишем приложение выбора времени, но оно будет немного отличаться от предыдущего примера. Вместо надписи у нас будет кнопка, вызывающая диалоговое окно выбора времени (рис. 6.7), а само выбранное время будет отображено в качестве уведомления (рис. 6.8). Разметка приложения приведена в листинге 6.6, *а*, а полный код приложения — в листинге 6.6, *б*.

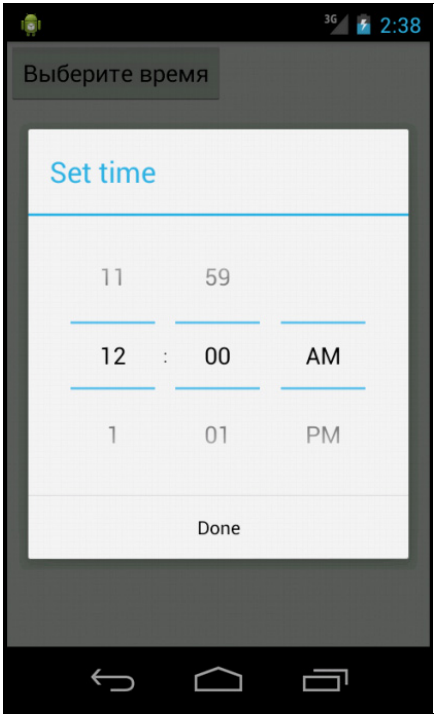


Рис. 6.7. Диалоговое окно выбора времени

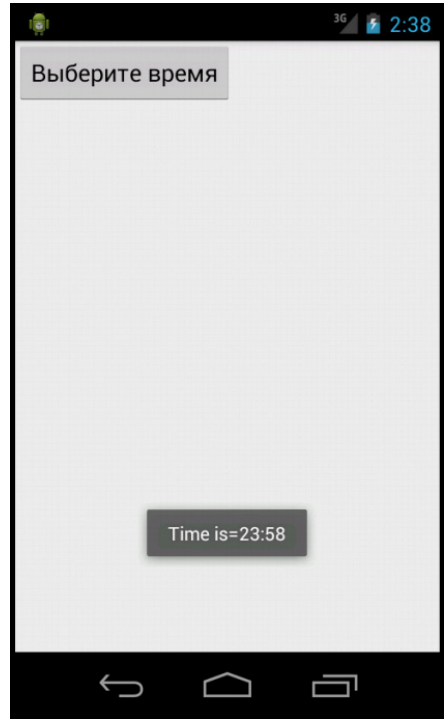


Рис. 6.8. Выбранное пользователем время

Листинг 6.6, *а*. Выбор времени: разметка приложения

```
<?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"
    >
    <Button android:text="Выберите время"
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
</Button>
</LinearLayout>
```

Листинг 6.6, *б*. Выбор времени: код приложения

```
package com.sample.timeselect;

import android.app.Activity;
import android.os.Bundle;
import android.app.Dialog;
import android.app.TimePickerDialog;
import android.widget.Button;
import android.widget.TimePicker;
import android.widget.Toast;
import android.view.View;

public class MainActivity extends Activity {
       private Button b1;
       static final int TIME_DIALOG_ID = 0;

       @Override
        protected void onCreate(Bundle savedInstanceState) {
              super.onCreate(savedInstanceState);
              setContentView(R.layout.activity_main);
              // Кнопка
              b1 = (Button) findViewById(R.id.button1);
              b1.setOnClickListener(new View.OnClickListener() {
              // Обработчик нажатия кнопки — вызываем диалог
                     @SuppressWarnings("deprecation")
                     public void onClick(View v) {
                            showDialog(TIME_DIALOG_ID);
                     }
        });
        }

        private TimePickerDialog.OnTimeSetListener mTimeSetListener =
        new TimePickerDialog.OnTimeSetListener() {
        // Обработчик нажатия кнопки Set диалога выбора времени
         public void onTimeSet(TimePicker view, int hourOfDay,
  int minute) {
      // Отображаем уведомление с выбранным пользователем временем
          Toast.makeText(MainActivity.this,
                     "Time is="+hourOfDay+":"+minute,
                     Toast.LENGTH_LONG).show();
        }
        };
        @Override
        protected Dialog onCreateDialog(int id) {
               switch (id) {
                     case TIME_DIALOG_ID:
```

```
return new TimePickerDialog(this, mTimeSetListener, 0, 0, false);
                }
        return null;
        }
}
```

6.3. Меню

Уведомление и диалоговые окна — это хорошо, но редко какое приложение обходится без меню.

Операционная система Android предлагает три вида меню:

- `OptionsMenu` — меню выбора опций, появляется внизу экрана при нажатии кнопки **Menu** на мобильном устройстве;
- `ContextMenu` — контекстное меню, появляется при долгом касании (две или более секунды) сенсорного экрана;
- `Submenu` — подменю, привязывается к конкретному пункту меню (меню выбора опций или контекстного меню). Пункты подменю не поддерживают вложенного меню.

В случае с `OptionsMenu` существует два типа меню:

- `IconMenu` — меню со значками, добавляет значки к тексту в пункты меню. Это единственный тип меню, поддерживающий значки. В `IconMenu` может быть максимум шесть пунктов;
- `ExpandedMenu` — расширенное меню, представляет собой вертикальный выпадающий список меню с кнопкой **More**, открывающей это расширенное меню. Характерно только для старых версий Android, в новых меню просто отображается в виде единого списка команд без кнопки **More**.

6.3.1. Меню выбора опций

Начнем с создания самого часто использующегося меню — меню выбора опций, которое появляется, когда пользователь нажмет кнопку **Menu** на мобильном устройстве.

Меню можно создать в файле разметки или с помощью метода `add()`. Мы будем использовать второй способ, оставив файл разметки без изменений.

Первым делом нужно определить идентификаторы создаваемых пунктов меню, делается это так:

```
public static final int IDM_NEW = 101;
public static final int IDM_OPEN = 102;
public static final int IDM_SAVE = 103;
public static final int IDM_EXIT = 104;
```

В нашем меню будет четыре пункта с идентификаторами `IDM_NEW`, `IDM_OPEN`, `IDM_SAVE`, `IDM_EXIT`.

Далее для каждого пункта меню нужно вызвать метод `add()`:

```
menu.add(Menu.NONE, IDM_NEW, Menu.NONE, "Новая
игра").setAlphabeticShortcut('n');
```

В нашем случае мы не только добавляем новый пункт меню, но и устанавливаем для него клавишу быстрого доступа. Если клавиша быстрого доступа не нужна, то оператор добавления нового пункта меню можно записать короче:

```
menu.add(Menu.NONE, IDM_NEW, Menu.NONE, "Новая игра");
```

Методу `add()` нужно передать четыре параметра:

- идентификатор группы меню — используется для создания сложных меню и позволяет связать новый пункт меню с группой других его пунктов. В нашем случае можно воспользоваться значением `Menu.NONE`, потому что идентификатор группы задавать не нужно;
- идентификатор меню — позволяет однозначно идентифицировать пункт меню, далее идентификатор поможет определить, какой пункт меню был выбран пользователем;
- порядок расположения пункта в меню — по умолчанию пункты размещаются в меню в порядке их добавления методом `add()`, поэтому в качестве значения этого параметра можно тоже указать `Menu.NONE`;
- заголовок — задает заголовок пункта меню, видимый пользователем. Вы можете задать как текстовую константу, так и указать строковый ресурс.

Теперь создайте новый проект с именем `OptionsMenu`, имя пакета — `com.samples.optmenu`. Файл разметки оставьте по умолчанию, только измените строковый ресурс `hello` и вместо `"Hello World!"` установите значение `"Нажмите кнопку Меню"` (рис. 6.9). Этим мы установим подсказку для пользователя, чтобы он знал, что нужно нажать кнопку **Menu**. В нашем случае пользователем являетесь вы, но все равно измените строковый ресурс — вдруг, пока запустится эмулятор, вы забудете, что хотели сделать.

Наше приложение будет работать так. Пользователь при нажатии кнопки **Menu** видит меню, состоящее из четырех пунктов (рис. 6.10). При выборе пункта меню пользователь видит уведомление, соответствующее выбранному пункту меню (рис. 6.11). Java-код приложения представлен в листинге 6.7.

Листинг 6.7. Меню опций

```
package com.samples.optmenu;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
```

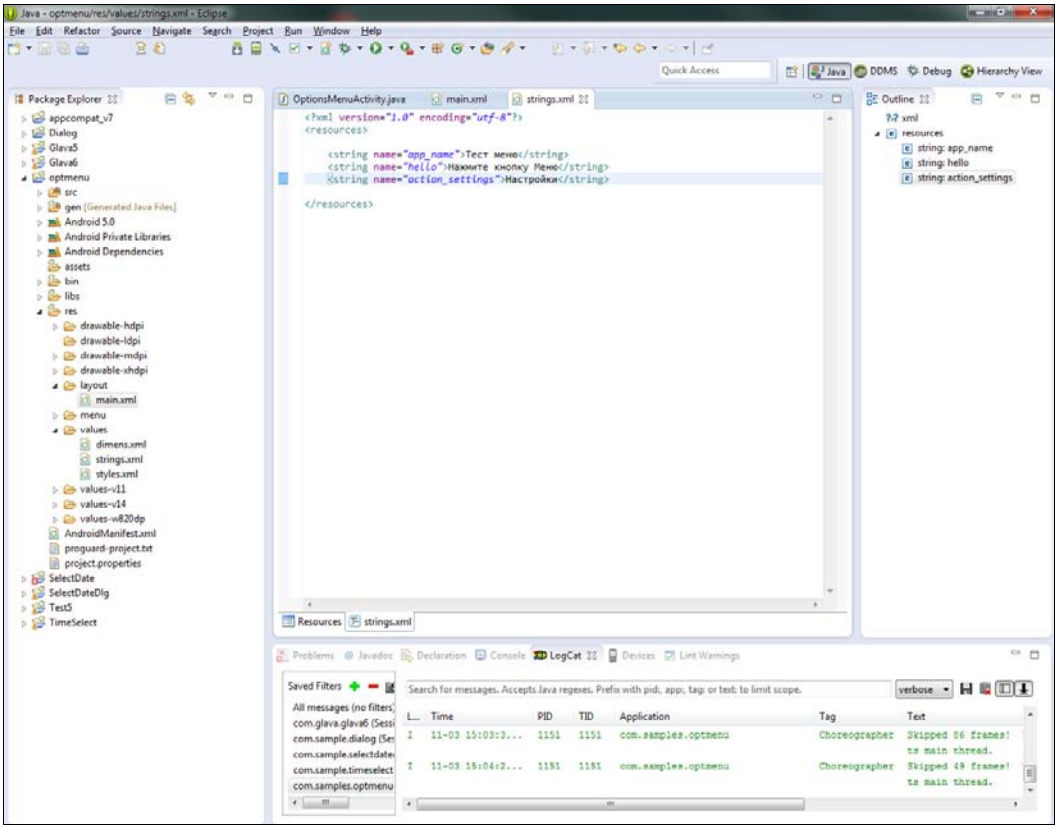


Рис. 6.9. Редактирование строкового ресурса

```
import android.widget.Toast;
import android.view.Gravity;
public class OptionsMenuActivity extends Activity {
// Описываем идентификаторы пунктов меню
        public static final int IDM_NEW = 101;
        public static final int IDM_OPEN = 102;
        public static final int IDM_SAVE = 103;
        public static final int IDM_EXIT = 104;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
    // Создаем пункты меню
       menu.add(Menu.NONE, IDM_NEW, Menu.NONE, "Новая игра");
       menu.add(Menu.NONE, IDM_OPEN, Menu.NONE, "Открыть игру");
```

```
        menu.add(Menu.NONE, IDM_SAVE, Menu.NONE, "Сохранить игру");
        menu.add(Menu.NONE, IDM_EXIT, Menu.NONE, "Выход");

        return(super.onCreateOptionsMenu(menu));
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        CharSequence t;
        // Устанавливаем реакции на выбор пункта меню
        // В нашем случае устанавливается текст уведомления
        switch (item.getItemId()) {
        case IDM_NEW: t = "Новая игра"; break;
        case IDM_OPEN: t = "Открыть игру"; break;
        case IDM_SAVE: t = "Сохранить игру"; break;
        case IDM_EXIT: t = "Выход"; break;
        default: return false;
        }
        Toast toast = Toast.makeText(this, t, Toast.LENGTH_LONG);
        toast.setGravity(Gravity.CENTER, 0, 0);
        // Выводим уведомление
        toast.show();
        return true;
    }
}
```

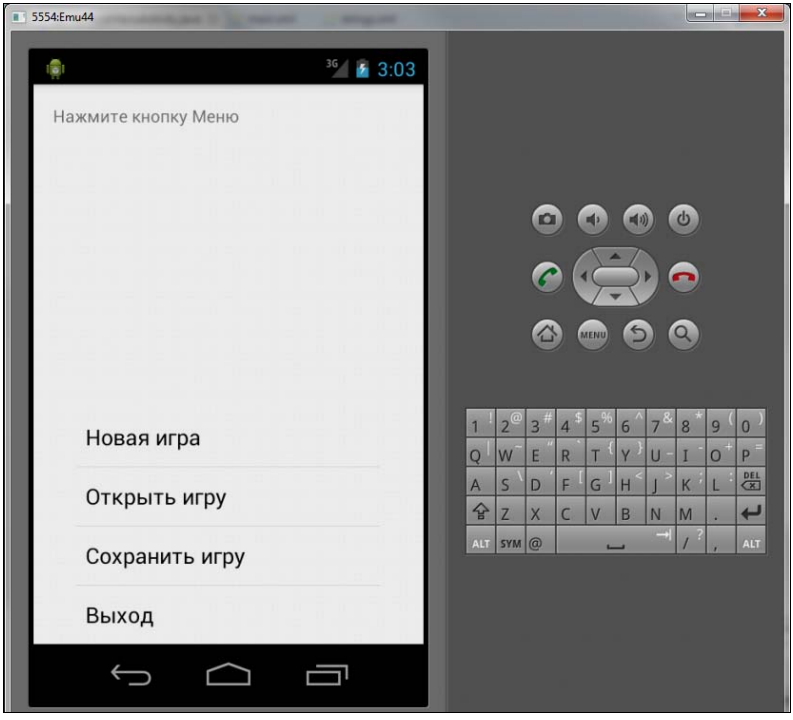


Рис. 6.10. Меню опций

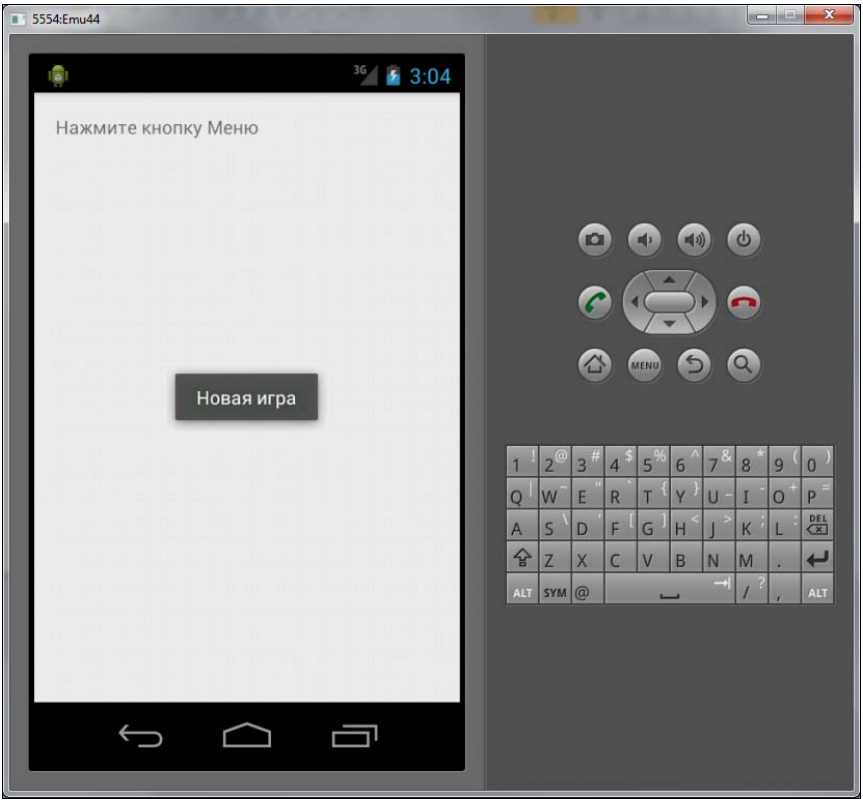


Рис. 6.11. Выбран первый пункт меню

6.3.2. Меню со значками

Меню со значками создается полностью аналогично меню опций. Только при вызове метода `add()` нужно еще вызвать метод `setIcon()`, подобно вызову метода `setAlphabeticShortcut()`:

```
menu.add(Menu.NONE, IDM_NEW, Menu.NONE, "Новая игра").setIcon(R.drawable.new);
```

При этом в каталоге res/drawable* должен присутствовать ресурс с именем `new` (попросту говоря, файл с именем new.png).

6.3.3. Расширенное меню

В старых версиях Android, если количество пунктов меню было более шести, автоматически появляется расширенное меню — вместо шестого пункта меню выводится пункт **More**, выбрав который пользователь получает доступ к оставшимся пунктам меню. В новых версиях такого нет. Вместо этого вы получите одно большое меню, которое можно листать, если оно не помещается на одном экране. Выглядит оно не очень презентабельно, поэтому или сокращайте количество пунктов меню, или используйте подменю (рис. 6.12).

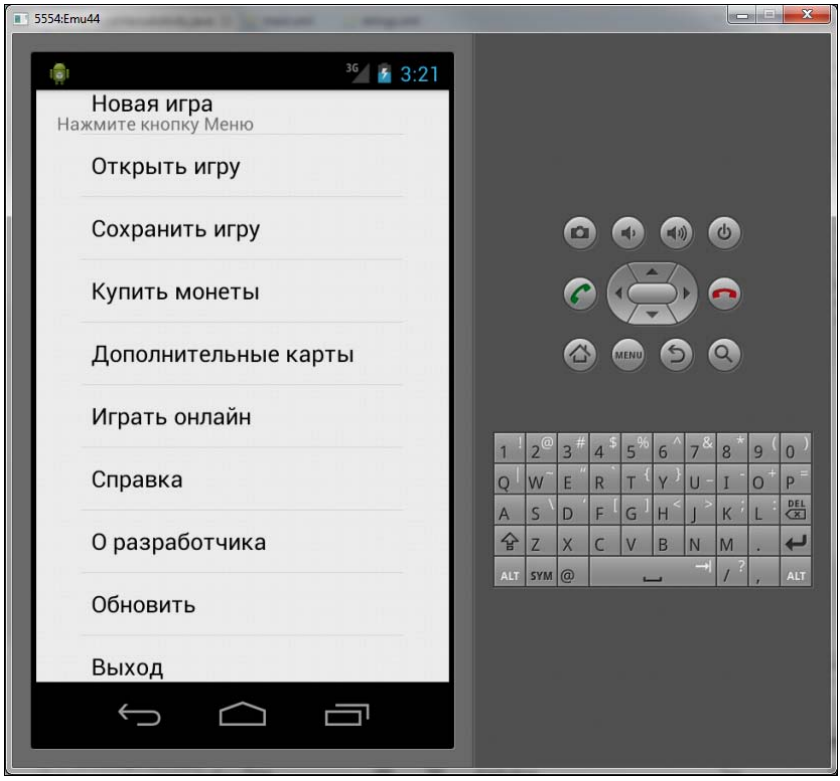


Рис. 6.12. Когда пункты меню не помещаются на экране...

6.3.4. Контекстное меню

Создать контекстное меню чуть сложнее, чем обычное. Ведь нужно не только создать меню, но и привязать его к определенному объекту интерфейса программы, — контекстное меню должно появляться не просто так, а при долгом нажатии на определенный объект.

Поскольку в нашей простой программе особых объектов нет (если не считать разметки и надписи), то меню будет появляться над разметкой (`LinearLayout`). А для этого понадобится изменить файл разметки — чтобы присвоить имя элементу `LinearLayout`. Но обо всем по порядку.

Первым делом нужно подключить дополнительные Java-пакеты. С появлением контекстного меню список импортируемых пакетов увеличился (сравните его с листингом 6.7):

```
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.Menu;
import android.view.MenuItem;
```

```
import android.view.ContextMenu;
import android.view.ContextMenu.ContextMenuInfo;
import android.widget.Toast;
import android.view.Gravity;
import android.widget.LinearLayout;
```

Помимо пакетов, непосредственно относящихся к контекстному меню, нам нужно добавить пакеты, относящиеся к линейной разметке (к ней мы будем привязывать наше меню), и пакет `android.view.View` — необходимый при создании меню.

Далее, как обычно, нужно определить идентификаторы нашего контекстного меню (чтобы не путать элементы контекстного меню с элементами меню опций, идентификаторы начинаются с двойки):

```
public static final int IDM_RESTORE = 201;
public static final int IDM_PAUSE = 202;
```

Далее создаем само контекстное меню:

```
@Override
    public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo
info) {
        super.onCreateContextMenu(menu, v, info);
        menu.add(Menu.NONE, IDM_PAUSE, Menu.NONE, "Пауза");
        menu.add(Menu.NONE, IDM_RESTORE, Menu.NONE, "Продолжить");
    }
```

Тип `View` требует пакет `android.view.View`, поэтому мы его и импортировали. Далее создаем обработчик выбора пункта меню:

```
@Override
    public boolean onContextItemSelected(MenuItem item) {
        CharSequence t;
        t = "";
        switch(item.getItemId()) {
        case IDM_PAUSE: t = "Пауза"; break;
        case IDM_RESTORE: t = "Продолжить"; break;
        default: super.onContextItemSelected(item);
        }
        Toast toast = Toast.makeText(this, t, Toast.LENGTH_LONG);
        toast.setGravity(Gravity.CENTER, 0, 0);
        toast.show();
        return true;
    }
```

Если сейчас запустить приложение, то наше меню ни при каких условиях не появится — ведь мы не привязали его к какому-либо виджету. Для привязки контекстного меню к нашей разметке используются операторы:

```
final LinearLayout game = (LinearLayout)findViewById(R.id.Main);
registerForContextMenu(game);
```

Чтобы эти операторы работали, нужно изменить файл разметки main.xml и присвоить нашей линейной разметке имя `main`. Файл main.xml для нашего приложения представлен в листинге 6.8.

Листинг 6.8. Файл разметки приложения с контекстным меню

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/Main"
    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_world"
    />
</LinearLayout>
```

Полный код приложения, содержащего меню опций и контекстное меню для линейной разметки, приведен в листинге 6.9.

Листинг 6.9. Полный код приложения

```
package com.sample.contextmenu;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.Menu;
import android.view.ContextMenu;
import android.view.MenuItem;
import android.widget.Toast;
import android.view.Gravity;
import android.widget.LinearLayout;
import android.view.ContextMenu.ContextMenuInfo;

public class MainActivity extends Activity {

       public static final int IDM_RESTORE = 201;
       public static final int IDM_PAUSE = 202;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
```

```
        final LinearLayout game = (LinearLayout)findViewById(R.id.Main);
        registerForContextMenu(game);

    }

 // Создание контекстного меню
    @Override
    public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo
info) {
        super.onCreateContextMenu(menu, v, info);
        menu.add(Menu.NONE, IDM_PAUSE, Menu.NONE, "Пауза");
        menu.add(Menu.NONE, IDM_RESTORE, Menu.NONE, "Возобновить");
    }

    // Обработка пунктов контекстного меню
    @Override
    public boolean onContextItemSelected(MenuItem item) {
        CharSequence t;
        t = "";
        switch(item.getItemId()) {
        case IDM_PAUSE: t = "Пауза"; break;
        case IDM_RESTORE: t = "Возобновить"; break;
        default: super.onContextItemSelected(item);
        }
        Toast toast = Toast.makeText(this, t, Toast.LENGTH_LONG);
        toast.setGravity(Gravity.CENTER, 0, 0);
        toast.show();
        return true;
    }
}
```

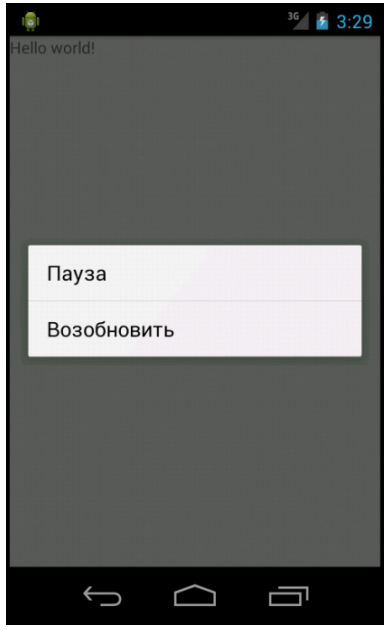


Рис. 6.13. Контекстное меню

Запустите приложение. Если вы запустили его в эмуляторе, для вызова контекстного меню нажмите левую кнопку мыши на основной рабочей области приложения, подождите 2 секунды и отпустите кнопку мыши. Результат приведен на рис. 6.13.

6.3.5. Подменю

Подменю можно добавить в любое другое меню, кроме самого подменю. Подменю полезно при создании очень сложных приложений, где пользователю доступно много функций.

Рассмотрим, как можно добавить подменю в меню опций. Для создания подменю используется метод `addSubMenu()`. Далее мы создадим два подменю в меню опций `File` и `Edit` — с соответствующими пунктами меню:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {

        SubMenu subFile = menu.addSubMenu("Файл");
        subFile.add(Menu.NONE, IDM_NEW, Menu.NONE, "Создать...");
        subFile.add(Menu.NONE, IDM_OPEN, Menu.NONE, "Открыть...");
        subFile.add(Menu.NONE, IDM_SAVE, Menu.NONE, "Сохранить");
        subFile.add(Menu.NONE, IDM_SAVEAS, Menu.NONE, "Сохранить как...");
        subFile.add(Menu.NONE, IDM_EXIT, Menu.NONE, "Выход");

        SubMenu subEdit = menu.addSubMenu("Правка");
        subEdit.add(Menu.NONE, IDM_COPY, Menu.NONE, "Копировать");
        subEdit.add(Menu.NONE, IDM_CUT, Menu.NONE, "Вырезать");
        subEdit.add(Menu.NONE, IDM_PASTE, Menu.NONE, "Вставить");

        return super.OnCreateOptionsMenu(menu);
}
```

Обработка команд подменю осуществляется аналогично обработке команд меню опций — разницы никакой нет.

Меню — очень важная часть приложения, и серьезное приложение вряд ли будет существовать без меню. Если у вас что-то не получилось, по следующему адресу вы найдете готовое приложение (которое мы создавали на протяжении *разд. 6.3*), демонстрирующее создание меню:

http://www.dkws.org.ua/mybooks/OptionsMenu.zip

Скачанный архив нужно распаковать в каталог, содержащий рабочее пространство Eclipse. В моем случае — это C:\Users\Den\workspace, у вас в 99 % случаев каталог будет таким же, за исключением собственно имени пользователя.

6.4. Диалоговое окно открытия файла

Android — одна из немногих операционных систем, в которой нет собственного диалогового окна открытия/сохранения файлов. Да, на фоне других операционных систем это кажется весьма странным и неудобным, как с точки зрения пользователя, так и программиста. Программисту приходится каждый раз выдумывать, как создать такое диалоговое окно, а пользователю — каждый раз привыкать к нему.

Именно поэтому мне не хотелось создавать еще одну версию диалогового окна открытия файлов, и я принялся искать уже готовые решения. Так и мне будет проще, и пользователям. Пересмотрев несколько уже готовых решений, я выбрал лучшее: **http://habrahabr.ru/post/203884/**. Можно также использовать и решение Android File Dialog (**https://code.google.com/p/android-file-dialog/**), но оно реализовано в виде отдельной деятельности. Если вас такой подход не смущает, то это тоже вполне приемлемый вариант.

Переписывать указанные решения (тем более, что они не мои) в книге я не стану. По приведенным ссылкам все достаточно подробно расписано (а в первом случае еще и на русском языке). Поэтому, надеюсь, вы разберетесь самостоятельно, а в следующей главе мы поговорим о работе с двумерной графикой в Android-приложениях.

ГЛАВА 7

Графика

В этой главе мы поговорим о создании статической графики путем рисования в объекте `View` из разметки или же рисования непосредственно на канве. Анимация будет рассмотрена в следующей части книги — в *главе 12*.

7.1. Класс *Drawable*

Для рисования на формах и изображениях используется графическая библиотека `android.graphics.drawable`. Класс `Drawable` определяет различные виды графики — например: `BitmapDrawable`, `ShapeDrawable`, `LayerDrawable` и др.

Существуют два способа определения и инициализации объектов `Drawable`. Первый заключается в использовании ресурсов из каталога res/drawable, а второй — в создании XML-файла со свойствами объекта `Drawable`.

В Android-приложения вы можете включать изображения следующих форматов:

- PNG — рекомендуемый формат;
- JPEG — поддерживаемый формат (тем не менее, лучше использовать PNG);
- BMP — поддерживается, но использовать не рекомендуется из-за большого размера файлов этого формата;
- GIF — формат поддерживается, но его использование настоятельно не рекомендуется. Палитра формата GIF включает всего 256 цветов, чего явно мало для экрана современного смартфона. К тому же разработчики Android рекомендуют использовать формат PNG.

> ***Особенности применения форматов***
>
> Конечно, для каждого формата есть свое применение. Формат PNG отлично подходит для изображений кнопок и других элементов графического интерфейса. Формат JPEG вы будете использовать для работы с фотографиями — от него никуда не денешься. Формат BMP — это изобретение Microsoft, вот пусть сами его и используют. Только у Microsoft есть столько дискового пространства, чтобы хранить изображения в формате BMP. Формат GIF поддерживает анимацию — это единственное его преимущество, но позже вы узнаете, что анимацию можно создать средствами Android. Поэтому вообще не вижу необходимости в этом формате.

Ресурсы изображений, находящиеся в каталоге res/drawable, во время компиляции программы оптимизируются утилитой aapt. Если вам нужно использовать растровые изображения без оптимизации, поместите их в каталог res/raw — при компиляции файлы из этого каталога не будут подвержены оптимизации.

Рассмотрим подробнее процесс добавления ресурса в проект. Предположим, что нам нужно добавить в него два файла: p1.jpg и p2.jpg. Подготовьте три варианта каждого файла: с высоким разрешением, со средним и с низким. Значение разрешения зависит от выбранной платформы и от самого мобильного устройства (см. *главу 1*).

Файлы с высоким разрешением нужно поместить в каталог res/drawable-hdpi, файлы с низким разрешением — в каталог res/drawable-ldpi, а со средним — в каталог res/drawable-mdpi. После этого вернитесь в окно Eclipse, нажмите клавишу <F5> и в области **Package Explorer** вы увидите добавленные файлы (рис. 7.1).

Теперь перейдите на вкладку **Images & Media** палитры компонентов и добавьте элемент `ImageView`. При его добавлении с помощью графического редактора раз-

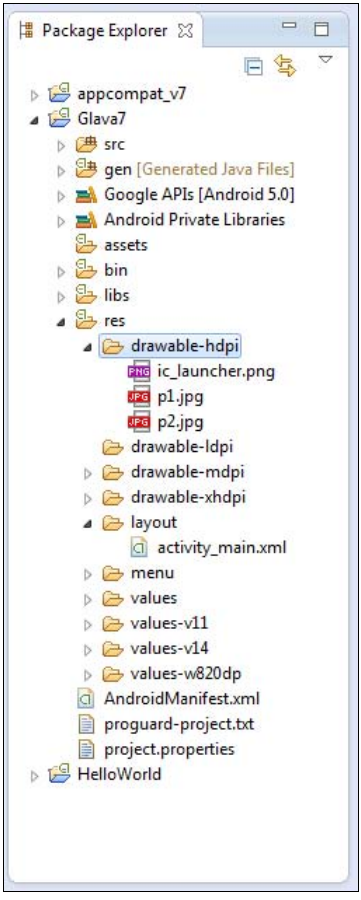


Рис. 7.1. Добавленные графические файлы

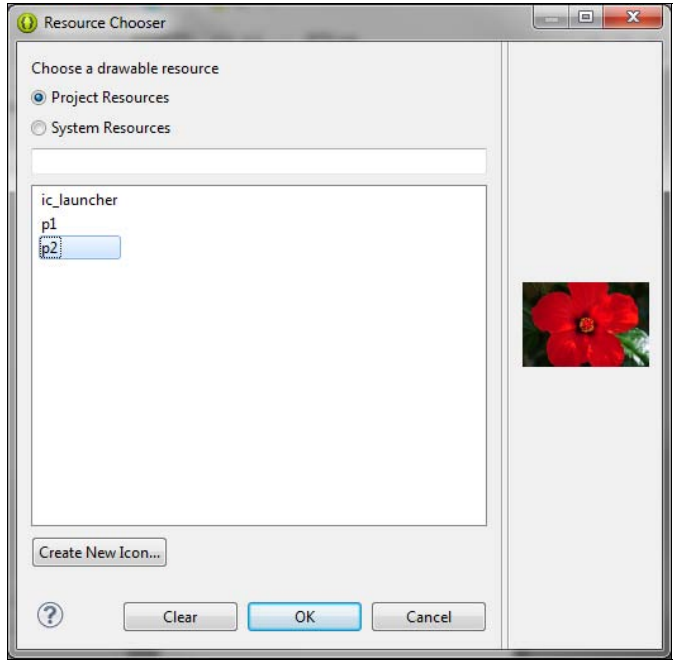


Рис. 7.2. Выбор ресурса

метки откроется окно, в котором нужно выбрать изображение для отображения в `ImageView` (рис. 7.2).

Выбранное изображение появится в редакторе разметки (рис. 7.3), а в файл activity_main.xml будет добавлен код, приведенный в листинге 7.1 (я только удалил отсюда код текстовой надписи, которая добавляется в проект по умолчанию).

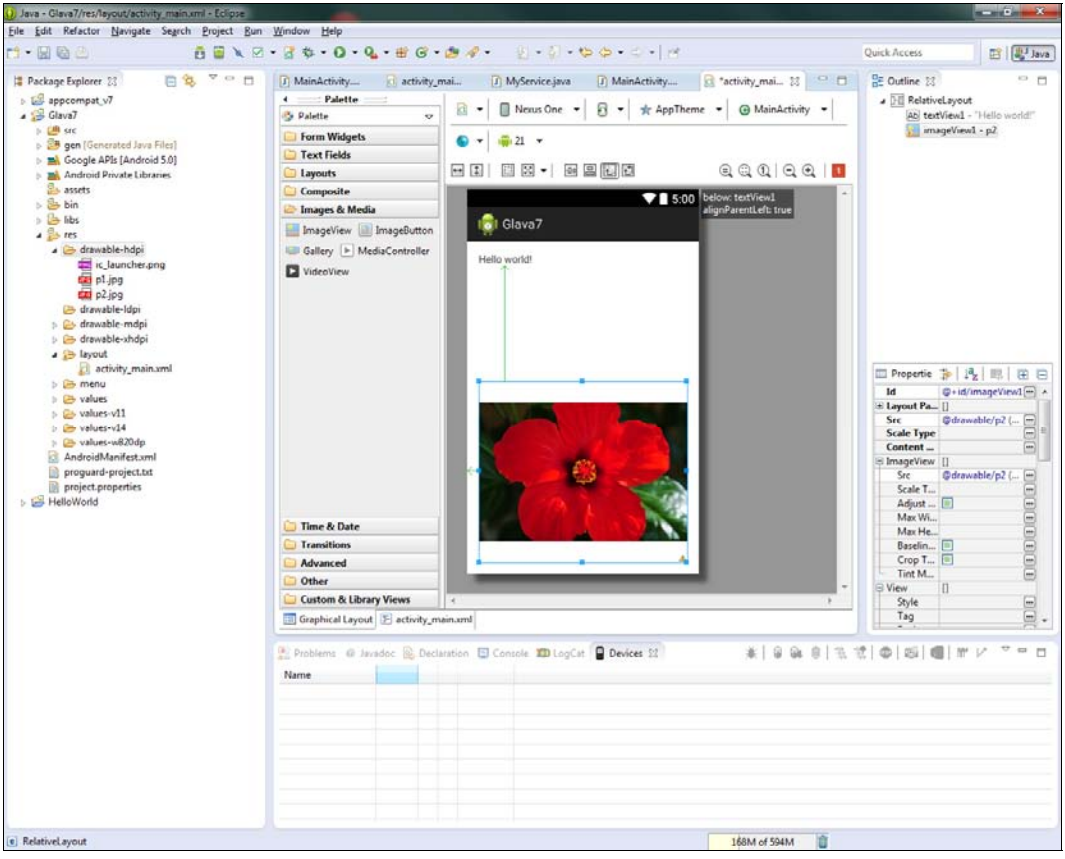


Рис. 7.3. Выбранное изображение

Листинг 7.1. Пример описания элемента `ImageView`

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.sample.glava7.MainActivity" >
```

```
    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/textView1"
        android:layout_marginTop="159dp"
        android:src="@drawable/p2" />

</RelativeLayout>
```

Обратите внимание, что в новом ADT-плагине появилось и новое предупреждение. В редакторе кода оно выглядит как желтый треугольник с восклицательным знаком напротив кода `ImageView`. Если подвести к нему указатель мыши, то вы увидите само предупреждение:

Missing contentDescription attribute on image

Это предупреждение не приведет к какой-либо ошибке при сборке проекта и его можно проигнорировать. Но если вам не нравятся какие-либо предупреждения и недостатки кода, тогда нужно предварительно добавить в файл res/values/strings.xml новую строку `desc`:

```
<string name="desc">Описание картинки</string>
```

Это и будет описание картинки, как вы уже догадались. Обязательно сохраните исправленный файл, нажав комбинацию клавиш <Ctrl>+<S>. Кроме того, в файл разметки необходимо добавить для `ImageView` атрибут `contentDescription`:

```
android:contentDescription="@string/desc"
```

и также сохранить файл разметки.

После этого предупреждение больше не будет появляться. Полный код `ImageView` выглядит так:

```
<ImageView
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
         android:contentDescription="@string/desc"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/textView1"
        android:layout_marginTop="159dp"
        android:src="@drawable/p2" />
```

Напрямую значение атрибута `contentDescription` указывать не следует, иначе получите еще одно предупреждение. Подробнее об этом атрибуте можно прочитать здесь:

http://developer.android.com/reference/android/view/View.html#attr_android:contentDescription.

Продолжим исследовать область `ImageView`. Имя ресурса задается параметром `android:src`.

В коде программы объект `Drawable` инициализируется так:

```
Resources R = mContext.getResources();
Drawable exitImage = R.getDrawable(R.drawable.exit_image);
```

Представим, что вы создали два разных объекта `Drawable` для одного и того же ресурса. Если вы потом измените свойство для одного из объектов `Drawable`, это же свойство будет автоматически изменено и для второго объекта `Drawable`, поскольку они используют один и тот же ресурс.

Узнать максимальную высоту и ширину `ImageView` можно методами:

```
int getMaxHeight()
int getMaxWidth()
```

Для загрузки изображения с определенного адреса (URI) теоретически может применяться метод `setImageURI()`, однако его использование иногда приводит к задержкам, связанным с передачей данных по сети. Применение этого метода ограничивается еще и тем, что интернет-адрес изображения должен указываться в формате, понятном операционной системе Android, а это не обычный URL, к которым мы привыкли при работе в Интернете. Другими словами, если вы укажете URL вроде **http://server.ru/image.jpg**, у вас ничего не выйдет. И вообще, метод `setImageURI()` работает некорректно. Вы думаете, что можно попробовать загрузить картинку так:

```
imageView.setImageURI(Uri.fromFile(file));
```

но у вас ничего не получается? Вы не один такой. Эта проблема описана здесь:

https://code.google.com/p/android/issues/detail?id=2733

К сожалению, она пока не решена, поэтому приходится «изощряться».

Итак, что нужно сделать, чтобы загрузить изображение с URI? Следует сначала получить объект `Drawable`, а потом передать его методу `getDrawable()`:

```
Uri imgUri=Uri.parse("file:///data/data/MYFOLDER/myimage.png");
d = Drawable.createFromPath(imgUri.getPath());
ImgView.getDrawable(d);
```

Вместо локального файла можно указать удаленный URL. Можно также сначала получить `Bitmap` из изображения, а потом конвертировать его в `Drawable`. В группе разработчиков Google приводится решение для этого в виде следующей удобной функции, которая возвращает `Bitmap` изображения, заданного строкой `url`:

```
private Bitmap getImageBitmap(String url) {
            Bitmap bm = null;
            try {
                URL aURL = new URL(url);
                URLConnection conn = aURL.openConnection();
                conn.connect();
```

```
            InputStream is = conn.getInputStream();
            BufferedInputStream bis = new BufferedInputStream(is);
            bm = BitmapFactory.decodeStream(bis);
            bis.close();
            is.close();
        } catch (IOException e) {
            Log.e(TAG, "Error getting bitmap", e);
        }
        return bm;
    }
```

Получив `Bitmap`, можно считать, что половина дела сделана. Осталось конвертировать его в `Drawable`. Это можно сделать так:

```
Bitmap b = getImageBitmap("http://server.ru/image.jpg");
Drawable d = new BitmapDrawable(getResources(),b);
```

Теперь у нас есть объект `d`, который можно передать методу `getDrawable()`.

Еще об одном способе загрузки изображения вы узнаете в *главе* 9, когда мы будем рассматривать работу с файлами.

7.2. Класс *TransitionDrawable*

В предыдущем разделе мы не зря добавили изображения в каталог ресурсов: сейчас мы рассмотрим класс `TransitionDrawable`, который используется для создания переходов между изображениями.

Для нашего приложения файл res/layout/main.xml уже готов, осталось создать только файл transition.xml в каталогах res/drawable*. В этом файле определяется эффект перехода между изображениями и описываются изображения, которые будут меняться во время перехода. Код файла transition.xml приведен в листинге 7.2.

Листинг 7.2. Файл transition.xml

```
<?xml version="1.0" encoding="utf-8"?>
<transition xmlns:android="http://schemas.android.com/apk/res/android">
<item android:drawable="@drawable/p1"></item>
<item android:drawable="@drawable/p2"></item>
</transition>
```

Организация самого перехода возлагается на Java-код. Полный код приложения приведен в листинге 7.3. Чтобы он был более понятным, объясню, как должно работать будущее приложение. При загрузке приложения в `ImageView1` загружается изображение, указанное в файле разметки. При щелчке на `ImageView1` загружается следующее изображение, указанное в файле transition.xml. Длительность плавного перехода между картинками задается методом `startTransition()` в миллисекундах, причем 1000 мс = 1 с.

Листинг 7.3. Переход между изображениями

```
package com.samples.hel31;

import android.app.Activity;
import android.os.Bundle;
import android.content.res.Resources;
import android.graphics.drawable.TransitionDrawable;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ImageView;

public class A31Activity extends Activity implements OnClickListener {
    /** Called when the activity is first created. */

       private ImageView image;
       private TransitionDrawable trans;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // находим ImageView1 в разметке
        image = (ImageView)findViewById(R.id.imageView1);
        // устанавливаем обработчик щелчка
        image.setOnClickListener(this);

        Resources res = this.getResources();
        // создаем объект класса TransisionDrawable — он и отвечает
        // за переход между изображениями
        trans = (TransitionDrawable)res.getDrawable(R.drawable.transition);

    }

    // реакция на щелчок на ImageView1
    public void onClick(View v) {
       image.setImageDrawable(trans);
       trans.startTransition(1000);
    }
}
```

Запустите приложение и щелкните на изображении. К сожалению, сделать снимок экрана нельзя — все равно ничего не будет понятно.

Вы можете использовать следующие методы класса `TransitionDrawable`:

- `setCrossFadeEnabled(boolean enabled)` — включает эффект `CrossFade`, который заключается в плавном затухании при переходе между первым и вторым изображениями;

- `boolean isCrossFadeEnabled()` — возвращает `true`, если эффект `CrossFade` включен;
- `resetTransition()` — сбрасывает переход, показывается только первый слой;
- `reverseTransition (int duration)` — переход в обратном порядке;
- `startTransition(int duration)` — запускает процесс перехода.

7.3. Класс *ShapeDrawable*

Класс `ShapeDrawable` используется для рисования двумерных фигур: линий, овалов, прямоугольников и т. п. Для создания графических примитивов служит следующий набор производных классов:

- `PathShape`;
- `RectShape`;
- `ArcShape`;
- `OvalShape`;
- `RoundRectShape`.

Класс `ShapeDrawable` является расширением класса `Drawable`, поэтому вы можете использовать его так же, как и класс `Drawable`. Например, установить прозрачность методом `setAlpha()` или цветовой фильтр методом `setColorFilter()`.

В конструкторе `ShapeDrawable` рисуемый по умолчанию примитив определяется как `RectShape`, т. е. как прямоугольник. Для объекта `ShapeDrawable` нужно установить границы — если этого не сделать, примитив не будет нарисован. Также необходимо установить цвет границы, иначе фигура будет черной, и вы ее не увидите.

Класс `RectShape` можно использовать и для рисования горизонтальных и вертикальных линий. Для этого задайте ширину (или высоту) прямоугольника в 1–2 пиксела:

```
// создаем фигуру (прямоугольник)
ShapeDrawable shape = new ShapeDrawable(new RectShape());
// устанавливаем высоту 1 пиксел, ширину — 200 пикселов
shape.setIntrinsicHeight(1);
shape.setIntrinsicWidth(200);
// устанавливаем цвет — красный
shape.getPaint().setColor(Color.RED);
```

Аналогично можно нарисовать овал:

```
// создаем фигуру (овал)
ShapeDrawable shape = new ShapeDrawable(new OvalShape());
// устанавливаем высоту 1 пиксел, ширину — 200 пикселов
shape.setIntrinsicHeight(1);
shape.setIntrinsicWidth(200);
```

```
// устанавливаем цвет — красный
shape.getPaint().setColor(Color.RED);
```

Нарисовать прямоугольник с закругленными сторонами (`RoundRectShape`) сложнее — тут используется следующий конструктор:

```
RoundRectShape(float[] outerRad, RectF inset, float[] innerRad)
```

Параметры конструктора трактуются так:

- `outerRad` — массив из восьми значений радиуса закругленных сторон внешнего прямоугольника. Первые два значения соответствуют верхнему левому углу, вторые — верхнему правому, третьи — нижнему правому, четвертые — нижнему левому;
- `inset` — объект класса `RectF`, он определяет расстояние от внутреннего прямоугольника до каждой стороны внешнего прямоугольника. Конструктор объекта `RectF` принимает четыре параметра: пары X и Y координат левого верхнего и правого нижнего углов внутреннего прямоугольника;
- `innerRad` — массив из восьми значений радиуса закруглений углов для внутреннего прямоугольника. Пары значений такие же, как в случае с `outerRad`, а если на внутреннем прямоугольнике закруглений не будет, передается `null`.

Например:

```
float[] oR = new float[] { 5, 5, 5, 5, 5, 5, 5, 5 };
float[] iR = new float[] { 5, 5, 5, 5, 5, 5, 5, 5 };
RectF rf = new RectF(8, 8, 8, 8);

// создаем фигуру (RoundRect)
ShapeDrawable shape = new ShapeDrawable(new RoundRectShape(oR, rf, iR));
// устанавливаем высоту и ширину
shape.setIntrinsicHeight(150);
shape.setIntrinsicWidth(200);
// устанавливаем цвет — красный
shape.getPaint().setColor(Color.RED);
```

Класс `Path` служит для создания множественного контура (пригодится для построения графов и других фигур сложной формы), состоящего из линий, квадратичных и кубических кривых. Для установки точек используется метод `moveTo()`, а для создания линии — метод `lineTo()`, параметры метода — это координаты конечной точки линии, а координаты начальной точки задаются методом `moveTo()`.

Пример использования класса `Path`:

```
Path p = new Path();
p.moveTo(60, 0);
p.lineTo(100,150);
p.lineTo(200,250);
```

```
ShapeDrawable shape = new ShapeDrawable(new PathSpahe(p, 100, 100));
shape.setIntrinsicHeight(150);
shape.setIntrinsicWidth(200);
shape.getPaint().setColor(Color.RED);
```

Класс `ArcShape()` служит для создания дуги, конструктор принимает два параметра типа `float`. Первый из них задает угол начала прорисовки дуги в градусах, а второй — угловой размер дуги, тоже в градусах:

```
ArcShape (float startAngle, float sweepAngle);
```

ГЛАВА 8

Мультимедиа

8.1. Поддерживаемые форматы

В *главе 7* была рассмотрена поддержка графических форматов, и вы узнали, что Android поддерживает форматы BMP, JPG, GIF и PNG. В этой главе мы поговорим о поддержке аудио- и видеоформатов (табл. 8.1).

Таблица 8.1. *Поддерживаемые форматы аудио/видео*

Тип мультимедиа	Сжатие	Поддерживаемые действия	Форматы
Аудио (музыка)	Без сжатия	Запись/воспроизведение	PCM
	Без сжатия	Воспроизведение	WAVE
	Без потерь	Не поддерживается	FLAC
	С мин. потерями	Воспроизведение	MP3, MP4, OGG, AAC, HE-AACv1, HE-AACv2
	Midi	Воспроизведение	MID, RXT, XMF, OTA, IMY, RTTTL
Аудио (речь)	С мин. потерями	Запись/воспроизведение	AMR-NB
	С мин. потерями	Воспроизведение	AMR-WB
Видео	Почти без потерь	Воспроизведение	H.264
	С мин. потерями	Запись/воспроизведение	H.263, MPEG-4 SP

Если в вашем приложении планируется запись аудио или видео, то в файл манифеста нужно добавить соответствующие разрешения:

```
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
<uses-permission android:name="android.permission.RECORD_VIDEO"/>
```

8.2. Работа с аудиозаписями

Существуют два различных способа записи и воспроизведения аудироликов:

- `MediaPlayer`/`MediaRecorder` — стандартный метод работы со звуком, который может храниться либо в файле, либо может быть представлен потоковыми данными. Для обработки аудио создается отдельный поток;
- `AudioTrack`/`AudioRecorder` — прямой (`raw`) доступ к аудиоданным. Полезен при манипуляциях со звуком в памяти, записи звука в буфер при воспроизведении или в любых других случаях, не требующих наличия потока или файла. Отдельный поток при обработке звука не создается.

8.2.1. Воспроизведение звука с помощью *MediaPlayer*

Рассмотрим, как можно воспроизвести звук с помощью `MediaPlayer`. Первым делом нужно создать экземпляр `MediaPlayer`:

```
MediaPlayer media = new MediaPlayer();
```

Далее надо указать источник звука — пусть это будет прямой (`raw`) ресурс:

```
media = MediaPlayer.create(this, R.raw.music1);
```

Можно загрузить звуковой файл:

```
media.setDataSource(путь);
media.prepare();
```

Например:

```
path = "/sdcard/Download/music/Track01.mp3";
mMediaPlayer = new MediaPlayer();
mMediaPlayer.setDataSource(path);
mMediaPlayer.prepare();
mMediaPlayer.start();
```

Запустить воспроизведение звука можно методом `start()`:

```
media.start();
```

Метод `pauseMP()` приостанавливает воспроизведение, продолжить воспроизведение можно методом `startMP()`:

```
media.pauseMP();
media.startMP();
```

Остановить воспроизведение и освободить ресурсы можно методами `stop()` и `release()`:

```
media.stop();                // останавливаем воспроизведение
media.release();             // освобождаем память
```

Для поддержки `MediaPlayer` нужно подключить следующий класс:

```
import android.media.MediaPlayer;
```

Получить дополнительную информацию о классе `MediaPlayer` можно по адресу:

http://developer.android.com/reference/android/media/MediaPlayer.html

8.2.2. Запись звука с помощью *MediaRecorder*

Прежде чем приступить к записи звука, нужно определиться с его источником (свойство `MediaRecorder.AudioSource`):

- `MIC` — встроенный микрофон;
- `VOICE_UPLINK` — исходящий голосовой поток при телефонном звонке (то, что вы говорите);
- `VOICE_DOWNLINK` — входящий голосовой поток при телефонном звонке (то, что говорит ваш собеседник);
- `VOICE_CALL` — запись телефонного звонка;
- `CAMCORDER` — микрофон, связанный с камерой, если таковой доступен;
- `VOICE_RECOGNITION` — микрофон, используемый для распознавания голоса, если таковой доступен.

После выбора источника звука нужно задать формат записываемого звука (свойство `MediaRecorder.OutputFormat`):

- `THREE_GPP` — формат 3GPP;
- `MPEG_4` — формат MPEG4;
- `AMR_NB` — формат AMR_NB, лучше всего подходит для речи.

Последовательность действий для записи звука будет следующей. Первым делом нужно создать экземпляр `MediaRecorder`:

```
MediaRecorder media = new MediaRecorder();
```

Затем указать источник звука — например, микрофон:

```
media.setAudioSource(MediaRecorder.AudioSource.MIC);
```

Третий шаг — установить результирующий формат и сжатие звука:

```
media.setOutputFormat(MediaRecorder.OutputFormat.AMR_NB);
media.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
```

Теперь устанавливаем путь к файлу, в котором будут сохранены аудиоданные:

```
media.setOutputFile(path);
```

Подготавливаем и запускаем запись:

```
media.prepare();
media.start();
```

Остановить запись можно методом `stop()`:

```
media.stop();
```

Для поддержки MediaRecorder надо подключить следующий пакет:

```
import android.media.MediaRecorder;
```

8.2.3. Использование *AudioRecord*/*AudioTrack*

Возможностей MediaPlayer/MediaRecorder в большинстве случаев должно хватить. Но для манипуляции прямыми (raw) аудиоданными, полученными, например, непосредственно из микрофона, следует использовать классы AudioRecord и AudioTrack. Первый класс служит для записи звука, второй — для воспроизведения.

Чтобы приложение могло записывать данные с помощью AudioRecord, нужно в файле манифеста объявить соответствующее разрешение:

```
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

Теперь рассмотрим процесс записи с помощью AudioRecord. Первым делом нужно создать экземпляр AudioRecord и установить источник звука:

```
short[] myAudio = new short[10000];
AudioRecord audioRecord = new AudioRecord(
                            MediaRecorder.AudioSource.MIC, 11025,
                            AudioFormat.CHANNEL_IN_MONO,
                            AudioFormat.ENCODING_PCM_16BIT, 10000);
```

Наша конфигурация подходит для записи голоса со встроенного микрофона в буфер myAudio. Мы записываем 11 025 образцов в секунду, а размер буфера — 10 000 образцов, следовательно, длительность записи — менее секунды. Для более длительной записи нужно увеличить размер буфера.

- Первый параметр конфигурации — это источник звука (MIC). Для его указания вы можете использовать свойства MediaRecorder.AudioSource.
- Второй параметр (11025) — это частота звука (в Гц). Такая частота подходит только для записи голоса, для CD-качества необходима частота 44100.
- Мы здесь записываем монозвук (CHANNEL_IN_MONO), для записи стерео измените третий параметр — его значение должно выглядеть так: AudioFormat.CHANNEL_IN_STEREO.
- Кодирование звука задается четвертым параметром. Здесь мы можем использовать либо 16-битное кодирование (что мы и делаем), либо 8-битное (AudioFormat.ENCODING_PCM_8BIT).
- Последний параметр — это размер в байтах буфера, в который будет производиться запись (10000). Другими словами — это общий размер выделенной памяти. Для правильного задания этого параметра лучше было бы воспользоваться методом getMinBufferSize(), но для простоты примера мы указали просто значение в байтах.

Далее нужно начать запись:

```
audioRecord.startRecording();
```

Поскольку у нас прямой доступ к микрофону, то просто указать файл, в который надо поместить прочитанный звук, нельзя. Нужно еще вручную считать этот звук с микрофона. Для этого используется метод `read()`:

```
audioRecord.read(myAudio, 0, 10000);
```

Остановить запись можно методом `stop()`:

```
audioRecord.stop();
```

Для непосредственной манипуляции со звуком вам нужно подключить следующие пакеты:

```
import android.media.AudioFormat;
import android.media.AudioManager;
import android.media.AudioRecord;
import android.media.AudioTrack;
import android.media.MediaRecorder;
```

Теперь поговорим о воспроизведении звука средствами `AudioTrack`. Конструктору объекта `AudioTrack` нужно передать:

- тип потока — `AudioManager.STREAM_MUSIC` (микрофон) или `STREAM_VOICE_CALL` (голосовой звонок). Другие варианты используются реже;
- частоту в герцах (Гц) — значения такие же, как и для записи звука;
- конфигурацию канала — `AudioFormat.CHANNEL_OUT_STEREO` или `AudioFormat.CHANNEL_OUT_MONO`. Можно также использовать значение `CHANNEL_OUT_5POINT1` для звука 5.1;
- тип кодирования звука — значения такие же, как и для записи;
- размер буфера в байтах;
- режим буфера — `AudioTrack.MODE_STATIC` (подходит для небольших звуков, которые полностью помещаются в памяти) или `AudioTrack.MODE_STREAM` (для потокового звука).

Пример инициализации объекта `AudioTrack`:

```
AudioTrack audioTrack = new AudioTrack(
                            AudioManager.STREAM_MUSIC, 11025,
                            AudioFormat.CHANNEL_OUT_MONO,
                            AudioFormat.ENCODING_PCM_16BIT, 4096,
                            AudioTrack.MODE_STREAM);
```

Далее нужно начать воспроизведение методом `play()`. Поскольку мы все делаем вручную, то должны также вручную записать звуковые данные на устройство воспроизведения. Это делается методом `write()`. Первый параметр этого метода — буфер со звуковыми данными, а третий параметр — размер буфера. Второй параметр — смещение относительно начала буфера. Если смещение равно 0, то воспроизведение будет начато с начала буфера. Остановить воспроизведение можно методом `stop()`:

```
audioTrack.play();
audioTrack.write(myAudio, 0, 10000);
audioTrack.stop();
```

А теперь самое интересное. Если в случаях с `MediaPlayer` и `MediaRecorder` можно было разобраться самому, то классы `AudioRecorder` и `AudioTrack` требуют наглядного примера.

Сейчас мы напишем небольшое приложение, главное окно которого будет содержать две кнопки: **Запись** и **Воспроизведение**. При нажатии кнопки **Запись** будет инициирована запись с микрофона на протяжении 5 секунд. При нажатии кнопки **Воспроизведение** будет воспроизведен записанный звук.

Файл разметки нашего приложения представлен в листинге 8.1, код приложения — в листинге 8.2.

Листинг 8.1. Файл разметки приложения AudioDemo

```
<?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:id="@+id/status"
       android:text="Ready" android:textSize="20sp"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content" />
<Button android:id="@+id/record"
       android:text="Запись"
       android:textSize="20sp" android:layout_width="wrap_content"
       android:layout_height="wrap_content" />
<Button android:id="@+id/play"
       android:text="Воспроизведение" android:textSize="20sp"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content" />
</LinearLayout>
```

Листинг 8.2. Код приложения AudioDemo

```
package com.samples.audio_demo;

import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
// Подключаем необходимые пакеты
import android.media.AudioFormat;
```

```
import android.media.AudioManager;
import android.media.AudioRecord;
import android.media.AudioTrack;
import android.media.MediaRecorder;

public class AudioDemo extends Activity implements Runnable {

        private TextView statusText;

        public void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // Находим надпись и кнопки
        statusText = (TextView) findViewById(R.id.status);
        Button actionButton = (Button) findViewById(R.id.record);

        // Устанавливаем обработчик для кнопки записи
        actionButton.setOnClickListener(new View.OnClickListener() {
                public void onClick(View view) {
                        record_thread();
                }
        });
        // Находим кнопку Play и устанавливаем обработчик для нее
        Button replayButton = (Button) findViewById(R.id.play);
        replayButton.setOnClickListener(new View.OnClickListener() {
                public void onClick(View view) {
                        Thread thread = new Thread(AudioDemo.this);
                        thread.start();
                }
        });
}

        String text_string;
        final Handler mHandler = new Handler();
        final Runnable mUpdateResults = new Runnable() {
        public void run() {
                updateResultsInUi(text_string);
                }
        };

private void updateResultsInUi(String update_txt) {
        statusText.setText(update_txt);
}

// Поток записи
private void record_thread() {
        Thread thread = new Thread(new Runnable() {
        public void run() {
```

```
            text_string = "Запись...";
            mHandler.post(mUpdateResults);
            record();
            text_string = "Готово!";
            mHandler.post(mUpdateResults);
        }
});
        thread.start();
}

        // Параметры записи звука
        private int audioEncoding = AudioFormat.ENCODING_PCM_16BIT;
        int frequency = 11025;       //Гц
        // Размер буфера записи
        int bufferSize = 50*AudioTrack.getMinBufferSize(frequency,
        AudioFormat.CHANNEL_OUT_MONO, audioEncoding);

        // Создаем объект AudioRecord для записи звука
        public AudioRecord audioRecord = new AudioRecord(
                    MediaRecorder.AudioSource.MIC,
                    frequency, AudioFormat.CHANNEL_IN_MONO,
                    audioEncoding, bufferSize);

        // Создаем объект AudioTrack с теми же параметрами
        public AudioTrack audioTrack = new AudioTrack(
                    AudioManager.STREAM_MUSIC, frequency,
                    AudioFormat.CHANNEL_OUT_MONO,
                    audioEncoding, 4096,
                    AudioTrack.MODE_STREAM);

        // Буфер для звука
        short[] buffer = new short[bufferSize];

        // Функция записи звука
        public void record() {
               try {
               // Начинаем запись
                      audioRecord.startRecording();
                      // Читаем звук в буфер
                      audioRecord.read(buffer, 0, bufferSize);
                      // Останов записи
                      audioRecord.stop();
               } catch (Throwable t) {
                      Log.e("AudioDemo","Ошибка 1");
               }
        }
```

```
                // Воспроизведение звука
                public void run() {
                int i=0;
                while(i<bufferSize) {
                        audioTrack.write(buffer, i++, 1);
                }
                return;
        }

// Действие при паузе
@Override
protected void onPause() {
        if(audioTrack!=null) {
        if(audioTrack.getPlayState()==AudioTrack.PLAYSTATE_PLAYING) {
                audioTrack.pause();
        }
}
        super.onPause();
}
}
```

8.3. Работаем с видеозаписями

Для записи и воспроизведения видео используются уже знакомые нам классы `MediaRecorder` и `MediaPlayer`. Для работы с видео нужно добавить в файл манифеста следующую строку:

```
<uses-permission android:name="android.permission.RECORD_VIDEO" />
```

Источником видео (свойство `MediaRecorder.VideoSource`) может быть только встроенная камера — `MediaRecorder.VideoSource.CAMERA`.

Формат видео задается свойством `OutputFormat`:

- `THREE_GPP` — формат 3GPP, в последнее время активно используется для записи мобильного видео;
- `MPEG_4` — популярный формат MPEG4.

Кодек видео можно выбрать с помощью свойства `MediaRecorder.VideoEncoder`:

- `H264` — кодек H.264;
- `H263` — кодек H.263;
- `MPEG_4_SP` — кодек MPEG4.

Рассмотрим последовательность действий по записи видео. Как обычно, сначала мы создаем объект класса `MediaRecorder`:

```
MediaRecorder VideoRecorder = new MediaRecorder();
```

Указываем источник видео (встроенная камера):

```
VideoRecorder.setVideoSource(MediaRecorder.VideoSource.CAMERA);
```

Третий шаг — установка формата файла и кодека:

```
VideoRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
VideoRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.H263);
```

Путь к файлу, в котором будет сохранено видео, задается так:

```
VideoRecorder.setOutputFile(путь);
```

Осталось только начать запись:

```
VideoRecorder.prepare();
VideoRecorder.start();
```

Остановить запись можно методом `stop()`.

Также важен метод `setDisplay()`, позволяющий указать проигрывателю, куда выводить изображение. В разметке приложения нужно разместить компонент `SurfaceView` (он находится на вкладке **Advanced**), вызвать его метод `getHolder()` и полученный объект передать в метод `setDisplay()`.

Чтобы узнать размер проигрываемого изображения, можно использовать методы `getVideoHeight()` и `getVideoWidth()`.

Рассмотрим воспроизведение видео:

```
// Создаем новый объект
MediaPlayer MP = new MediaPlayer();
// Загружаем видео из ресурса
MP = MediaPlayer.create(this, R.raw.my_video1);
// Устанавливаем поверхность для воспроизведения видео
mHolder = mSurfaceView.getHolder();
MP.setDisplay(mHolder);
// Аналогично можно загрузить видео из файла с помощью
// следующих двух операторов
// MP.setDataSource(path);
// MP.prepare();
// Запускаем воспроизведение видео
MP.start();
// Останавливаем воспроизведение и освобождаем ресурсы:
MP.stop();
MP.release();
```

ГЛАВА 9

Методы хранения данных

9.1. Три метода доступа к данным

В Android-приложениях вы можете использовать три метода доступа к данным.

- Непосредственный доступ к файлам на SD-карте — обращение к файлам, находящимся на SD-карте мобильного устройства. Например, вы можете создать записную книжку, которая при запуске считывает текстовый файл с заметками пользователя, а при выходе — сохраняет отредактированную пользователем версию заметок. Аналогично можно загружать с SD-карты и изображения. До сих пор мы загружали изображения преимущественно из ресурсов приложения, но такой способ подходит разве что для загрузки изображений, которые используются для организации интерфейса пользователя (изображения кнопок и т. д.). Тяжеловесные изображения не следует добавлять как ресурс — этим вы существенно увеличите размер приложения.
- Предпочтения (Preferences) — не всем приложениям нужен доступ к SD-карте. Например, вы разработали игру и желаете хранить где-то лучший результат пользователя. Использовать для этой цели файлы или тем более базы данных — это все равно, что палить из пушки по воробьям, т. е. цель не оправдывает средства. Для хранения настроек приложения Android предоставляет отдельный механизм — предпочтения.
- Базы данных — сложные приложения могут использовать базы данных для хранения пользовательских данных. База данных предоставляет уже готовые средства по организации и поиску данных, и вам не нужно изобретать велосипед заново.

Некоторые приложения могут комбинировать несколько способов доступа к данным. Пусть вы разрабатываете программу для организации персональных фотографий пользователя — фотоальбом. При этом сами фотографии будут храниться на SD-карте. Но ваша программа позволяет для каждой фотографии добавить небольшую строку — комментарий (где и когда была сделана фотография, кто на ней изображен и т. п.). И такие комментарии к фотографиям целесообразно хранить в базе данных. А вот настройки приложения — например, номер последнего комментария, лучше всего хранить в предпочтениях.

В этой главе мы рассмотрим первые два способа доступа к данным. Третий способ будет описан в *главе 13*, поскольку он требует введения в базы данных и изучения основ языка SQL.

9.2. Чтение и запись файлов

9.2.1. Текстовые файлы

В пакете java.io.* находятся классы, необходимые для работы с файлами: `FileInputStream`, `FileOutputStream`, `InputStream`, `OutputStream` и др.

Работа с файлами в Android осуществляется так же, как и в случае с обычным Java-приложением. Если вы знакомы с Java, то весь приведенный здесь материал вам должен быть также знаком.

Для чтения файла служат классы `FileInputStream` и `InputStreamReader`. Первый применяется для открытия файла, второй — для организации чтения файла:

```
FileInputStream fis = new FileInputStream("test.txt");
InputStreamReader in = new InputStreamReader(fis, "UTF-8");
```

Класс `InputStreamReader` удобно использовать, если нужно определить кодировку файла. В противном случае (если с кодировкой все нормально) удобнее задействовать класс `FileReader`.

В листинге 9.1 приводится функция, возвращающая содержимое текстового файла. Функции нужно передать объект класса `File`. Создать объект класса `File` можно так:

```
File aFile = new File("file.txt");
```

Можно передать функцию `getContents()` так, чтобы она принимала имя файла:

```
static public String getContents(String aFile) {
```

Такая переделка становится возможной, поскольку конструктор класса `FileReader` принимает как строку, так и объект класса `File`:

```
FileReader(String filePath)
FileReader(File fileObj)
```

Листинг 9.1. Функция `getContents()`

```
static public String getContents(File aFile) {
    // Буфер строк
    // Чтобы сделать код совместимым со старой версией JDK,
    // просто замените StringBuilder на StringBuffer
    StringBuilder contents = new StringBuilder();

    try {
      // Читаем по одной строке за раз
      // FileReader подразумевает, что с кодировкой все нормально
```

```
      BufferedReader input =  new BufferedReader(new FileReader(aFile));
      try {
        String line = null; // Переменная для чтения строки

        while (( line = input.readLine()) != null){
          // Прочитанную строку добавляем в буфер contents
          // После каждой строки добавляется разделитель строк
          // (последовательность символов, разделяющая строки)
          contents.append(line);
          contents.append(System.getProperty("line.separator"));
        }
      }
      finally {
        input.close();
      }
    }
    catch (IOException ex){
      ex.printStackTrace();
    }
    // Возвращаем буфер строк как одну большую строку
    return contents.toString();
  }
```

Теперь рассмотрим `setContents()` — функцию записи в файл (листинг 9.2). Ей нужно передать два параметра: файл и строку, которую нужно в него записать. Переделать функцию для работы только с именем файла не получится, поскольку мы будем использовать методы класса `File` для определения существования файла и возможности записи в него.

Листинг 9.2. Функция записи в файл

```
static public void setContents(File aFile, String aContents)
                                throws FileNotFoundException, IOException {
    if (aFile == null) {
      throw new IllegalArgumentException("File should not be null.");
    }
    if (!aFile.exists()) {
      throw new FileNotFoundException ("File does not exist: " + aFile);
    }
    if (!aFile.isFile()) {
      throw new IllegalArgumentException("Should not be a directory: " + aFile);
    }
    if (!aFile.canWrite()) {
      throw new IllegalArgumentException("File cannot be written: " + aFile);
    }

    // Используем буферизацию
    Writer output = new BufferedWriter(new FileWriter(aFile));
```

```
    try {
      // Метод write() записывает строку в файл
      output.write( aContents );
    }
    finally {
      output.close();
    }
  }
```

Рассмотрим пример использования этих двух функций:

```
// Определяем файл
File aFile = new File("file.txt");
// Получаем содержимое файла
String old_content = getContents(aFile);
// Определяем новое содержимое файла
String new_content = "New content";
// Записываем в файл
setContents(aFile, new_content);
```

9.2.2. Файлы изображений

Отдельного разговора заслуживает чтение и запись файлов изображений. Сначала рассмотрим чтение графических файлов. Добавьте в файл разметки приложения элемент `ImageView`:

```
<ImageView id="imageToShow">
```

Java-код для отображения графического файла будет следующим:

```
// Задаем имя файла на SD-карте
String fname = "/sdcard/myImages/my_photo.jpg";
// Создаем объект класса File
File imageFile = new File(fname);
// Если файл существует...
if(imageFile.exists()){
       // Загружаем изображение
       Bitmap bmp = BitmapFactory.decodeFile(fname);
       // Находим область для отображения картинки
       ImageView myImage = (ImageView) findViewById(R.id.imageToShow);
       // Устанавливаем картинку
       myImage.setImageBitmap(myBitmap);
}
```

Последовательность действий для сохранения изображения будет следующей:

```
// Это наше изображение, например, полученное с камеры
Bitmap takenPicture;
// Устанавливаем имя файла, в который нужно сохранить изображение
FileOutputStream out = openFileOutput("mypic.png",
Context.MODE_WORLD_WRITEABLE);
```

```
// Устанавливаем метод сжатия — в нашем случае PNG
takenPicture.compress(CompressFormat.PNG, 100, out);
// Сбрасываем содержимое буферов, чтобы убедиться, что
// файл физически записан на SD-карту
out.flush();
// Закрываем поток
out.close();
```

9.3. Работа с URI

Не всегда приходится работать с локальными файлами. Иногда файлы нужно загружать из Интернета или из других источников. А для этого нужно задать URI файла. Для работы с URI в Android используется класс `Uri`, который мы сейчас и рассмотрим.

Собственно URI состоит из трех частей: схемы, зависимой от схемы части и пути. Возьмем для примера вот такой URI: `http://dkws.org.ua/pages/index.php`. Схемой здесь является `http`. Зависимая от схемы часть (она отличается для разных `Uri`) здесь: `dkws.org.ua/pages/index.php`. Последняя часть: `/pages` — это и есть путь (обратите внимание, что `index.php` — это не путь).

У протокола HTTP (как и у протокола FTP) есть свои параметры, но сейчас мы не будем их рассматривать.

У класса `Uri` самый важный метод — `parse()`. Именно он занимается разбором URI, переданного программистом или пользователем.

С интернет-адресами все ясно. Далее мы рассмотрим несколько нестандартных URI, которые можно использовать только в Android.

Представим, что нам нужно загрузить видеоролик (формат 3GP) в компонент `VideoView`. С помощью URI можно указать программе, откуда загрузить ролик.

Представим, что ролик находится в ресурсах программы. Тогда `Uri` будет следующим:

```
Uri cURI;
cURI = Uri.parse("android.resource://your.app.package/" + R.raw.video);
```

Как видите, в качестве URI может выступать не только интернет-адрес. В нашем случае схема здесь: `android.resource`.

А теперь представим, что видео находится в каталоге video на SD-карте:

```
Uri cURI;
cURI = Uri.parse("file:///sdcard/video/video.3gp");
```

Далее нужно использовать метод `setVideoURI(URI uri)` компонента `VideoView` для загрузки видео. Обратите внимание, что этому методу нельзя напрямую передать URI. Ему нужно передавать объект класса `Uri`, который мы создали ранее! Такое поведение свойственно для многих других компонентов/классов в Android. Непосредственно URI практически никогда не передаются. Нужно выполнить разбор `Uri`

с помощью метода `parse()`, а затем передать полученный объект для последующей обработки.

Итак, вот код загрузки видео:

```
Uri cURI;
cURI = Uri.parse("file:///sdcard/video/video.3gp");
VideoView videoView;
videoView.setURI(cURI);
```

Но и это еще не все! Метод `parse()` можно использовать для парсинга географических координат, номеров телефонов и даже контактов из адресной книги:

```
Uri uri = Uri.parse("geo:45.357183,67.65021");
Uri uri = Uri.parse("tel:3334455");
Uri uri = Uri.parse("content://contacts/people/Den");
```

Подробнее о классе `Uri` можно прочитать по адресу:

http://developer.android.com/reference/android/net/Uri.html

9.4. Предпочтения: сохранение настроек приложения

Предпочтения — это специальный механизм чтения и записи пар «ключ-значение» для примитивных типов данных. Предпочтения обычно используются для сохранения и восстановления параметров приложения — например, имени графической темы (скина), текста приветствия, названия шрифта и т. д.

Информация сохраняется в XML-файле на Android-устройстве. Например, если приложение com.samples.hello создает разделяемое предпочтение, Android создает новый XML-файл в каталоге /data/data/com.samples.hello/shared_prefs.

Получить доступ к предпочтениям можно с помощью метода `Activity.getPreferences()`. Позже мы рассмотрим этот метод, как и метод `getSharedPreferences()`.

Существуют разные типы доступа к предпочтениям:

- `MODE_PRIVATE` — предпочтение доступно только создавшему его приложению;
- `MODE_WORLD_READABLE` — предпочтение доступно всем приложениям для чтения;
- `MODE_WORLD_WRITEABLE` — предпочтение доступно всем приложениям для записи.

Для хранения настроек приложения используются предпочтения четырех типов:

- `CheckBoxPreference` — позволяет хранить состояние независимого переключателя;
- `EditTextPreference` — самое универсальное предпочтение, позволяет хранить текстовую строку;
- `ListPreference` — предпочтение списка;
- `RingtonePreference` — предпочтение рингтона.

Чаще всего используется предпочтение `EditTextPreference`, его мы сейчас и рассмотрим. Начнем с сохранения предпочтений. Для этого применяется следующий Java-код:

```
// Создаем объект SharedPreferences
// Определяем название и режим предпочтений
SharedPreferences p = getSharedPreferences("myAppPrefs", MODE_PRIVATE);
// Начинаем редактирование предпочтений
Editor prefEditor = p.edit();
// Сохраняем две пары — username = den, password = 123456
prefEditor.putString("username", "den");
prefEditor.putString("password", "123456");
prefEditor.commit();
```

Прочитать предпочтения можно методом `getString()`:

```
SharedPreferences p = getSharedPreferences("myAppPrefs", MODE_PRIVATE);
String username = p.getString("username", "");
String password = p.getString("password", "");
```

Тут как бы все понятно. Вот только, сами понимаете — это еще не все. Впереди огромная часть работы. Во-первых, вам нужно создать для вашего приложения *деятельность* — окно настроек. Пусть в приложении должно быть меню, а в нем — пункт **Settings**, при выборе которого и будет отображаться это окно. Во-вторых, вам надо обеспечить загрузку параметров при запуске приложения и сохранение параметров при закрытии окна настроек. Согласитесь, довольно большой «кусок» работы.

Однако должен вас обрадовать. Система Android может выполнить всю рутинную работу за вас. Она создаст деятельность с настройками (которая будет выглядеть унифицированно с аналогичными окнами других приложений), она организует загрузку и сохранение настроек приложения. Вам же останется обеспечить только запуск этой деятельности (например, через пункт меню или кнопку **Settings**) и чтение настроек (ранее было показано, как это сделать), когда это вам необходимо. Ведь читать настройки нужно не только для их загрузки в окно настроек, но и для организации работы программы. Так, в нашем случае перед подключением к серверу потребуются имя пользователя и пароль. Поэтому сначала их надо прочитать, а уже потом использовать. В любом случае система Android возьмет на себя все, что касается обработки окна настроек и сохранения самих настроек, а от вас потребуется минимум усилий.

Теперь рассмотрим практический пример. Создайте файл res/xml/preferences.xml, описывающий предпочтения (листинг 9.3).

Листинг 9.3. Файл res/xml/preferences.xml

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
```

```
<EditTextPreference android:title="User Name"
       android:key="username"
       android:summary="Your username">
</EditTextPreference>
<EditTextPreference android:title="Password"
       android:password="true"
       android:key="password"
       android:summary="Your password">
</EditTextPreference>
</PreferenceScreen>
```

В файле описания предпочтений мы объявили два предпочтения с именами `username` и `password`, оба типа `EditTextPreference`. Для пароля мы задали свойство `password` = `true`, что обеспечит сокрытие символов пароля в окне приложения.

Далее считаем, что наш проект называется com.samples.prefs. Поэтому в каталог src/com/samples/prefs нужно добавить файл MyPreferences.java (листинг 9.4). Этот файл обеспечивает загрузку предпочтений из XML-файла.

Листинг 9.4. Файл src/com/samples/prefs/MyPreferences.java

```
package com.samples.prefs;
import android.os.Bundle;
import android.preference.PreferenceActivity;

public class MyPreferences extends PreferenceActivity {
       @Override
       public void onCreate(Bundle savedInstanceState) {
              super.onCreate(savedInstanceState);
              addPreferencesFromResource(R.xml.preferences);
       }
}
```

Однако вы не можете просто так добавить файл MyPreferences.java в каталог src/com/samples/prefs — для этого вы должны использовать интерфейс Eclipse, иначе среда не увидит «чужой» файл.

Чтобы добавить новый класс, щелкните правой кнопкой мыши на файле com.samples.prefs (как показано на рис. 9.1) и выберите команду **New | Class**. Затем в появившемся окне введите имя нового класса: `MyPreferences` (рис. 9.2). После нажатия кнопки **Finish** вы увидите окно текстового редактора, в котором появился код-заглушка для нового класса. Замените этот код на код из листинга 9.4.

Теперь перейдем к редактированию основного файла кода приложения. Основная деятельность (основное окно) должно запустить деятельность `PreferenceActivity` в случае необходимости (например, при выборе пункта **Settings** в меню приложения). Для простоты примера в листинге 9.5 мы запускаем деятельность `PreferenceActivity` при запуске основной деятельности.

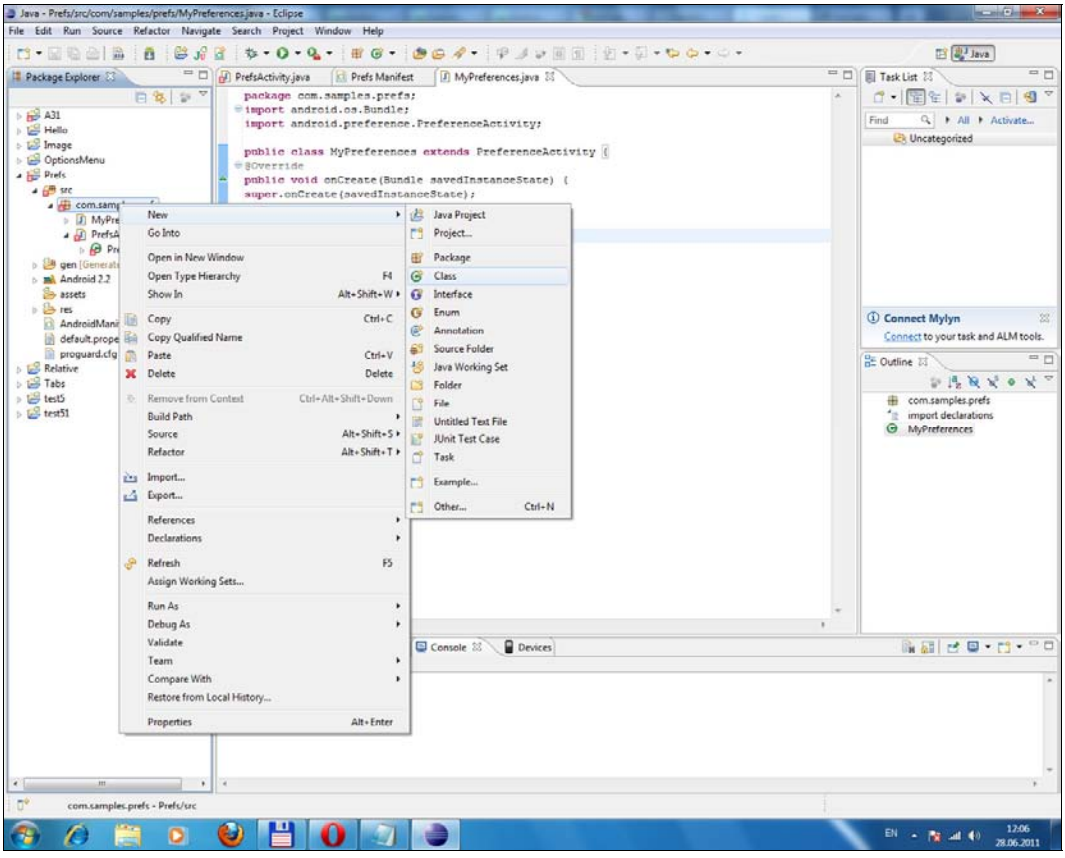


Рис. 9.1. Команда добавления нового класса

Листинг 9.5. Запуск деятельности предпочтений

```
package com.samples.prefs;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;

public class Prefs extends Activity {
       /** Called when the activity is first created. */
       @Override
       public void onCreate(Bundle savedInstanceState) {
              super.onCreate(savedInstanceState);
              setContentView(R.layout.main);
              // Запускаем окно с настройками
              Intent i = new Intent(this, MyPreferences.class);
              startActivity(i);
       }
}
```

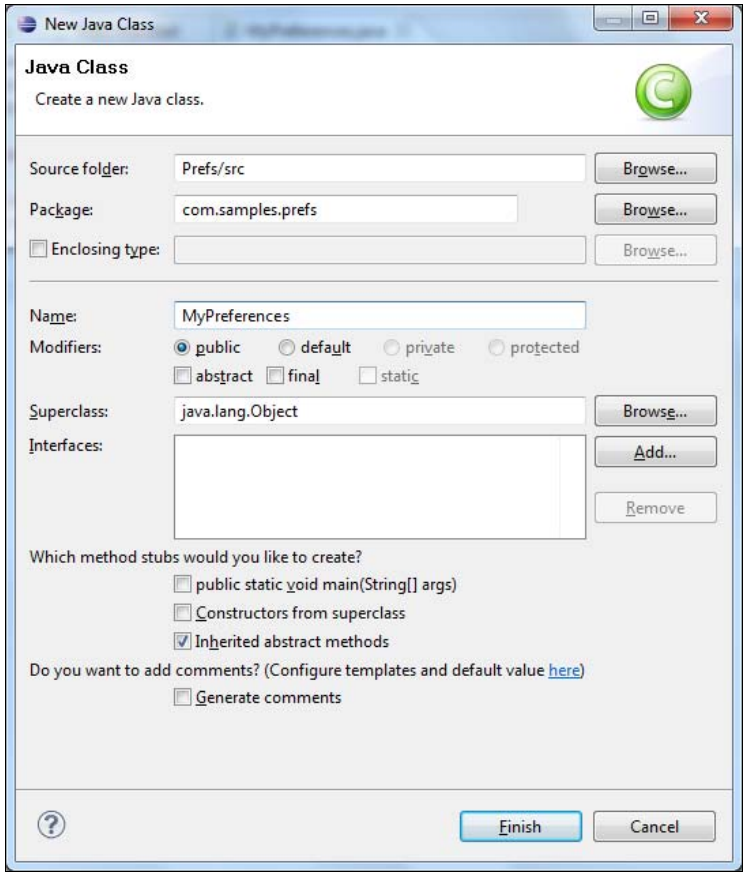


Рис. 9.2. Окно создания нового класса

Практически все. Надо лишь отредактировать файл манифеста, добавив в него сведения об активности предпочтений (листинг 9.6).

Листинг 9.6. Файл AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
       package="com.samples.prefs"
       android:versionCode="1"
       android:versionName="1.0">
<application android:icon="@drawable/icon" android:label="@string/app_name">
<activity android:name=".Prefs"
       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=".MyPreferences" />
</application>
<uses-sdk android:minSdkVersion="8" />
</manifest>
```

Осталось только все протестировать. Запустите приложение, и вы увидите окно установки настроек (рис. 9.3). Выберите первый параметр — **User Name**. Появится диалоговое окно ввода строки — укажите любое имя пользователя (рис. 9.4).

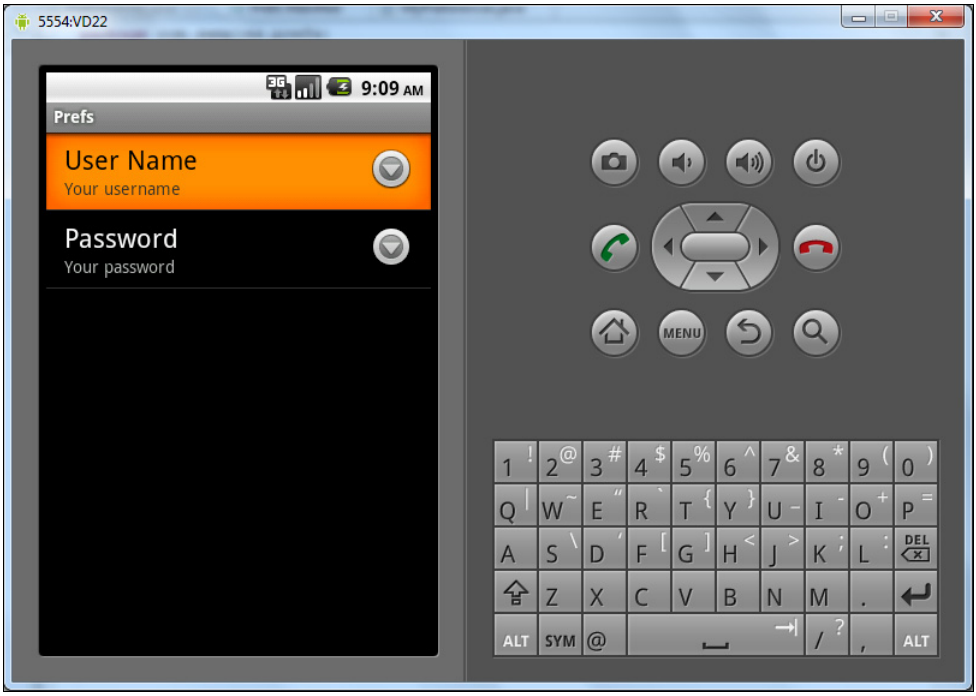


Рис. 9.3. Окно установки параметров приложения

Установите значение второго параметра. Поскольку мы указали, что второй параметр — это пароль, все его символы будут закрыты точками (рис. 9.5).

Если нажать в эмуляторе кнопку **Назад**, то мы увидим само приложение (рис. 9.6). Теперь проверим, сохранились ли настройки. Для этого закройте окно эмулятора и запустите проект снова на выполнение. После этого выберите первый параметр — появится то же диалоговое окно редактирования параметра, но в нем уже будет введенное ранее значение. Что и требовалось доказать!

На этом заканчивается вторая часть книги. В следующей части мы поговорим о более сложных материях: службах, доступе к базе данных, подключению к Интернету и т. д. Самое интересное еще впереди!

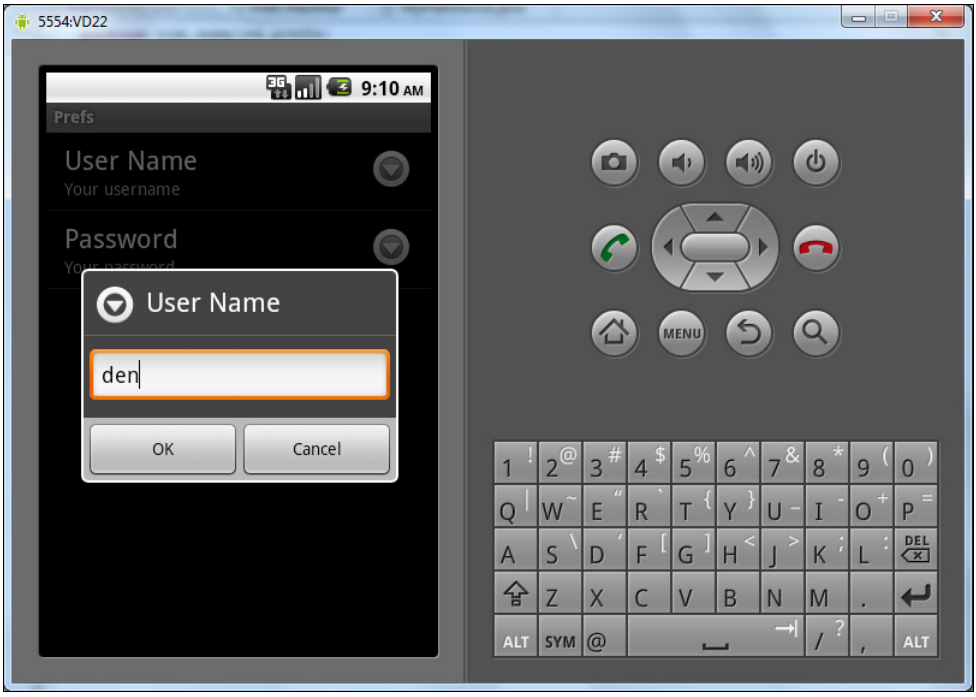


Рис. 9.4. Окно ввода текстового значения

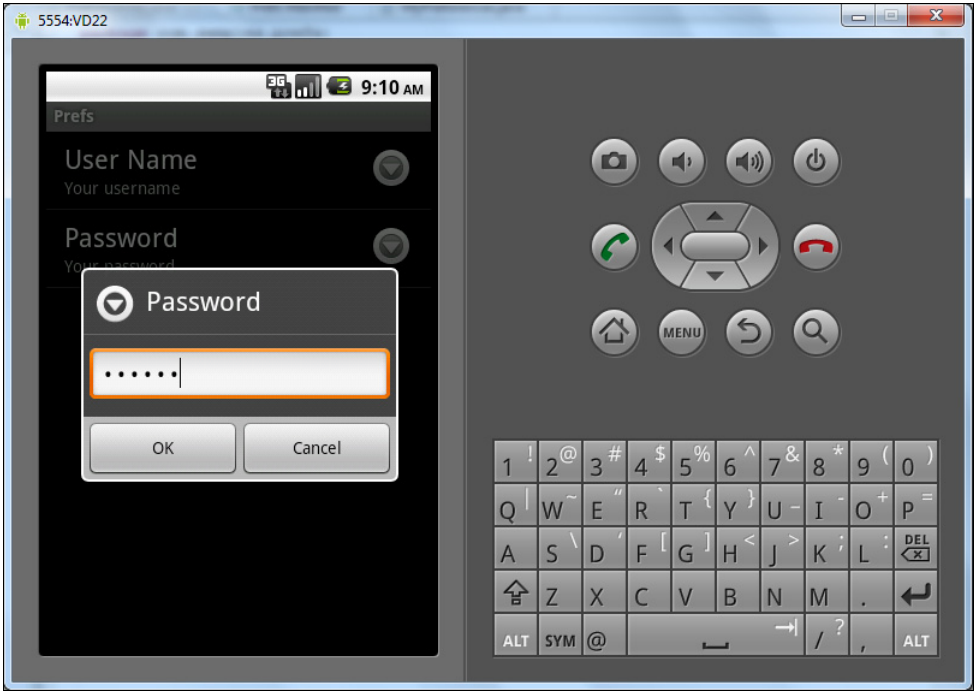


Рис. 9.5. Ввод пароля

Рис. 9.6. Основное окно приложения

ЧАСТЬ III

Профессиональное программирование

ГЛАВА 10

Деятельности и намерения. Передача данных между деятельностями

10.1. Еще раз о компонентах приложения

В *главе 4* мы познакомились с компонентами Android-приложения. Настало время взглянуть на них в другом свете — с высоты уже полученных знаний.

Итак, любое приложение может содержать следующие компоненты: `Activity` (деятельность), `Service` (служба), `BroadcastReceiver` (приемник широковещательных намерений) и `ContentProvider` (контент-провайдер).

Приложение должно состоять как минимум из одного такого компонента. В самых сложных случаях в приложении будут присутствовать все четыре компонента. Операционная система (и другие Android-приложения при соответствующих разрешениях) может вызывать необходимый ей компонент приложения.

Какие компоненты должны быть именно в вашем приложении? Все зависит от самого приложения, т. е. от того, что вам нужно в нем сделать.

- Если надо организовать интерфейс пользователя (например, получить от пользователя ввод), тогда в вашем приложении обязательно должна быть деятельность (`Activity`).
- При разработке фонового процесса — например, для воспроизведения музыки или фонового обновления деятельности, вам пригодится служба (`Service`).
- Для обращения к ресурсам телефона — например, доступа к контактам телефона, вам нужен контент-провайдер (`ContentProvider`).
- Для получения же широковещательных сообщений (с целью реакции на какие-то события телефона) используется `BroadcastReceiver`.

Теперь разберемся, что такое *намерения*. Намерение (`Intent`) — это действие. Компоненты приложения вызываются с помощью намерений. Например, вы можете вызвать браузер с помощью намерения:

```
Intent browser = Intent(Intent.ACTION_VIEW);
browser.setData(Uri.parse("http://www.dkws.org.ua"));
startActivity(browser);
```

Аналогично, вы можете вызвать компонент своего собственного приложения. Пусть у вас есть приложение, в котором имеются две (или более) деятельности (два или более окна). Вызвать вспомогательное окно можно только с помощью намерения.

Представим, что вы разрабатываете игру. В вашем приложении будет главное окно (главная деятельность) и, собственно, деятельность игры. Деятельность игры запускается по нажатию кнопки **Play** в главном окне (или элемента главного меню приложения — без разницы). Обработчик нажатия кнопки будет такой:

```
Intent startGame = new Intent(this, PlayGame.class);
startActivity(startGame);
```

В нашем проекте должен быть класс `PlayGame`. «Болванка» для этого класса приведена в листинге 10.1. Наш класс ничего не делает — он просто находит в файле разметки деятельности кнопку `end_game` и устанавливает для нее обработчик — функцию `finish()`. Напомню, чтобы добавить класс `PlayGame` в проект, нужно щелкнуть на проекте (в области **Package Explorer**) правой кнопкой мыши и выбрать команду **New | Class**.

Листинг 10.1. Файл PlayGame.java

```
package com.samples.startgame;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class PlayGame extends Activity {
       public void onCreate(Bundle savedInstanceState) {
              super.onCreate(savedInstanceState);
              // Файл разметки называется game.xml
              setContentView(R.layout.game);
              // Обработка кнопки завершения игры
              Button endButton = (Button) findViewById(R.id.end_game);
              endButton.setOnClickListener(new View.OnClickListener() {
                     public void onClick(View view) {
                            finish();
                     }
              });
       }
}
```

Файл разметки для деятельности `PlayGame` представлен в листинге 10.2.

Листинг 10.2. Файл разметки для деятельности `PlayGame` (res/layout/game.xml)

```
<?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"
>
<Button android:id="@+id/end_game"
       android:layout_width="100dip"
       android:layout_height="100dip"
       android:text="@string/end_game"
       android:layout_centerInParent="true"
/>
</LinearLayout>
```

В файле манифеста нужно зарегистрировать нашу новую деятельность и установить для нее действия: `VIEW` и `DEFAULT` (листинг 10.3).

Листинг 10.3. Файл манифеста для приложения с несколькими деятельностями

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
       android:versionCode="1"
       android:versionName="1.0"
       package="com.samples.startgame">
       <application android:icon="@drawable/icon"
              android:label="@string/app_name">
          <activity android:name=".Menu"
              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=".PlayGame"
              android:label="@string/app_name">
              <intent-filter>
              <action android:name="android.intent.action.VIEW" />
              <category android:name="android.intent.category.DEFAULT" />
              </intent-filter>
          </activity>
       </application>
<uses-sdk android:minSdkVersion="3" />
</manifest>
```

10.2. Однозадачный режим

Приложение может быть запущено пользователем несколько раз, в результате в памяти устройства окажутся запущенными несколько экземпляров одной и той же деятельности, что приведет к ее перерасходу. Чтобы избежать подобной ситуации, разработчик может контролировать поведение каждой деятельности с помощью файла манифеста. Для каждого элемента `<activity>`, обладающего `intent`-фильтрами `MAIN` и `LAUNCHER`, добавьте следующий параметр:

```
android:launchMode="singleInstance"
```

Должно получиться примерно так:

```
<activity android:name=".Menu"
       android:launchMode="singleInstance"
       android:label="@string/app_name">
       <intent-filter>
         <action android:name="android.intent.action.MAIN" />
         <category android:name="android.intent.category.LAUNCHER" />
       </intent-filter>
</activity>
```

В результате будет запущен только один экземпляр деятельности. Но вы можете задать еще более жесткое ограничение. По умолчанию каждая дочерняя деятельность запускается как отдельная задача. Чтобы все деятельности запускались как одна задача, используется следующий режим запуска:

```
android:launchMode="singleTask"
```

10.3. Ориентация экрана

Любое Android-устройство, оснащенное акселерометром, способно определить, в каком положении сейчас находится устройство. В зависимости от показаний акселерометра, может изменяться ориентация экрана: с альбомной на портретную (книжную) и наоборот. Но не всегда это хорошо. Одно дело, если вы разрабатываете офисное приложение, но совсем другое, когда разрабатывается игра. В таком случае изменение ориентации экрана может быть не совсем желательным. Но вы можете принудительно задать ориентацию экрана для каждой деятельности. Для этого в файле манифеста в элемент `<activity>` нужно добавить параметр `orientation`:

```
android:screenOrientation="portrait"
android:screenOrientation="landscape"
```

Значение `portrait` означает портретную (книжную) ориентацию, значение `landscape` — альбомную.

Скрыть клавиатуру можно путем добавления следующего параметра:

```
android:configChanges="orientation|keyboardHidden"
```

Иногда необходимо знать, когда скрыта клавиатура или когда изменена ориентация экрана. Тогда вам нужно переопределить метод `onConfigurationChanged()` (листинг 10.4).

Листинг 10.4. Переопределение метода `onConfigurationChanged()`

```
@Override
public void onConfigurationChanged(Configuration newConfig) {
    super.onConfigurationChanged(newConfig);

    // Проверяем ориентацию экрана
    if (newConfig.orientation == Configuration.ORIENTATION_LANDSCAPE) {
        Toast.makeText(this, "landscape", Toast.LENGTH_SHORT).show();
    } else
    if (newConfig.orientation == Configuration.ORIENTATION_PORTRAIT){
        Toast.makeText(this, "portrait", Toast.LENGTH_SHORT).show();
    }
    // Проверяем видимость клавиатуры
    if (newConfig.hardKeyboardHidden == Configuration.HARDKEYBOARDHIDDEN_NO) {
              Toast.makeText(this, "keyboard visible",
                             Toast.LENGTH_SHORT).show();
    } else
    if (newConfig.hardKeyboardHidden == Configuration.HARDKEYBOARDHIDDEN_YES) {
              Toast.makeText(this, "keyboard hidden",
                             Toast.LENGTH_SHORT).show();
    }
}
```

10.4. Сохранение и восстановление состояния деятельности

В *главе 9* мы рассмотрели способ хранения настроек приложения — предпочтения. Сейчас мы рассмотрим способ сохранения и восстановления состояния деятельности. Перед завершением работы деятельности вызывается функция `onSaveInstanceState()`, сохраняющая состояние деятельности. При повторном создании деятельности вызывается функция `onRestoreInstanceState()`. Она используется для восстановления состояния деятельности.

Рассмотрим, как реализовать функции `onSaveInstanceState()` и `onRestoreInstanceState()` (листинг 10.5).

Листинг 10.5. Сохранение и восстановление состояния деятельности

```
String my_string = "Some text";
float[] my_array = {1.0, 1.2, 1.3};
// Сохранение состояния деятельности
@Override
```

```
protected void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        // Сохраняем связанную информацию
        outState.putString("str", my_string);
        outState.putFloatArray("array", my_array);
}
// Восстановление состояния деятельности
@Override
public void onRestoreInstanceState(Bundle savedInstanceState) {
        super.onRestoreInstanceState(savedInstanceState);
        // Восстанавливаем информацию
        my_string = savedInstanceState.getString("str");
        my_array = savedInstanceState.getFloatArray("array");
}
```

10.5. Передача данных между деятельностями

Существует несколько способов передачи данных между деятельностями. Первый заключается в использовании публичных членов класса — примерно так:

```
public int flag;
...
MainActivity.flag = 0;
```

В этом способе нет ничего нового, и если вы знаете Java (искренне надеюсь, что это так), то я уверен, вы уже использовали этот способ. Но хочется чего-то, ориентированного на Android.

Со вторым способом — предпочтениями — вы также уже знакомы из *главы 3*. Осталось только два способа: экстраданные и использование базы данных SQLite. Сейчас мы узнаем, как задействовать экстраданные, а затем — в *главе 13* — будет подробно рассмотрено использование базы данных SQLite.

Ранее был приведен пример, когда одна деятельность вызывает другую:

```
Intent startGame = new Intent(this, PlayGame.class);
startActivity(startGame);
```

Представим, что нам нужно передать вызываемой деятельности данные. Для этого служит метод `putExtra()`:

```
int flag = 0;
int coins = 100;
...
Intent startGame = new Intent(this, PlayGame.class);
startGame.putExtra("flag", flag);
startGame.putExtra("coins", coins);
startActivity(startGame);
```

Первый параметр метода `putExtra()` — это название экстрапеременной, по которому можно обратиться к ней с целью получения ее значения. Второй параметр — это переменная, значение которой передается.

Для получения значения экстрапеременной (в коде класса `PlayGame`, файл PlayGame.java) вызывается метод `getExtra()`:

```
// Функция getIntent() возвращает намерение, запустившее эту деятельность
Intent i = getIntent();
// Получаем экстрапеременную с именем flag,
// если таковая не найдена, возвращаем значение по умолчанию — 0
flag = i.getIntExtra("flag", 0);
// Получаем экстрапеременную с именем coins, значение по
// умолчанию не задано
coins = i.getStringExtra("coins");
```

На этой ноте завершается десятая глава. В следующей главе мы поговорим о потоках, службах и широковещательных приемниках.

ГЛАВА 11

Потоки, службы и широковещательные приемники

11.1. Потоки

Потоки позволяют выполнять несколько задач одновременно, что дает возможность более эффективно использовать системные ресурсы. Потоки удобно создавать для фонового выполнения какой-либо части программы. Самый простой пример многопоточной программы — это музыкальный проигрыватель. Такая программа запускается и отображает список файлов и кнопки управления воспроизведением. Вы нажимаете кнопку **Play**, и запускается отдельный поток — поток воспроизведения композиции. Но ведь программа должна обрабатывать и нажатия других кнопок — например, кнопок **Stop** и **Pause**. Иначе бы после нажатия кнопки **Play** не было бы возможности остановить, приостановить воспроизведение или переключиться на другую композицию. Благодаря потокам такая возможность у нас есть.

11.1.1. Запуск потока

Представим, что мы создаем тот же музыкальный проигрыватель. В листинге 11.1 представлен обработчик кнопки **Play**, вызывающий функцию `play()` для воспроизведения музыки. Запуск этой функции происходит без создания нового потока.

Листинг 11.1. Версия обработчика кнопки *Play* (без потоков)

```
Button playButton = (Button) findViewById(R.id.play);
playButton.setOnClickListener(new View.OnClickListener() {
       public void onClick(View view){
              play();
       }
});
```

Как видите, ничего сверхъестественного нет. Просто вызывается функция `play()`.

Теперь реализуем то же самое, но с запуском функции `play()` в отдельном потоке. Первым делом нужно создать поток:

```
Thread myThread = new Thread(
// здесь будет описание объекта Runnable
);
```

Затем описать объект `Runnable` — это делается в конструкторе потока:

```
new Runnable() {
       public void run() {
              play();
       }
}
```

Как можно видеть, функция `play()` вызывается внутри потока.

Наконец, мы запускаем поток:

```
myThread.start();
```

Полный код создания потока выглядит так:

```
Thread myThread = new Thread(
       new Runnable() {
              public void run() {
                     play();
              }
       }
);
myThread.start();
```

Что будет делать функция `play()`? Все, что нужно сделать в потоке, — т. е. код этой функции зависит от создаваемой программы. В нашем случае будет создан объект класса `MediaPlayer` и вызван метод `start()` для воспроизведения музыки. Примерно так:

```
MediaPlayer media = new MediaPlayer();
media = MediaPlayer.create(this, R.raw.music1);
media.start();
```

«Усыпить» на время поток можно методом `sleep()`:

```
// засыпаем на 1 секунду
myThread.sleep(1000);
```

11.1.2. Установка приоритета потока

Для установки приоритета процесса используется метод `setPriority()`, который нужно вызвать до метода `start()`. Значение приоритета может лежать в диапазоне от `Thread.MIN_PRIORITY (1)` до `Thread.MAX_PRIORITY (10)`:

```
myThread.setPriority(10);
myThread.start();
```

11.1.3. Отмена выполнения потока

Мы должны позаботиться и об останове потока. Использовать метод `stop()` не рекомендуется, поскольку он оставляет приложение в неопределенном состоянии. Правильнее поток `myThread` завершать так (измените идентификаторы на идентификаторы вашего потока):

```
if(myThread != null) {
        Thread dummy = myThread;
        myThread = null;
        dummy.interrupt();
}
```

Существует и другой способ. Он заключается в том, что все запускаемые потоки вы объявляете демонами. В этом случае все запущенные потоки будут автоматически завершены при завершении основного потока приложения. На мой взгляд, это самый удобный вариант:

```
myThread.setDaemon(true);
myThread.start();
```

11.1.4. Обработчики *Runnable*-объектов: класс *Handler*

При сложных вычислениях может понадобиться очередь `Runnable`-объектов. Помещая такой объект в очередь, вы можете задать время его запуска, — например, спустя какое-то количество миллисекунд после помещения в очередь, или же указать точное время запуска объекта.

Для демонстрации обработчика потока мы разработаем программу, запускающую фоновый процесс, который будет каждые 200 мс получать текущее время и обновлять текстовую надпись в главной деятельности приложения. Кроме того, мы организуем кнопку **Start**, которую вы сможете нажимать. После первого нажатия надпись «Start» заменяется числом — количеством нажатий кнопки. По нажатию кнопки **Start** будет выполняться какое-либо действие, в нашем случае — это просто изменение надписи на кнопке. В то же время в фоновом режиме наш поток будет обновлять текстовую надпись.

Создайте новый проект. Код разметки приложения приведен в листинге 11.2.

Листинг 11.2. Разметка приложения

```
<?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:id="@+id/text"
android:layout_width="fill_parent"
```

```
android:layout_height="wrap_content"
android:text="@string/hello"
/>
<Button android:id="@+id/start"
android:text="Start"
/>
</LinearLayout>
```

Код приложения приведен в листинге 11.3.

Листинг 11.3. Код приложения

```
package com.samples.my_timer;

import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.os.SystemClock;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class MyTimer extends Activity {
       // Сколько раз была нажата кнопка
       private int buttonPressed=0;
       TextView buttonLabel;
       // Счетчик времени
       private long sTime = 0L;
       private TextView myTimeLabel;
       // Обработчик потока: обновляет сведения о времени
       private Handler myHandler = new Handler();

       @Override
       public void onCreate(Bundle savedInstanceState) {

       super.onCreate(savedInstanceState);
       setContentView(R.layout.main);
       if (sTime == 0L) {
              sTime = SystemClock.uptimeMillis();
              myHandler.removeCallbacks(TimeUpdater);
              // Добавляем Runnable-объект TimerUpdater в очередь
              // сообщений, объект должен быть запущен после
              // задержки в 100 мс.
              myHandler.postDelayed(TimeUpdater, 100);
       }
       myTimeLabel = (TextView) findViewById(R.id.text);
       buttonLabel = (TextView) findViewById(R.id.start);
```

```
        Button startButton = (Button) findViewById(R.id.start);
        startButton.setOnClickListener(new View.OnClickListener() {
        // Обработчик нажатия кнопки
        public void onClick(View view){
                buttonLabel.setText(++buttonPressed);
                }
                });
        }
        // Описание Runnable-объекта — нашего потока
        private Runnable TimeUpdater = new Runnable() {
                public void run() {
                // "Вычисляем время"
                        final long start = sTime;
                        long millis = SystemClock.uptimeMillis() — start;
                        int sec = (int) (millis / 1000);
                        int min = seconds / 60;
                        seconds = seconds % 60;
                        // Выводим время
                        myTimeLabel.setText("" + min + ":"
                        + String.format("%02d",sec));
                        // Задержка в 200 мс
                        myHandler.postDelayed(this, 200);
                } // void run()
        };
        @Override
        protected void onPause() {
        // Удаляем Runnable-объект
                myHandler.removeCallbacks(TimeUpdater);
                super.onPause();
        }
        @Override
        protected void onResume() {
                super.onResume();
        // Добавляем Runnable-объект
                myHandler.postDelayed(TimeUpdater, 100);
        }
}
```

Кроме метода `postDelayed()` вы можете использовать метод `postAtTime()`:

```
postAtTime(Runnable r, long uptimeMillis)
```

В этом случае объект `r` добавляется в очередь сообщений, запуск объекта производится во время, заданное вторым параметром (в миллисекундах).

Самый простой способ помещения объекта в очередь — метод `post()`, когда указывается только помещаемый объект, но не указывается время выполнения объекта:

```
post(Runnable r)
```

Подробно об обработчиках потока вы можете прочитать в руководстве разработчика:

http://developer.android.com/reference/android/os/Handler.html

11.2. Службы

Служба — компонент Android-приложения, запускаемый в фоновом режиме без всякого интерфейса пользователя. Служба может быть запущена или остановлена любым компонентом приложения. Пока служба запущена, любой компонент может воспользоваться предоставляемыми службой возможностями. Приведем несколько примеров использования служб.

Первая деятельность запускает службу, загружающую изображения на веб-сайт. Вторая деятельность подключается к этой службе с целью узнать, сколько файлов уже загружено, чтобы отобразить эту информацию пользователю.

Другой пример — первая деятельность позволяет пользователю выбрать композиции для воспроизведения (реализует список воспроизведения). Воспроизведением композиций занимается служба. Вторая деятельность предоставляет пользователю интерфейс управления воспроизведением: кнопки **Play**, **Stop**, **Pause** и т. д. При нажатии одной из кнопок происходит обращение к службе для выполнения определенного действия: остановки воспроизведения, приостановки (паузы) и т. п.

Службы отлично подходят для выполнения постоянных или регулярных операций и для обработки различных событий. Службы запускаются, останавливаются и контролируются разными компонентами приложения, в том числе другими службами, деятельностями (активностями), приемниками широковещательных намерений. Если вам нужно выполнять задачи, которые не зависят от взаимодействия с пользователем, то службы — это лучший выбор.

У запущенных служб приоритет выше, чем у невидимых деятельностей, поэтому менее вероятно, что служба будет при распределении ресурсов завершена преждевременно. Даже если такое произойдет, то, в отличие от деятельности, ваша служба автоматически перезапустится, как только окажется достаточно доступных ресурсов.

Если служба взаимодействует с пользователем, то нужно повысить ее приоритет до уровня деятельностей, которые работают на переднем плане. Это гарантирует, что служба завершится только в крайнем случае, но при этом ваше приложение может немного подтормаживать, что испортит от него впечатление.

Иногда вместо создания собственной службы проще использовать уже имеющиеся системные службы (табл. 11.1). Все зависит от того, какую задачу вам нужно решить.

Дополнительную информацию об этих службах вы сможете получить по ссылке: **http://developer.android.com/**.

Жизненный цикл службы изображен на рис. 11.1.

Таблица 11.1. *Системные службы*

Служба	Описание
Account Service	Управляет пользовательскими учетными записями
Activity Service	Управляет деятельностями
Alarm Service	Отправляет разовые или периодические оповещения
Clipboard Service	Используется для управления буфером обмена
Connectivity Service	Управляет сетевыми соединениями
Download Service	Управляет загрузками
Input Method Service	Управляет текстовым вводом
Layout Inflater Service	Управляет компоновкой экрана
Location Service	Занимается отслеживанием координат
NFC Service	Управляет NFC
Notification Service	Управляет уведомлениями
Power Service	Управляет энергопотреблением
Search Service	Управляет поиском
Sensor Service	Используется для доступа к датчикам
Telephony Service	Управляет телефонными функциями
Vibrator Service	Управляет доступом к виброзвонку
Wallpaper Service	Управляет обоями домашнего экрана
WiFi Service	Управляет соединениями Wi-Fi

Рассмотрим общий алгоритм создания службы. Первым делом нужно создать класс, расширяющий класс `Service`. В Eclipse для этого надо щелкнуть правой кнопкой мыши на проекте и выбрать команду **New | Class**. В качестве суперкласса следует указать класс `android.app.Service` (рис. 11.2).

По умолчанию будет создана «болванка» класса службы, представленная в листинге 11.4.

Листинг 11.4. «Болванка» класса службы

```
package com.samples.my_first_service;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;

public class MyService extends Service {

       @Override
       public IBinder onBind(Intent arg0) {
```

```
            // TODO Auto-generated method stub
            return null;
        }

}
```

В файле манифеста приложения служба описывается так:

```
<service android:name=".myService"></service>
```

Затем элемент `<service>` добавляется в элемент `<application>`. Пример кода манифеста представлен в листинге 11.5.

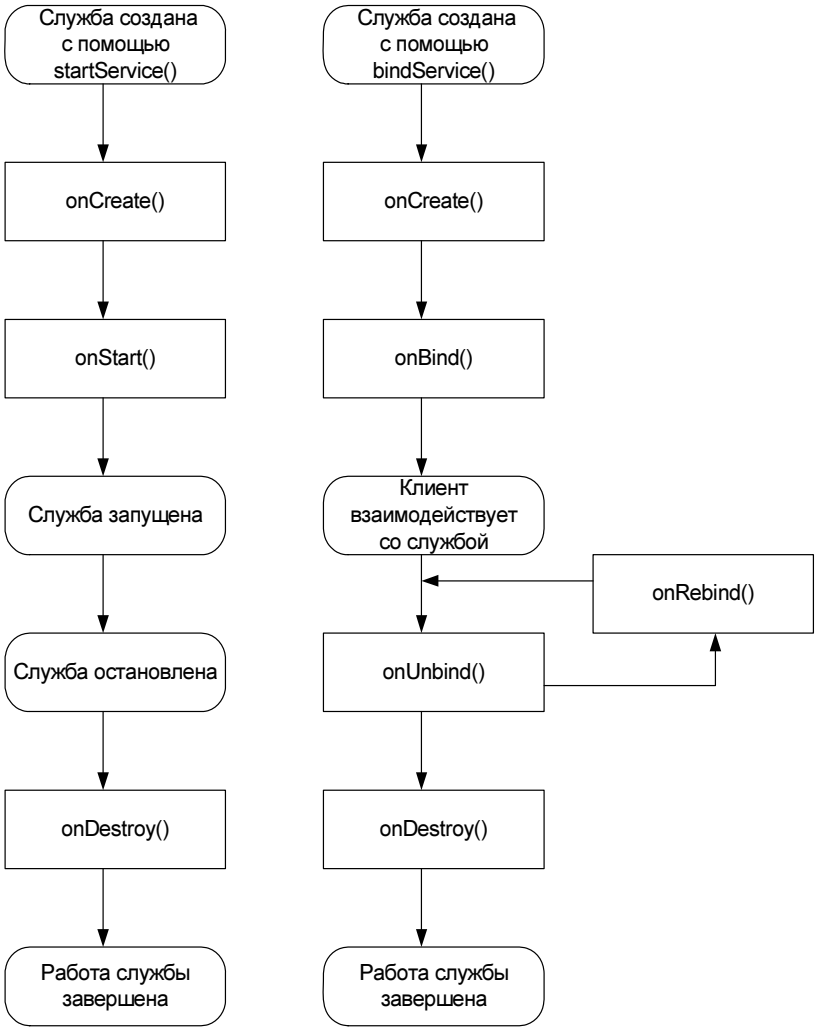


Рис. 11.1. Жизненный цикл службы

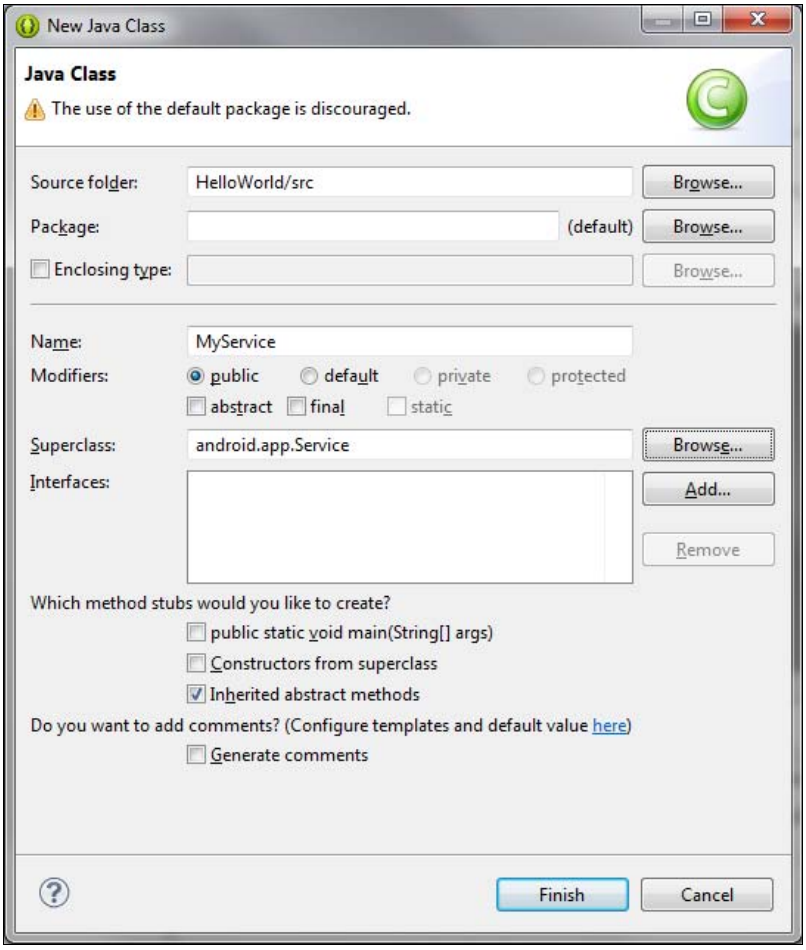


Рис. 11.2. Создание класса службы

Листинг 11.5. Пример кода манифеста для приложения с компонентом `Service`

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.cookbook.simple_service"
      android:versionCode="1"
      android:versionName="1.0">
      <application android:icon="@drawable/icon"
            android:label="@string/app_name">
            <activity android:name=".SimpleActivity"
            android:label="@string/app_name">
                  <intent-filter>
                        <action android:name="android.intent.action.MAIN" />
                        <category
                  android:name="android.intent.category.LAUNCHER" />
```

```
                </intent-filter>
            </activity>
            <service android:name=".SimpleService"></service>
        </application>
        <uses-sdk android:minSdkVersion="3" />
</manifest>
```

Теперь нужно переопределить методы `onCreate()` и `onDestroy()`. Для этого щелкните правой кнопкой на файле класса в Eclipse и выберите команду **Source | Override/Implement Methods** (рис. 11.3), после чего в открывшемся окне (рис. 11.4) установите флажки напротив упомянутых методов.

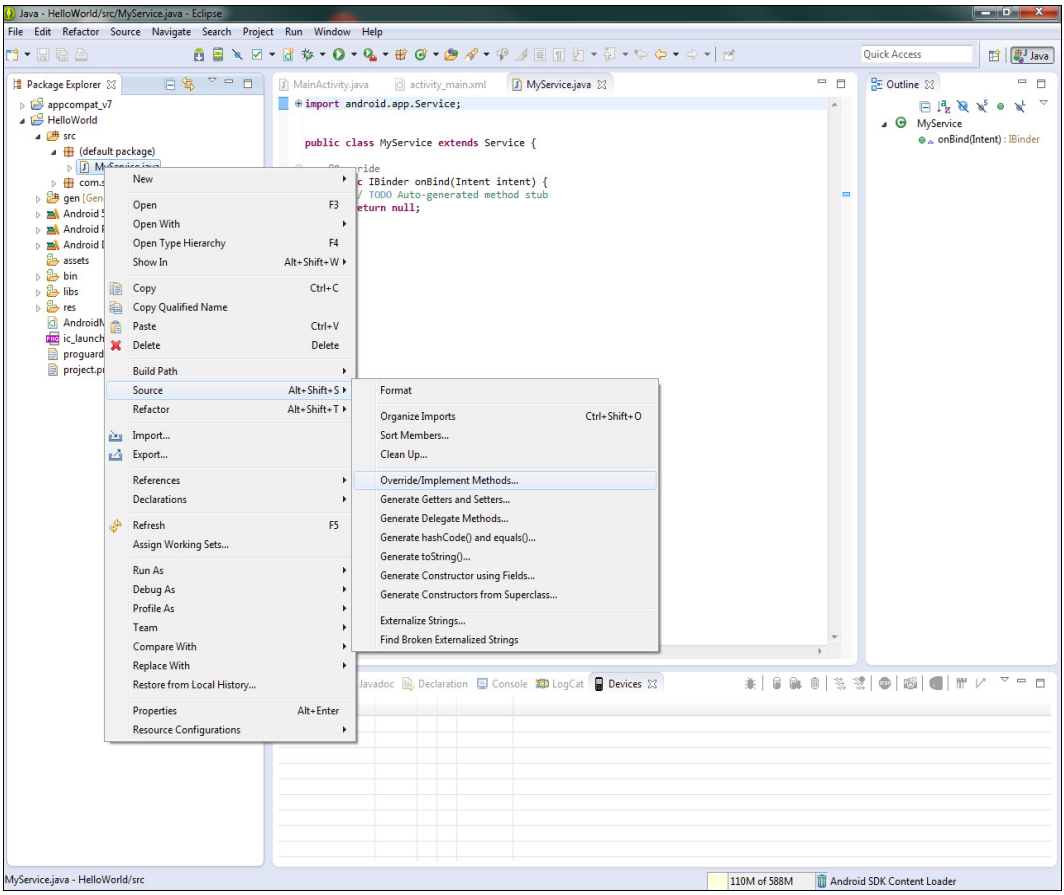


Рис. 11.3. Меню **Source**

Метод `onBind()` нужно переопределять в том случае, если новый компонент привязывается к этой службе после ее создания.

Запустите службу функцией `startService()`. Остановить службу можно функцией `stopService()`. Представим, что у нас есть деятельность `MainActivity` с кнопками

Start и **Stop**. Кнопка **Start** запускает сервис `MyService`, описанный в классе `MyService` (файл MyService.java). Обработчик нажатия этой кнопки будет выглядеть так:

```
startButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View view){
                startService(new Intent(MainActivity.this,
                MyService.class));
        }
});
```

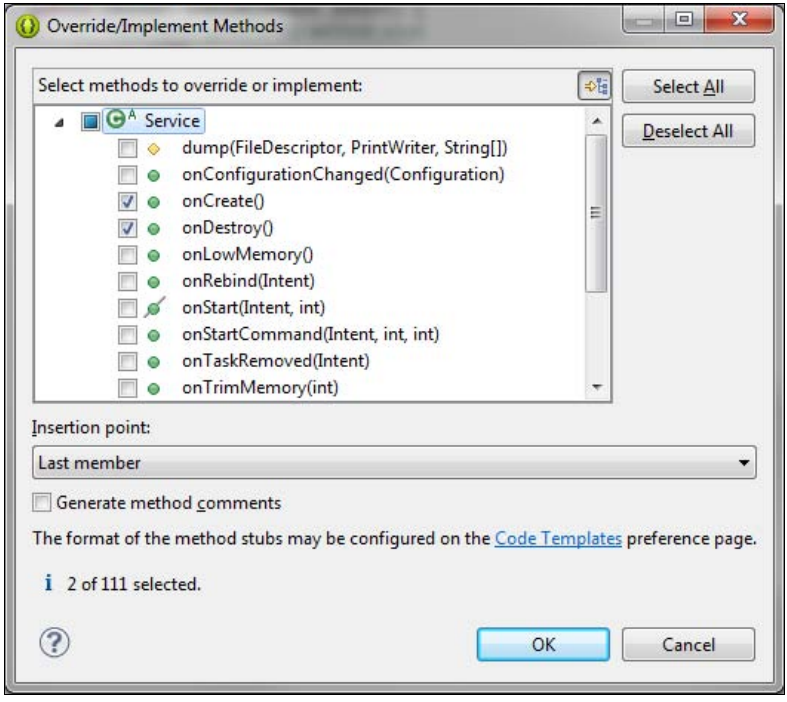


Рис. 11.4. Переопределяем методы `onCreate()` и `onDestroy()`

Обработчик кнопки **Stop** будет таким:

```
stopButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v){
                stopService(new Intent(MainActivity.this,
                MyService.class));
        }
});
```

В листинге 11.6 приведена расширенная «болванка» службы. Наша служба при запуске и останове выводит соответствующее уведомление.

Листинг 11.6. Расширенная «болванка» службы (файл MyService.java)

```
package com.samples.my_first_service;

import android.app.Service;
import android.content.Intent;
import android.media.MediaPlayer;
import android.os.IBinder;
import android.widget.Toast;

public class MyService extends Service {

        @Override
        public IBinder onBind(Intent arg0) {
                return null;
        }

        @Override
        public void onCreate() {
                super.onCreate();
                Toast.makeText(this,"Service started...",
                Toast.LENGTH_LONG).show();
                }

        @Override
        public void onDestroy() {
                super.onDestroy();
                Toast.makeText(this, "Service destroyed...",
                Toast.LENGTH_LONG).show();
        }
}
```

Если служба разрешает другим приложениям связываться с собой, то привязка выполняется посредством следующих методов обратного вызова:

- `IBinder onBind(Intent intent);`
- `onUnbind(Intent intent);`
- `onRebind(Intent intent).`

В метод `onBind()` передают объект `Intent`, который был параметром в методе `bindService()`, а в метод обратного вызова `onUnbind()` — объект `Intent`, который передавали в метод `unbindService()`. Если служба разрешает связывание, то метод `onBind()` возвращает канал связи, который могут использовать клиенты, чтобы работать со службой. Метод `onRebind()` может быть вызван после метода `onBind()`, если со службой соединяется новый клиент.

В завершение повествования о службах приведу код, выводящий в лог (вывод можно просмотреть в Eclipse) список всех запущенных служб:

```
ActivityManager am = (ActivityManager) this
       .getSystemService(ACTIVITY_SERVICE);
List<ActivityManager.RunningServiceInfo> services = am.getRunningServices(50);

for (int i = 0; i < services.size(); i++) {
       ActivityManager.RunningServiceInfo rsi = services.get(i);
       Log.i("Service", "Process " + rsi.process + " with component "
                     + rsi.service.getClassName());
}
```

11.3. Широковещательные приемники

Широковещательные приемники прослушивают широковещательные сообщения системы. Примеры таких сообщений:

- низкий заряд батареи;
- нажата кнопка камеры;
- установлено новое приложение.

Помимо системных событий, пользователь может создавать свои события, — например, когда поток завершил вычисления или когда поток начал работу.

Широковещательный приемник — это объект класса `BroadcastReceiver` или одного из его подклассов. Самый главный метод этого класса — `onReceive()`, который вызывается, когда приемник получает сообщение.

Рассмотрим пример запуска службы на основании получения широковещательного сообщения — нажатия кнопки камеры. Приемник прослушивает сообщения, фильтр намерения установлен на действие `Intent.ACTION_CAMERA_BUTTON`, которое соответствует нажатию кнопки камеры.

Зарегистрировать приемник можно функцией `registerReceiver()`, а удалить его — функцией `unregisterReceiver()`.

Итак, начнем создавать приложение. Код основной деятельности `MainActivity` представлен в файле MainActivity.java (листинг 11.7).

Листинг 11.7. Код основной деятельности (файл MainActivity.java)

```
package com.samples.my_receiver;

import android.app.Activity;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;

public class MainActivity extends Activity {
       // Создаем объект iReceiver класса MyReceiver
       MyReceiver iReceiver = new MyReceiver();
```

```
        /** Called when the activity is first created. */
        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.main);
                // Создаем фильтр намерения
                IntentFilter iFilter =
                        new IntentFilter(Intent.ACTION_CAMERA_BUTTON);
                // Добавляем действие в фильтр
                iFilter.addAction(Intent.ACTION_PACKAGE_ADDED);
                // Регистрируем приемник
                registerReceiver(iReceiver, iFilter);
        }
        @Override
        protected void onDestroy() {
        // Уничтожаем приемник при завершении приложения
                unregisterReceiver(iReceiver);
                super.onDestroy();
        }
}
```

Разберемся, что здесь что. Первым делом мы объявили объект `iReceiver` класса `MyReceiver`. Этот класс будет описан позже. Затем мы создали фильтр намерения:

```
IntentFilter iFilter = new IntentFilter(Intent.ACTION_CAMERA_BUTTON);
iFilter.addAction(Intent.ACTION_PACKAGE_ADDED);
```

После этого нужно только зарегистрировать приемник с помощью функции `registerReceiver()`. Первый параметр — это сам приемник, второй — фильтр намерений:

```
registerReceiver(iReceiver, iFilter);
```

Теперь надо создать класс `MyReceiver`. Как это сделать с помощью Eclipse, вы уже знаете, поэтому привожу сразу код класса (листинг 11.8).

Листинг 11.8. Код класса `MyReceiver` (файл MyReceiver.java)

```
package com.samples.my_receiver;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;

// Класс MyReceiver является расширением класса BroadcastReceiver
public class MyReceiver extends BroadcastReceiver {
        // Переопределяем метод onReceive()
        @Override
        public void onReceive(Context rcvContext, Intent rcvIntent) {
                String action = rcvIntent.getAction();
```

```
            // Если действие = Intent.ACTION_CAMERA_BUTTON
            if (action.equals(Intent.ACTION_CAMERA_BUTTON)) {
            // то запускаем сервис MyService
                  rcvContext.startService(new Intent(rcvContext,
                  MyService.class));
            }
      }
}
```

Приемник анализирует полученное действие: если нажата кнопка камеры, то будет запущен сервис (служба) `MyService`, описанный в файле MyService.java. Код этого файла представлен в листинге 11.6. Вам нужно изменить только первую строчку — она должна выглядеть так:

```
package com.samples.my_receiver;
```

В этой главе мы рассмотрели создание потоков, служб и широковещательных приемников. А в следующей поговорим о создании анимации в Android-приложениях.

ГЛАВА 12

Создание анимации

12.1. Анимация преобразований

В Android используются два подхода к созданию анимации: кадровая анимация (Frame Animation) и анимация преобразований (Tween Animation). Сначала мы рассмотрим анимацию преобразований, а затем — кадровую анимацию.

Анимация преобразований выполняется путем выполнения над изображением ряда преобразований: вращения, добавления прозрачности, изменения размера и т. д. В пакете `android.view.animationpackages` находятся все необходимые для выполнения преобразований классы:

- `AnimationSet` (XML-элемент `<set>`) — представляет группу анимаций, которые запускаются вместе;
- `AlphaAnimation` (XML-элемент `<alpha>`) — управляет прозрачностью объекта;
- `RotateAnimation` (XML-элемент `<rotate>`) — управляет вращением объекта;
- `ScaleAnimation` (XML-элемент `<scale>`) — управляет масштабированием (изменением размера) объекта;
- `TranslateAnimation` (XML-элемент `<translate>`) — управляет позиционированием объекта.

Анимацию преобразования можно создать как в XML-файле, так и в программном коде. Начнем с XML-файла. Файл анимации помещают в каталог res/anim. В этом файле имеется единственный корневой элемент. Таким элементом может быть один из элементов преобразования: `<rotate>`, `<alpha>`, `<scale>`, `<translate>` или `<set>`, который является контейнером для этих четырех элементов (он позволяет задать несколько преобразований).

По умолчанию все описанные в `<set>` преобразования выполняются одновременно. Чтобы они выполнялись последовательно (в порядке описания в XML-файле), нужно использовать атрибут `startOffset`, задающий задержку между выполнением преобразований в миллисекундах:

```
android:startOffset="1000"
```

У всех элементов преобразований есть общие атрибуты, описанные в табл. 12.1.

Таблица 12.1. *Общие атрибуты элементов преобразований*

Атрибут	Описание
`duration`	Продолжительность преобразования в миллисекундах
`startOffset`	Время смещения перед выполнением заданного эффекта, в миллисекундах. Можно использовать для задержки между преобразованиями
`fillBefore`	Если равен `true`, преобразование анимации выполняется перед началом анимации
`fillAfter`	Если равен `true`, преобразование выполняется после завершения анимации
`repeatCount`	Определяет количество повторений анимации
`repeatMode`	Позволяет задать режим повтора: 1 — анимация начинается заново с самого начала, 2 — анимация будет произведена в обратном порядке
`zAdjustment`	Смещение по оси *Z*: 0 (обычное смещение, без изменений); 1 (вершина); –1 (основание)
`interpolator`	Позволяет указать интерполятор анимации. Имя интерполятора указывается в формате `@[+][package:]type:name`. Пример: `android:interpolator="@android:anim/decelerate_interpolator`

Теперь рассмотрим собственные атрибуты элементов преобразования.

- У элемента `<set>` есть атрибут `shareInterpolator`. Если он равен `true`, то интерполятор, заданный атрибутом `interpolator` для `<set>`, будет использоваться для всех дочерних элементов преобразования, входящих в этот элемент `<set>`.
- У элемента `<alpha>` есть атрибуты `fromAlpha` и `toAlpha`. Первый задает начальное значение прозрачности объекта, второй — конечное. Значения прозрачности находятся в диапазоне от 0 до 1, где 0 — полная прозрачность объекта (объект почти невидим).
- У элемента `<translate>`, создающего вертикальную или горизонтальную анимацию, атрибутов больше:
  - `fromXDelta` — начальное положение по оси *X*;
  - `toXDelta` — конечное положение по оси *X*;
  - `fromYDelta` — начальное положение по оси *Y*;
  - `toYDelta` — конечное положение по оси *Y*.

  Атрибуты могут принимать либо абсолютное значение, либо значение в процентах: от –100 до 100 %. Можно также указывать процент относительно родителя: от `-100%p` до `100%p`.
- Элемент `rotate` служит для анимации вращения и поддерживает атрибуты:
  - `fromDegrees` — начальный угол вращения в градусах;
  - `toDegrees` — конечный угол вращения в градусах;

- `pivotX` — задает координату *X* центра вращения в пикселах;
- `pivotY` — задает координату *Y* центра вращения в пикселах.

❒ Элемент `<scale>` управляет изменением размера объекта. Вы можете использовать следующие атрибуты:

- `fromXScale` — начальный масштаб по оси *X*;
- `toXScale` — конечный масштаб по оси *X*;
- `fromYScale` — начальный масштаб по оси *Y*;
- `toYScale` — конечный масштаб по оси *Y*;
- `pivotX` — координата (*X*) закрепленного центра;
- `pivotY` — координата (*Y*) закрепленного центра.

Вернемся к практике. Первым делом нужно создать XML-файл анимации в каталоге res/anim. В листинге 12.1 приведен простой файл анимации, содержащий только элемент `<alpha>` внутри элемента `<set>`. В листинге 12.2 приведена более сложная анимация, т. к. задано два анимационных эффекта.

Листинг 12.1. Файл res/anim/alpha.xml

```
<?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:startOffset="0"
       android:duration="3000"/>
</set>
```

Листинг 12.2. Файл res/anim/advanced.xml

```
<?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:startOffset="0"
       android:duration="3000"/>
<translate
       android:toYDelta="-100"
       android:fillAfter="true"
       android:duration="2500"/>
</set>
```

Теперь займемся Java-кодом. Первым делом нужно подключить необходимые пакеты. Далее — загрузить картинку, над которой будет производиться анимация. Затем загрузить саму анимацию. Это можно сделать методом `loadAnimation()` из `AnimationUtils`. После чего запустить анимацию методом `startAnimation()`. Весь этот процесс описан в листинге 12.3.

Листинг 12.3. Создание анимации

```
// Подключаем необходимые пакеты
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.view.animation.Animation.AnimationListener;
...
// Загружаем картинку
ImageView image = (ImageView)findViewById(R.id.image);
// Загружаем анимацию из файла alpha.xml
Animation animation = AnimationUtils.loadAnimation(this, R.anim.alpha);
// Запускаем анимацию
image.startAnimation(animation);
```

Как видите, ничего сложного в анимации преобразования нет. Необходимо только подобрать нужный эффект (или группу эффектов).

12.2. Традиционная кадровая анимация

Традиционная кадровая анимация представляет собой последовательную смену различных изображений — подобно киноленте. Основой для этого типа анимации является класс `AnimationDrawable`.

Как и в случае с анимацией преобразований, начнем создавать анимацию с XML-файла, который, как обычно, нужно поместить в каталог res/anim. В листинге 12.4 приведен пример такого файла.

Листинг 12.4. Файл res/anim/frames.xml

```
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
android:oneshot="false">
<item android:drawable="@drawable/frame1" android:duration="100"/>
<item android:drawable="@drawable/frame2" android:duration="100"/>
<item android:drawable="@drawable/frame3" android:duration="100"/>
<item android:drawable="@drawable/frame4" android:duration="100"/>
<item android:drawable="@drawable/frame5" android:duration="100"/>
</animation-list>
```

У нас есть пять кадров (`@drawable/frame1` — `@drawable/frame5`), длительность отображения каждого кадра — 100 мс. Если параметр `android:oneshot` установлен в

false, то анимация будет циклической. Если же этот параметр принимает значение true, то анимация отобразится только один раз, а после остановки будет показан последний кадр.

Код запуска анимации несложен:

```
// Находим область ImageView
ImageView image = (ImageView)findViewById(R.id.image);
// Загружаем анимацию из файла frames.xml
image.setBackgroundResource(R.anim.frames);
AnimationDrawable myAnim = (AnimationDrawable)image.getBackground();
```

Запустить и остановить анимацию можно методами start() и stop():

```
myAnim.start();
myAnim.stop();
```

В листинге 12.5 приведен более полный код запуска анимации.

Листинг 12.5. Запуск кадровой анимации

```
// Подключаем необходимый пакет
import android.graphics.drawable.AnimationDrawable;
...
// Находим область ImageView
ImageView image = (ImageView)findViewById(R.id.image);
// Загружаем анимацию из файла frames.xml
image.setBackgroundResource(R.anim.frames);
AnimationDrawable myAnim = (AnimationDrawable)image.getBackground();
// Запускаем анимацию
myAnim.start();
```

В случае с кадровой анимацией можно обойтись и без XML-файла. Сначала нужно заполучить кадры, которые мы будем использовать для анимации (файлы frame1.png — frame5.png):

```
BitmapDrawabie frame1 =
(BitmapDrawable)getResources().getDrawable(R.drawable.frame1);
...
BitmapDrawabie frame5 =
(BitmapDrawable)getResources().getDrawable(R.drawable.frame5);
```

Затем создать объект класса AnimationDrawable. Атрибут oneshot устанавливается методом setOneShot() этого класса, а добавить кадры можно методом addFrame():

```
AnimationDrawable myAnim = new AnimationDrawable();
myAnim.setOneShot(false);
myAnim.addFrame(frame1, 100);
...
myAnim.addFrame(frame5, 100);
```

Далее порядок действий тот же. Нужно установить анимацию в качестве фона для объекта класса `ImageView` (объект `image`), сделать объект `AnimationDrawable` видимым и запустить анимацию:

```
image.setBackgroundDrawable(MyAnim);
MyAnim.setVisible(true, true);
MyAnim.start();
```

Какой из способов выбрать — решать вам. XML-файлы удобно использовать для постоянной анимации (кадры которой не изменяются), а вот формирование кадров через метод `addFrame()` полезно, когда производится загрузка файлов из внешнего источника, — например, из SD-карты устройства.

ГЛАВА 13

База данных SQLite

13.1. Несколько слов о базах данных

В этой книге мы рассмотрели почти все основные способы хранения данных на Android-устройстве. Осталось описать базы данных, которые используются для хранения более сложных структур данных.

Мы не станем изучать основы баз данных или основы языка запросов SQL. Предполагается, что вы уже знакомы с этими азами. Если это не так, то на полках книжного магазина, я уверен, вы найдете много книг, способных заполнить этот пробел в ваших знаниях.

В ОС Android используется система управления базами данных SQLite. Да, возможности SQLite относительно скромны. Но ведь Oracle в мобильном устройстве и не нужен. Зато SQLite на сегодняшний день — самая быстрая СУБД для несложных запросов.

На Android-устройстве база данных хранится в каталоге /data/data/<*имя пакета*>/databases. С помощью контент-провайдера несколько приложений могут использовать одну и ту же базу данных.

Основные этапы при работе с базой данных следующие:

1. Создание и открытие базы данных.
2. Создание таблицы.
3. Создание Insert-интерфейса (служит для вставки данных).
4. Создание Query-интерфейса (используется для выполнения запроса, обычно для выборки данных).
5. Закрытие базы данных.

Далее на примере создания приложения-блокнота будут продемонстрированы основные приемы при работе с базой данных: вставка, удаление и выборка записей. Наше простое приложение позволяет пользователю добавлять, удалять и просматривать небольшие текстовые записи — заметки.

13.2. Класс *SQLiteOpenHelper*

Класс `SQLiteOpenHelper` управляет созданием базы данных. Мы создадим класс `MyDataHelper`, который является расширением класса `android.database.sqlite.SQLiteOpenHelper`. Задача нашего класса — подготовить базу данных `mydatabase.db` и создать таблицу `table1`, которая будет хранить записи нашего блокнота.

Имя базы данных и таблицы задаются переменными:

```
private static final String DATABASE_NAME = "mydatabase.db";
private static final String TABLE_NAME = "table1";
```

Структура нашей таблицы будет следующей:

```
CREATE TABLE table1 (
id INTEGER PRIMARY KEY,
name TEXT
)
```

Таблица создается в методе `onCreate()`:

```
db.execSQL("CREATE TABLE " + TABLE_NAME + "(id INTEGER PRIMARY KEY, name
TEXT)");
```

Метод `onUpgrade()` задает действие при обновлении таблицы. Мы сначала удаляем существующую таблицу, а потом создаем ее заново:

```
db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
onCreate(db);
```

За выборку всех записей отвечает метод `selectAll()`. Мы сначала с помощью метода `query()` отправляем SQL-запрос, а затем читаем результат и возвращаем его в виде списка:

```
public List<String> selectAll() {
       List<String> list = new ArrayList<String>();
       Cursor cursor = this.db.query(TABLE_NAME, new String[] { "name" },
         null, null, null, null, "name desc");
       if (cursor.moveToFirst()) {
         do {
            list.add(cursor.getString(0));
         } while (cursor.moveToNext());
       }
       if (cursor != null && !cursor.isClosed()) {
         cursor.close();
       }
       return list;
   }
```

Полный код класса `MyDataHelper` приведен в листинге 13.1.

Листинг 13.1. Код класса `MyDataHelper` (файл MyDataHelper.java)

```
package com.samples.android_db;

import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteStatement;
import java.util.ArrayList;
import java.util.List;

public class MyDataHelper {
   // Имя базы данных
   private static final String DATABASE_NAME = "mydatabase.db";
   private static final String TABLE_NAME = "table1";
   private static final int DATABASE_VERSION = 1;

   private Context context;
   private SQLiteDatabase db;

   private static final String INSERT = "insert into "
      + TABLE_NAME + "(name) values (?)";
   private SQLiteStatement insertStmt;

   public DataHelper(Context context) {
      this.context = context;
      OpenHelper openHelper = new OpenHelper(this.context);
      this.db = openHelper.getWritableDatabase();
      this.insertStmt = this.db.compileStatement(INSERT);
   }

       // Действие при вставке записи
   public long insert(String name) {
      this.insertStmt.bindString(1, name);
      return this.insertStmt.executeInsert();
   }

   public void deleteAll() {
       // Удаляем все записи в таблице
       this.db.delete(TABLE_NAME, null, null);
   }

   public List<String> selectAll() {
      List<String> list = new ArrayList<String>();
      Cursor cursor = this.db.query(TABLE_NAME, new String[] { "name" },
        null, null, null, null, "name desc");
```

```
      if (cursor.moveToFirst()) {
         do {
            list.add(cursor.getString(0));
         } while (cursor.moveToNext());
      }
      if (cursor != null && !cursor.isClosed()) {
         cursor.close();
      }
      return list;
   }

   private static class OpenHelper extends SQLiteOpenHelper {

      OpenHelper(Context context) {
         super(context, DATABASE_NAME, null, DATABASE_VERSION);
      }

      @Override
      public void onCreate(SQLiteDatabase db) {
       // Запрос создания таблицы
         db.execSQL("CREATE TABLE " + TABLE_NAME + "(id INTEGER PRIMARY KEY,
name TEXT)");
      }

      @Override
      public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
{
       // Запрос удаления таблицы
         db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
         onCreate(db);
      }
   }
}
```

13.3. Разработка блокнота

Итак, каркас для нашего приложения создан. Теперь можно приступить к созданию самого приложения. Наше приложение будет хранить небольшие текстовые заметки. При запуске приложения заметки будут отображены в основной деятельности приложения, у пользователя будет возможность добавить новую заметку и удалить все заметки из таблицы.

Разработку приложения начнем с файла разметки, код которого представлен в листинге 13.2.

Листинг 13.2. Файл разметки /res/layout/main

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
       android:layout_width="fill_parent"
       android:layout_height="wrap_content">
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
       android:orientation="vertical"
       android:layout_width="fill_parent"
       android:layout_height="fill_parent">

<EditText android:layout_width="fill_parent"
       android:id="@+id/Edit"
       android:layout_height="wrap_content"/>

<Button android:layout_width="fill_parent"
       android:layout_height="35px"
       android:id="@+id/Insert"
       android:text="Вставить" />

<Button android:layout_width="fill_parent"
       android:layout_height="35px"
       android:id="@+id/DeleteAll"
       android:text="Удалить все" />

<Button android:layout_width="fill_parent"
       android:layout_height="35px"
       android:id="@+id/Refresh"
       android:text="Обновить" />

<TextView android:layout_width="fill_parent"
       android:text="@+id/TextView01"
       android:id="@+id/Records"
       android:layout_height="wrap_content" />
</LinearLayout>
</ScrollView>
```

Итак, у нас есть:

- текстовое поле `Edit` — используется для ввода заметки;
- кнопка `Insert` — используется для вставки в таблицу заметки, указанной в `Edit1`;
- кнопка `DeleteAll` — удаляет все записи;
- текстовая область `Records` — отображает записи таблицы;
- кнопка `Refresh` — обновляет текстовую область `Records`.

Основной код приложения, устанавливающий реакции на нажатия кнопок, представлен в листинге 13.3. Код снабжен подробными комментариями, поэтому, прежде чем перейти к практике, внимательно прочитайте этот листинг.

Листинг 13.3. Файл android_db.java (основной файл кода приложения)

```
package com.samples.android_db;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import java.util.List;
import com.samples.android_db.R;

public class Main extends Activity {
          // Наш класс MyDataHelper
          private MyDataHelper dh;

           @Override
           public void onCreate(Bundle savedInstanceState) {
               super.onCreate(savedInstanceState);
               setContentView(R.layout.main);
               // Находим кнопки и устанавливаем обработчики
               Button ButDel = (Button) findViewById(R.id.DeleteAll);
               Button ButRef = (Button) findViewById(R.id.Refresh);
               Button ButIns = (Button) findViewById(R.id.Insert);
               ButDel.setOnClickListener(delete_all_action);
               ButRef.setOnClickListener(refresh_action);
               ButIns.setOnClickListener(insert_action);
               // Находим текстовую область
               TextView Rc = (TextView) findViewById(R.id.Records);

               dh = getDataHelper();
               // В реальном приложении нужно удалить четыре следующих
               // оператора (удаление всех записей и добавление трех
               // тестовых записей)
               dh.deleteAll();
               dh.insert("Заметка 1");
               dh.insert("Заметка 2");
               dh.insert("Заметка 3");
              // Загружаем данные из таблицы в буфер sb
               List<String> names = this.dh.selectAll();
               StringBuilder sb = new StringBuilder();
               sb.append("Записи в таблице:\n");
```

```
            for (String name : names) {
               sb.append(name + "\n");
            }
            // Устанавливаем содержимое буфера sb в качестве
            // значения текстовой области
            Rc.setText(sb);
        }

        public DataHelper getDataHelper(){
           return new DataHelper(this);
        }
     // Обработчик нажатия кнопки Удалить все
     private OnClickListener delete_all_action = new OnClickListener()
    {
                public void onClick(View v) {
                       dh = getDataHelper();
                       // Удаляем все записи
                       dh.deleteAll();
            }
    };
    // Действие при нажатии кнопки Обновить
    private OnClickListener refresh_action = new OnClickListener() {
           public void onClick(View v) {
                  dh = getDataHelper();
                  // Находим текстовую область
                  TextView Rc = (TextView) findViewById(R.id.Records);
           // Выборка всех записей
                  List<String> names = dh.selectAll();
                  StringBuilder sb = new StringBuilder();
                  sb.append("Записи в таблице:\n");
                  for (String name : names) {
                         sb.append(name + "\n");
                  }
                  Rc.setText(sb);
           }
    };
    // Обработчик нажатия кнопки Вставить
    private OnClickListener insert_action = new OnClickListener() {
           public void onClick(View v) {
                  dh = getDataHelper();
                  // Находим текстовое поле
                  EditText memo = (EditText) findViewById(R.id.Edit);
                  // Получаем текст из поля и добавляем его в БД
                  dh.insert(memo.getText().toString());

           }
    };
}
```

Вот теперь, когда прочитанный материал «разложен по полочкам», можно приступать к созданию проекта в Eclipse. Создайте файлы проекта и запустите его. Вы увидите приложение, тестовая таблица которого содержит три проверочные заметки (рис. 13.1).

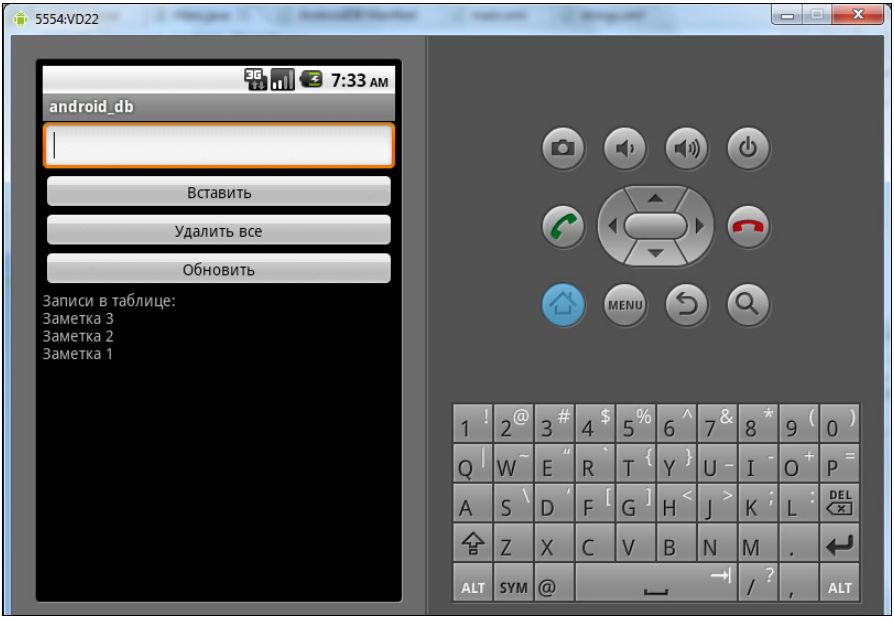


Рис. 13.1. Приложение запущено

Рис. 13.2. Виртуальная клавиатура

Активируйте текстовое поле — появится виртуальная клавиатура (рис. 13.2). Чтобы ее скрыть, нажмите кнопку **Назад** на вашем устройстве (или на клавиатуре эмулятора). Во время ввода текста появится словарь для облегчения ввода (рис. 13.3). Нажмите кнопку **Вставить** — в таблицу будет добавлена новая запись (рис. 13.4).

Рис. 13.3. Помощник при вводе

Рис. 13.4. Новая запись добавлена в таблицу

ГЛАВА 14

Соединение с внешним миром

14.1. Отправка SMS

Для отправки SMS используется класс `SmsManager`. В ранних версиях Android этот класс был помещен в пакет `android.telephony.gsm`, сейчас (начиная с версии 1.5) он находится в пакете `android.telephony`.

Отправить SMS довольно просто. Первым делом нужно добавить в файл манифеста соответствующее разрешение:

```
<uses-permission android:name="android.permission.SEND_SMS">
```

Затем надо определить объект класса `SmsManager` — для этого используется статический метод `getDefault()`. Далее следует определить получателя сообщения (его номер телефона) и текст самого сообщения:

```
import android.telephony.SmsManager;
import android.telephony.SmsMessage;
...
SmsManager sendSMS = SmsManager.getDefault();
String num = "номер получателя";
String msg = "My first SMS";
sendSMS.sendTextMessage(num, null, msg, null, null);
```

Отправляет SMS метод `sendTextMessage()`. Обычно достаточно указать первый и третий его параметры. Первый параметр задает номер телефона получателя, третий — текст сообщения. Второй параметр позволяет указать номер SMS-центра. Если указано значение `null`, будет использован номер SMS-центра, указанный в настройках устройства. Четвертый и пятый параметры служат для отслеживания факта отправки и доставки сообщения соответственно.

Разберемся, как использовать четвертый и пятый параметры:

```
// Определяем флаги отправки и доставки SMS
String SENT_SMS_FLAG = "SENT_SMS";
String DELIVER_SMS_FLAG = "DELIVER_SMS";
```

```
// Создаем соответствующие действия

// Действие, связанное с отправкой SMS
Intent sent_sms = new Intent(SENT_SMS_FLAG);
// Отложенная деятельность, связанная с sent_sms
PendingIntent spin = PendingIntent.getBroadcast(this,0,sent_sms,0);

// Аналогично для доставки:
Intent deliver_sms = new Intent(DELIVER_SMS_FLAG);
PendingIntent dpin = PendingIntent.getBroadcast(this,0,deliver_sms,0);
```

Теперь создаем объект `BroadcastReceiver`, необходимый для получения результата. Такой `BroadcastReceiver` нужно зарегистрировать для каждого отложенного действия:

```
// Получаем отчет об отправке
BroadcastReceiver sentReceiver = new BroadcastReceiver(){
@Override public void onReceive(Context c, Intent in) {
      switch(getResultCode()){
            case Activity.RESULT_OK:
                  // SMS отправлено, выполняем какие-то действия
                 break;
            default:
                 // Сбой
                 break;
     }
} };
// Получаем отчет о доставке
BroadcastReceiver deliverReceiver = new BroadcastReceiver(){
@Override public void onReceive(Context c, Intent in) {
      switch(getResultCode()){
            case Activity.RESULT_OK:
                  // SMS доставлено, выполняем какие-то действия
                  break;
            default:
                  // Сбой
                  break;
     }
} };
// Регистрируем BroadcastReceiver
registerReceiver(sentReceiver, new IntentFilter(SENT_SMS_FLAG));
registerReceiver(deliverReceiver, new IntentFilter(DELIVER_SMS_FLAG));
```

В большинстве случаев SMS не должно превышать 140 символов. Для отправки более длинных сообщений служит метод `divideMessage()`, разбивающий сообщения на фрагменты, равные максимальному размеру SMS-сообщения. Отправка такого сообщения осуществляется методом `sendMultipartTextMessage()`, который используется вместо метода `sendTextMessage()`. Для получения отчета о доставке (или от-

правке) нужно задействовать уже не одно отложенное событие, а массив таких событий. Количество элементов в таком массиве будет равно количеству частей, на которые было разбито исходное сообщение:

```
ArrayList<String> multiSMS = sendSMS.divideMessage(msg);
ArrayList<PendingIntent> sent_sms = new ArrayList<PendingIntent>();
ArrayList<PendingIntent> deliver_sms = new ArrayList<PendingIntent>();

for(int i=0; i< multiSMS.size(); i++){
        sentIns.add(sentIn);
        deliverIns.add(deliverIn);
}

sendSMS.sendMultipartTextMessage(num, null,
multiSMS, sentIns, deliverIns);
```

Ранее было сказано, что максимальная длина SMS составляет 140 байтов. Обратите внимание: именно байтов, а не символов. Когда вы отправляете SMS латинскими символами, то 140 байтов означает 140 символов. Когда же вы используете в SMS-сообщении символы национальных алфавитов — например, кириллицу, то максимальная длина сокращается до 70 символов. Для кодирования символов национальных алфавитов используется кодировка UCS-2, где каждый символ представлен двумя байтами (16-ю битами), поэтому количество символов, которые можно отправить в одной SMS, сокращается до 70. Помните об этом!

14.2. Работа с браузером

Запустить браузер для отображения заданной странички можно с помощью действия `ACTION_NEW` — сделать это достаточно просто:

```
Intent browser = Intent(Intent.ACTION_VIEW);
browser.setData(Uri.parse("http://www.dkws.org.ua"));
startActivity(browser);
```

Но запуск браузера нам мало интересен. Подумайте сами: напишете вы приложение, запускающее браузер. И какой толк от него будет? Пользователю проще напрямую запустить браузер.

Было бы гораздо интереснее создать свой браузер на основе класса `WebView`. Класс `WebView` использует для отображения веб-страниц движок WebKit (открытый движок браузера) — на этом же движке построен браузер Apple Safari и некоторые другие.

В файл манифеста для использования Интернета нужно добавить разрешение:

```
<uses-permission android:name="android.permission.INTERNET" />
```

К приложению надо подключить два пакета:

```
import android.webkit.WebView;
import android.webkit.WebSettings;
```

Есть два способа заполучить объект класса `WebView`. Первый заключается в использовании конструктора класса `WebView`, второй — в получении этого объекта из разметки приложения (предварительно его нужно поместить в разметку с помощью визуального редактора разметки):

```
// Первый способ
WebView browser = new WebView(this);
// Второй способ
WebView browser = (WebView) findViewById(R.id.webview);
```

После этого можно загрузить документ:

```
browser.loadUrl("http://www.dkws.org.ua/");
```

При желании можно загружать HTML-код из строки:

```
String html = "<html><body><h1>Hello</h1></body></html>";
browser.loadData(html, "text/html", "utf-8");
```

Для настройки браузера используется класс `WebSettings`:

```
WebSettings webSettings = webView.getSettings();
// Блокируем картинки для экономии трафика
webSettings.setBlockNetworkImage(true);
// Запрещаем сохранять данные форм
webSettings.setSaveFormData(false);
// Разрешаем JavaScript
webSettings.setJavaScriptEnabled(true);
// Запрещаем сохранять пароли
webSettings.setSavePassword(false);
// Устанавливаем размер шрифта по умолчанию (от 1 до 72)
webSettings.setDefaultFixedFontSize(2);
// Устанавливаем название нашего браузера
webSettings.setUserAgentString("My browser v 1.0");
```

Подробно о методах класса `WebSettings` (а значит, и о параметрах браузера) вы можете прочитать в руководстве разработчика Android:

http://developer.android.com/reference/android/webkit/WebSettings.html

Класс `WebView` тоже описан в руководстве разработчика:

http://developer.android.com/reference/android/webkit/WebView.html

ГЛАВА 15

Платформа Titanium Mobile

15.1. Основные сведения о Titanium Mobile

В последнее время очень популярны приложения RIA (Rich Internet Applications). Такие приложения предоставляют пользователю удобный интерфейс, анимацию, возможность перемещения объектов — и все это в окне браузера. Примерами средств разработки RIA-приложений служат Flash/Flex от Adobe, JavaFx от Sun, SilverLight от Microsoft, Laszlo от Laszlo Systems. Если вы до этого занимались веб-разработкой, то все эти средства вам знакомы.

Создавать RIA-приложения для операционной системы Android вы можете благодаря платформе Titanium Mobile. Да, теперь можно писать приложения для Android на языке JavaScript.

В качестве ресурсов приложения, разработанного с помощью Titanium Mobile, могут выступать HTML- и CSS-файлы. И, с одной стороны, может показаться, что платформа Titanium Mobile облегчает создание Android-приложений, поскольку программист, ранее не знакомый с Android-программированием, может использовать привычные средства HTML, CSS и JavaScript.

С другой стороны, не зная специфики разработки Android-приложения, вы не сможете создать полноценное приложение с помощью Titanium Mobile. По крайней мере, основы Android-программирования изучить придется. Но вам об этом не нужно беспокоиться, поскольку вы их уже освоили.

Платформу Titanium Mobile нужно воспринимать как надстройку над Android SDK, ориентированную на разработку именно RIA-приложений.

Основные возможности Titanium Mobile:

- поддержка стандартных веб-технологий HTML, CSS и JavaScript для всех платформ и языков PHP, Python и Ruby для настольных платформ;
- интегрированная поддержка AJAX, jQuery, MooTools, YUI и др.;
- доступ к функциям мобильного устройства: геолокация, акселерометр, карты и др.

15.2. Установка Titanium Studio

Для начала работы с Titanium Mobile нужно скачать и установить ее среду разработки Titanium Studio. Установку среды следует производить на компьютер, на котором уже установлен и настроен Android SDK.

Для загрузки Titanium Studio перейдите по следующему адресу:

http://www.appcelerator.com/titanium/

Нажмите кнопку **Download for Free** (рис. 15.1), заполните несложную регистрационную форму, и начнется загрузка установочного файла Titanium_Studio.exe.

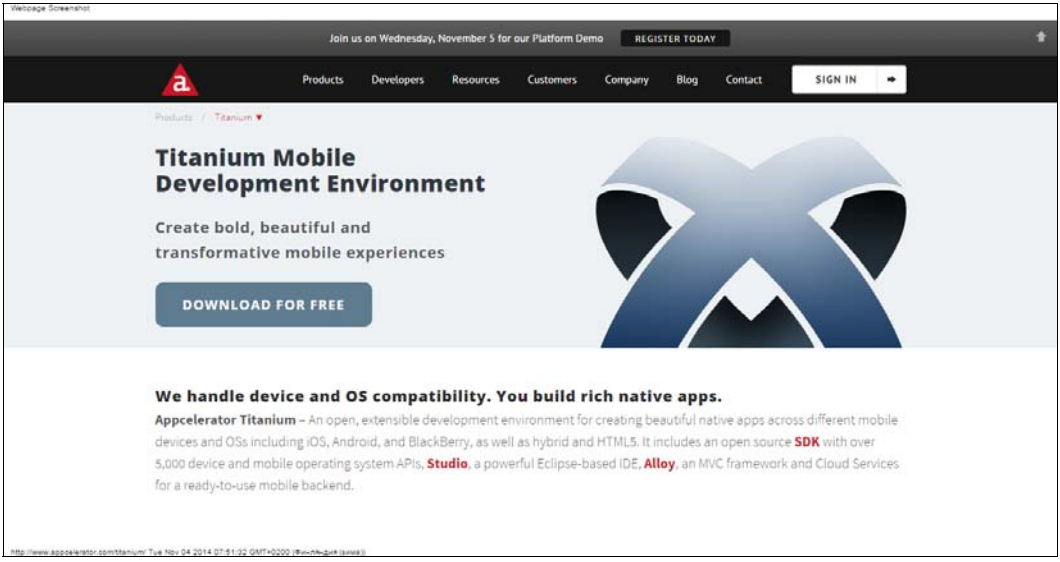


Рис. 15.1. Сайт установки Titanium Mobile

Загруженная вами версия распространяется бесплатно по лицензии Apache 2.0, с которой вы можете ознакомиться при установке Titanium Studio (рис. 15.2). Точнее, при установке вам сообщают только ссылку на лицензию, да и то устаревшую. Если на ней щелкнуть, то она приведет сюда: **http://www.appcelerator.com/legal/appexplore-agreement/** — текст все той же лицензии Apache 2.0.

На следующем шаге установки Titanium Studio, как обычно, предлагается выбрать каталог для установки (рис. 15.3).

Больше никаких вопросов не задается, и вся установка проходит без участия пользователя (рис. 15.4).

При первом запуске Titanium Studio вы увидите окно выбора каталога рабочего пространства — аналогичное окно появляется при запуске Eclipse (рис. 15.5).

Затем откроется окно ввода Email и пароля (рис. 15.6) — впишите Email и пароль, указанные при регистрации на сайте **www.appcelerator.com**.

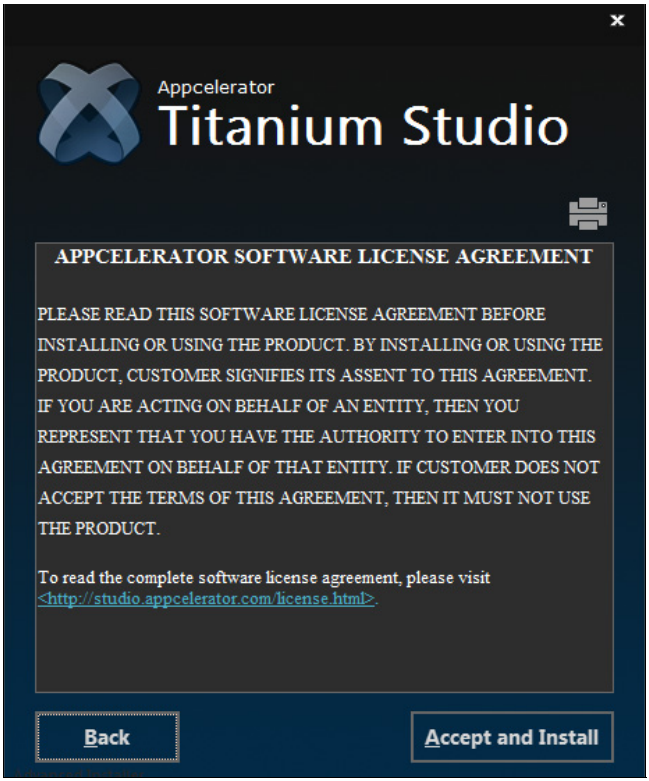


Рис. 15.2. Принимаем лицензию

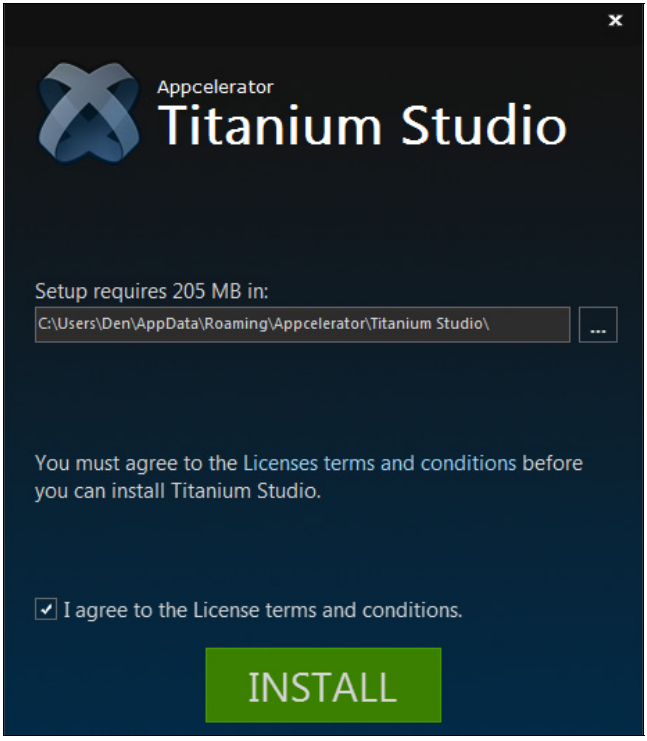


Рис. 15.3. Выбираем каталог для установки

Рис. 15.4. Установка продукта

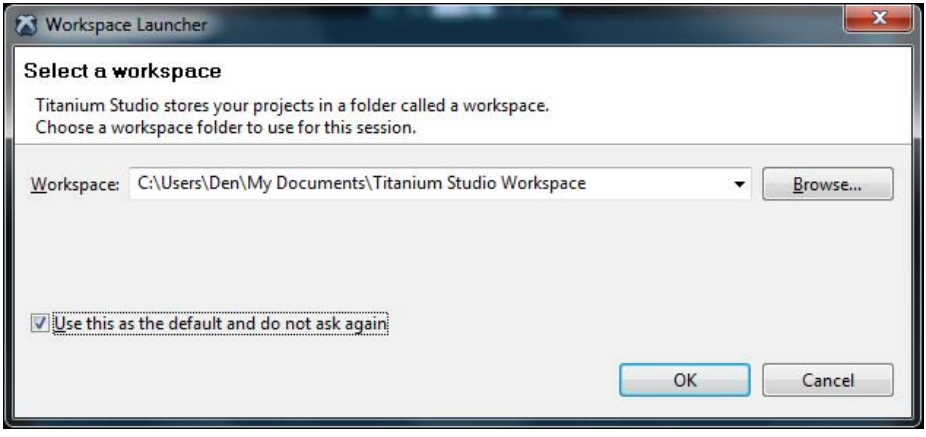


Рис. 15.5. Выбираем рабочий каталог

Рис. 15.6. Вводим Email и пароль при запуске Titanium Studio

Теперь откроется окно брандмауэра Windows — поскольку Titanium Studio попытается соединиться с сервером для проверки Email и пароля. Разрешите доступ Titanium Studio к Интернету, иначе среда разработки не будет работать (рис. 15.7).

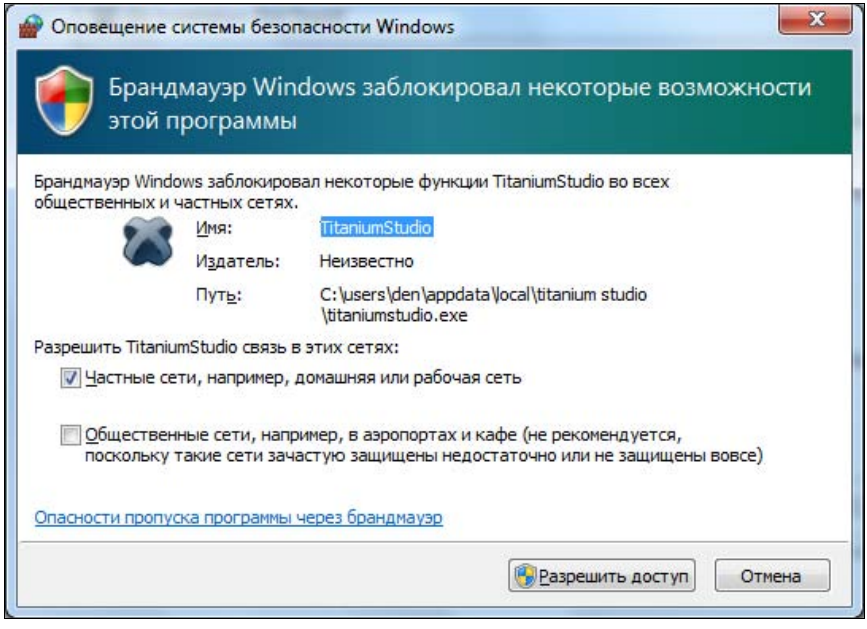


Рис. 15.7. Разрешаем доступ Titanium Studio к Интернету

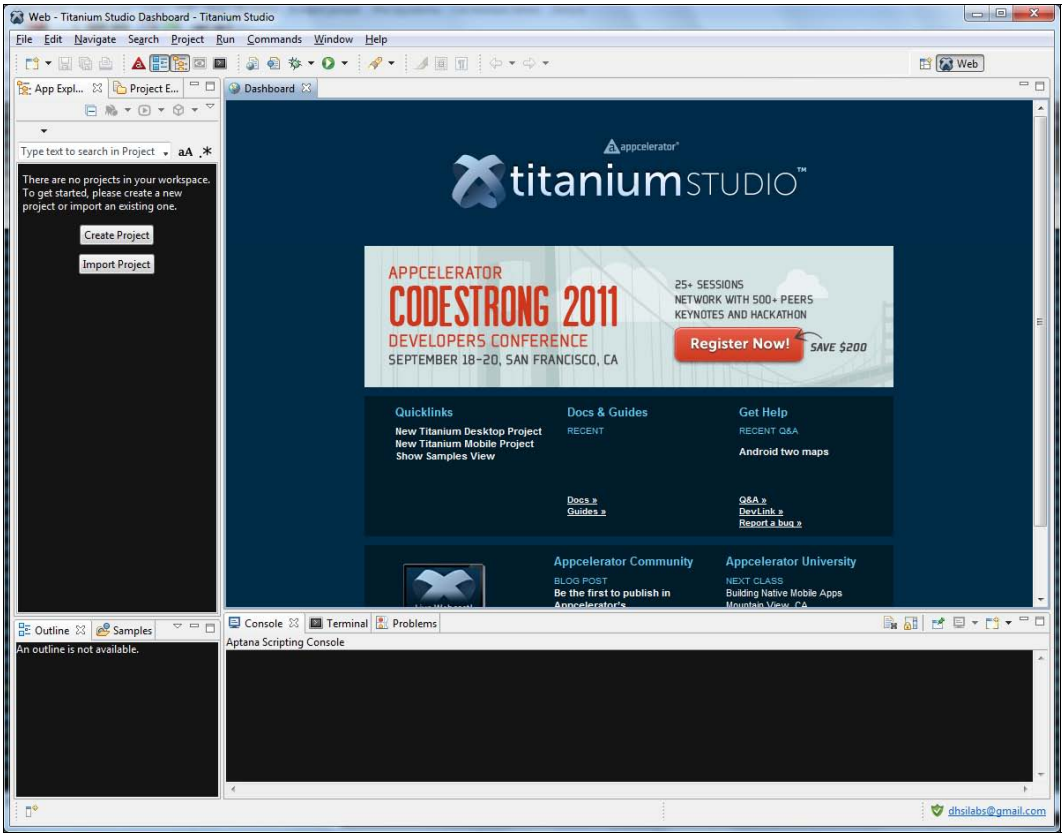


Рис. 15.8. Основное окно Titanium Studio

Ну и наконец, вы можете созерцать основное окно Titanium Studio (рис. 15.8). Вот теперь можно приступать к созданию RIA-приложения для системы Android.

> ***Интерфейс Titanium Studio***
>
> Вам не кажется, что Titanium Studio похожа на Eclipse? Компания Aptana разработала IDE Aptana Studio на базе Eclipse. Затем компания Appcelerator выкупила компанию Aptana и представила новый продукт — Titanium Studio. Отсюда и схожесть с Eclipse, но Android-разработчикам такая схожесть даже на руку — не нужно привыкать к новому интерфейсу. До этого использовалась среда разработки IDE Titanium Developer, которая по возможностям отставала от сегодняшней Titanium Studio и была не очень удобной.

15.3. Создание первого RIA-приложения с помощью Titanium Studio

15.3.1. Создание проекта

Попробуем создать ваше первое и очень простое RIA-приложение для Android. Нажмите кнопку **Create Project**. В открывшемся окне выберите тип проекта — **Titanium Mobile Project** (рис. 15.9).

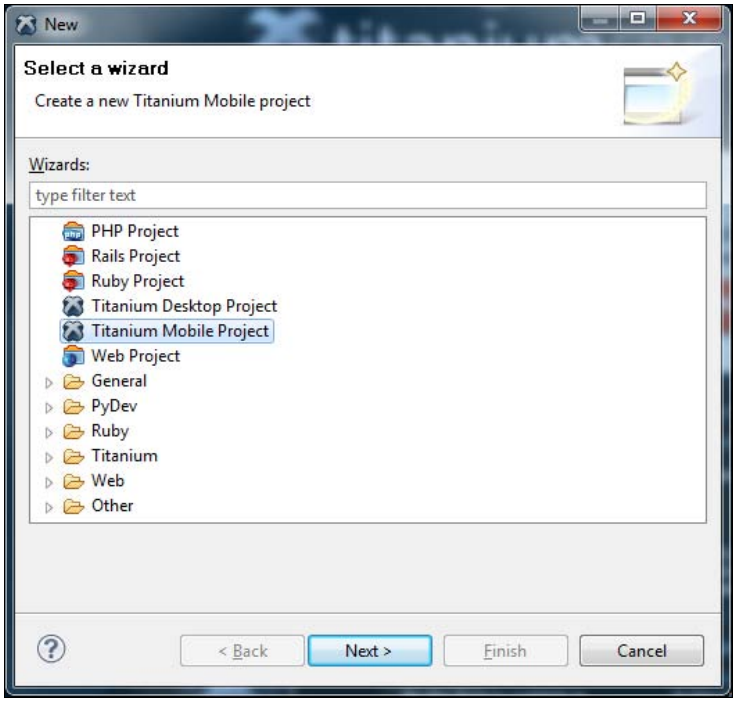


Рис. 15.9. Выбор типа проекта

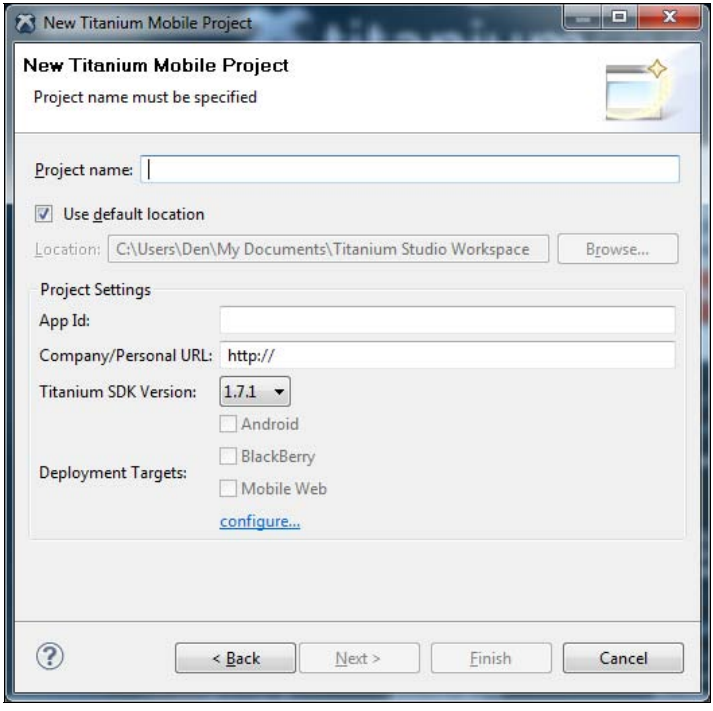


Рис. 15.10. Окно создания нового проекта

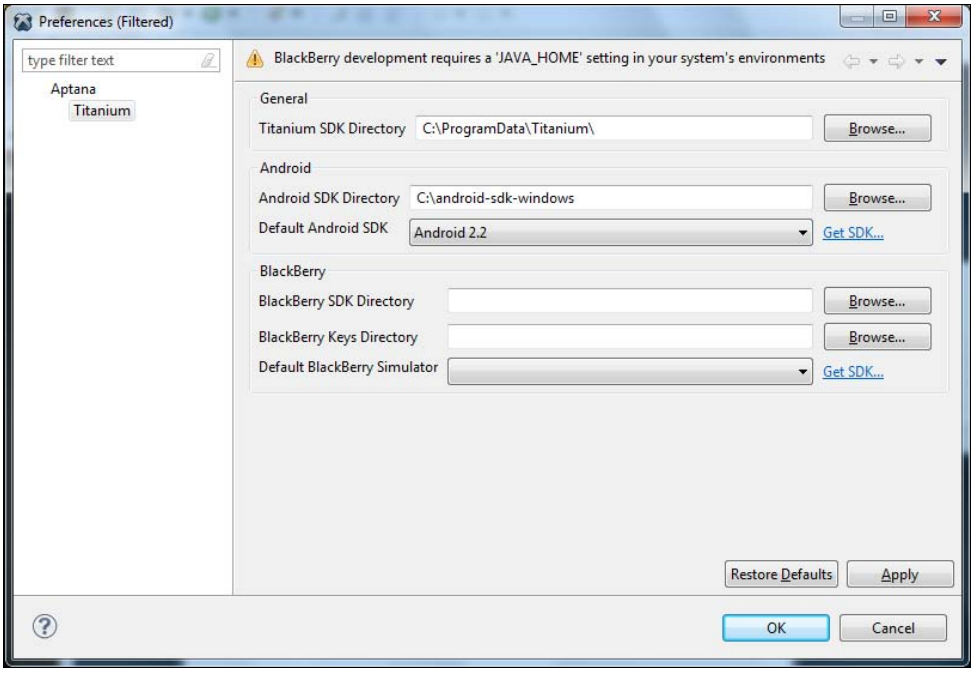


Рис. 15.11. Настройка Titanium Studio

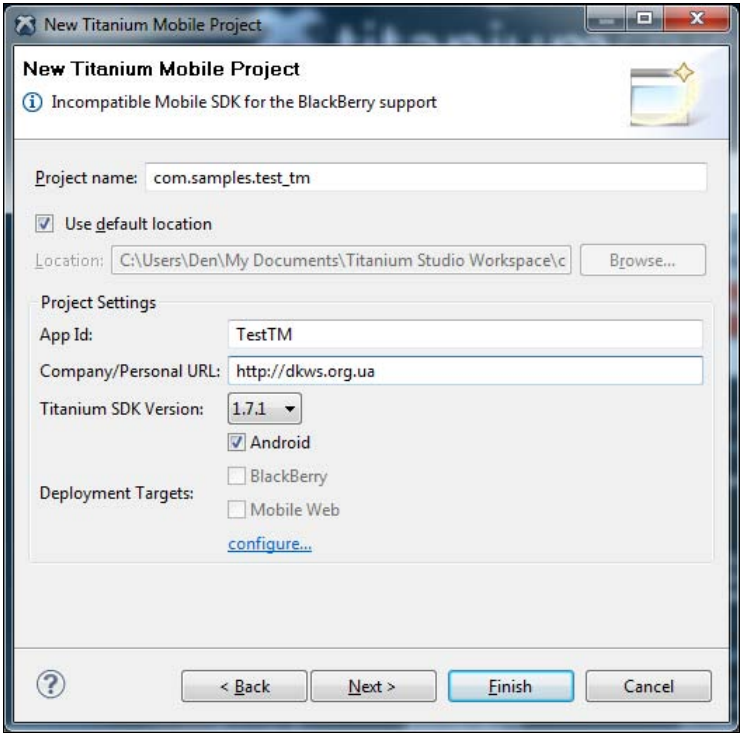


Рис. 15.12. Заполнение окна создания нового проекта

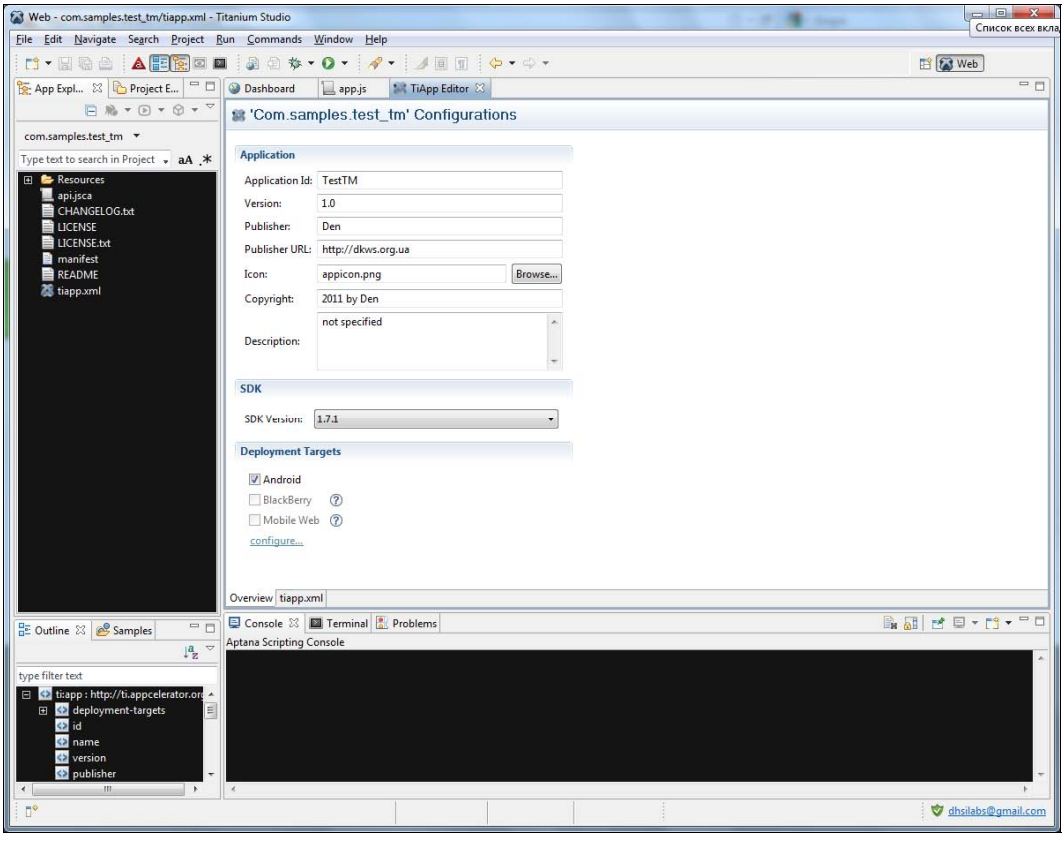


Рис. 15.13. Окно созданного проекта

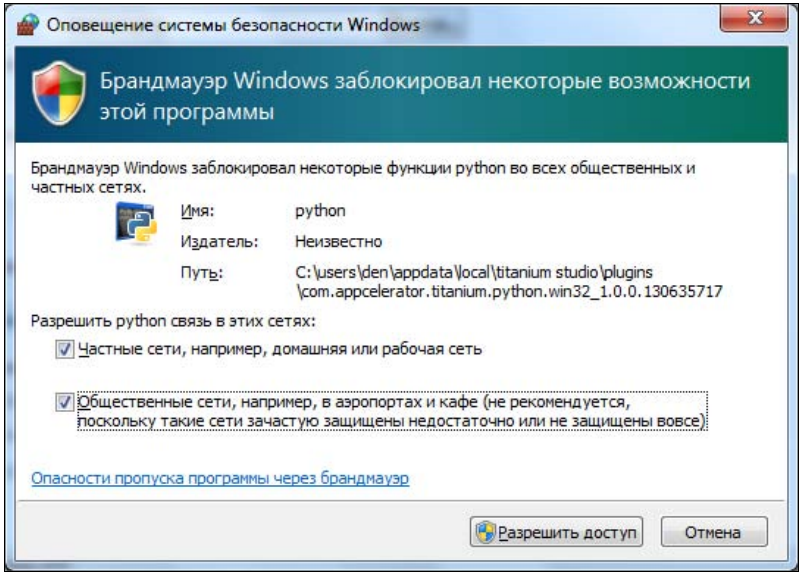


Рис. 15.14. Разрешаем доступ python к Интернету

На следующем шаге нужно заполнить информацией окно, изображенное на рис. 15.10, но сначала укажите путь к Android SDK — для этого нажмите ссылку **configure** и в открывшемся окне (рис. 15.11) укажите путь, нажмите кнопку **Apply**, а затем **OK**.

После этого вы вернетесь в окно создания нового проекта — в нем появится возможность выбора платформы Android. Заполните это окно, как показано на рис. 15.12, и нажмите кнопку **Finish**.

Далее вы увидите окно созданного проекта (рис. 15.13) — нажмите в нем кнопку **Run**. Да, мы просто попытаемся запустить пустой проект. Сразу после нажатия кнопки **Run** откроется окно брандмауэра Windows — нужно разрешить доступ к Интернету интерпретатору python (рис. 15.14).

15.3.2. Установка переменных окружения

Скорее всего, ваше приложение не запустится. Вы получите сообщение об ошибке — Titanium Mobile не может найти JDK, поскольку переменная окружения `JAVA_HOME` не установлена (рис. 15.15).

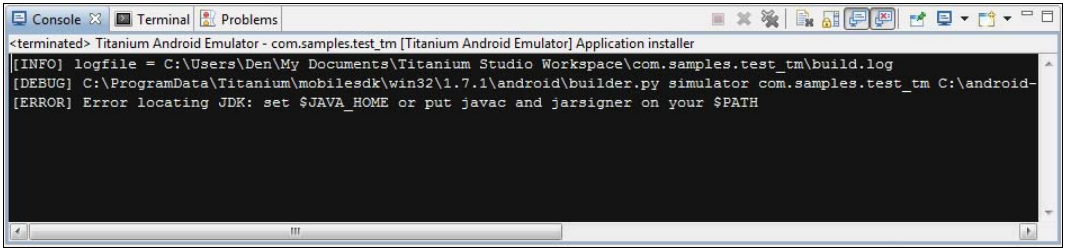


Рис. 15.15. Не установлена переменная окружения `JAVA_HOME`

Сейчас мы это исправим. Первым делом загляните в каталог C:\Program Files и узнайте точное имя каталога, в который установлена JDK. В моем случае каталог назывался так: C:\Program Files\Java\jdk1.6.0_26.

Откройте окно **Компьютер** и нажмите кнопку **Свойства системы**. В открывшемся окне перейдите на вкладку **Дополнительно** (рис. 15.16). Нажмите кнопку **Переменные среды**. Добавьте системную переменную `JAVA_HOME`, значение переменной — путь к JDK (рис. 15.17).

Также нужно изменить переменную `Path` — к ней надо добавить путь к JDK и Android SDK. Выделите переменную, нажмите кнопку **Изменить** и добавьте к значению оба пути, разделяя их точками с запятой. Должно получиться примерно следующее значение (у вас оно будет другим, поскольку путь зависит от установленных программ):

%SystemRoot%\system32;%SystemRoot%;%SystemRoot%\System32\
Wbem;%SystemRoot%\System32\WindowsPowerShell\v1.0\;C:\Program Files\
Samsung\Samsung PC Studio 3\;c:\Program Files\Java\jdk1.6.0_26\bin;
c:\ android-sdk-windows\

Дважды нажмите кнопку **OK**.

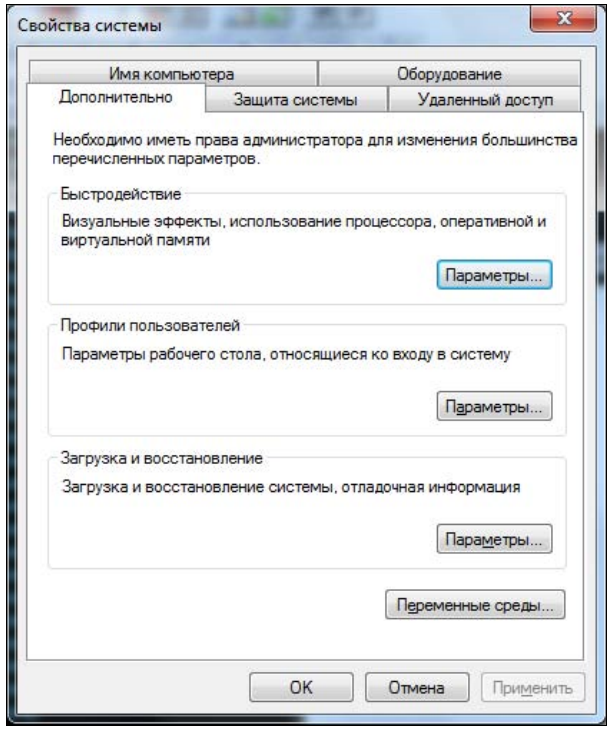


Рис. 15.16. Окно **Свойства системы**

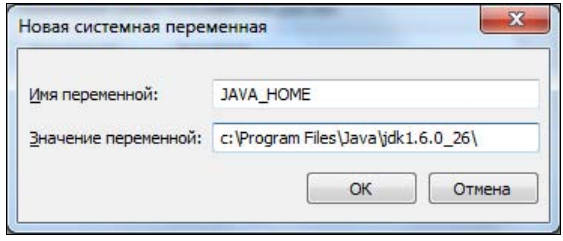


Рис. 15.17. Создание новой системной переменной

После изменения системных переменных перезагрузите Titanium Studio (чтобы процесс перечитал переменную `JAVA_HOME`). Ваш последний проект будет автоматически загружен, поэтому сразу нажмите кнопку **Run**. Запустится эмулятор Android и в него будет загружено созданное приложение (рис. 15.18).

15.3.3. Ситуация: компилятор javac не найден

Обратите внимание на путь к JDK, который вы добавили к переменной `Path`. Если вы указали не **c:\Program Files\Java\jdk1.6.0_26\bin** (я специально выделил здесь этот путь полужирным), а просто: c:\Program Files\Java\jdk1.6.0_26\ (как в случае с переменной `JAVA_HOME`), то Titanium Studio не найдет необходимый для компиляции программы компилятор javac, который находится в каталоге bin. А проявляться это

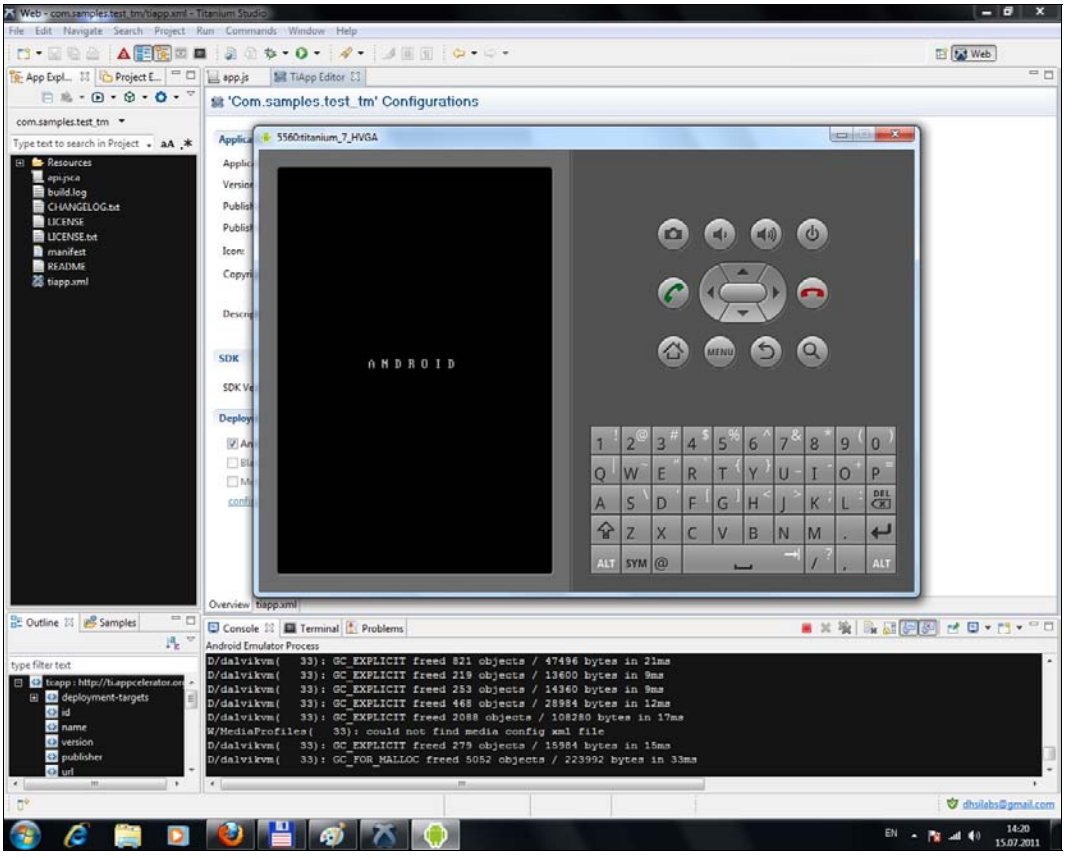


Рис. 15.18. Запуск приложения из Titanium Studio

будет в следующем: эмулятор Android по нажатию кнопки **Run** запустится подозрительно быстро (не через 5–6 минут, а через 1 минуту максимум), но в него не будет загружено наше Titanium-приложение.

Тогда откройте файл build.log, который находится в рабочем каталоге вашего проекта: <каталог рабочего пространства>\<имя проекта>. Если javac не найден, то в этом файле вы найдете сообщение о том, что команда javac не найдена, и размер этого файла будет около 500 байтов. А вот если все прошло нормально, то файл build.log займет объем порядка нескольких килобайтов, и в нем будут прописаны все подробности сборки проекта. В листинге 15.1 приводятся начальные строки файла build.log для нашего тестового проекта, если никаких проблем не возникло.

Листинг 15.1. Фрагмент файла build.log

```
Logfile initialized
[INFO] logfile = C:\Users\Den\My Documents\Titanium Studio
Workspace\com.samples.test_tm\build.log
[DEBUG] C:\ProgramData\Titanium\mobilesdk\win32\1.7.1\android\builder.py
simulator com.samples.test_tm C:\android-sdk-windows C:\Users\Den\My
Documents\Titanium Studio Workspace\com.samples.test_tm TestTM 7 HVGA
```

```
[INFO] Building com.samples.test_tm for Android ... one moment
[INFO] Titanium SDK version: 1.7.1 (06/21/11 14:28 1293a6d)
[DEBUG] Waiting for device to be ready ...
[TRACE] adb devices returned 1 devices/emulators
[TRACE] adb devices returned 1 devices/emulators
[TRACE] adb devices returned 1 devices/emulators
[TRACE] adb devices returned 1 devices/emulators
[TRACE] adb devices returned 1 devices/emulators
[TRACE] adb devices returned 1 devices/emulators
[TRACE] adb devices returned 1 devices/emulators
[DEBUG] Device connected... (waited 30 seconds)
[DEBUG] waited 30.237000 seconds on emulator to get ready
[INFO] Waiting for the Android Emulator to become available
[ERROR] Timed out waiting for android.process.acore
[DEBUG] TestTM installed? False
[WARN] Fastdev enabled, but server isn't running, deploying normally
[INFO] Copying project resources..
[INFO] Detected tiapp.xml change, forcing full re-build...
[TRACE] COPYING CREATED FILE: C:\Users\Den\My Documents\Titanium Studio
Workspace\com.samples.test_tm\Resources\android\appicon.png => C:\Users\Den\My
Documents\Titanium Studio
Workspace\com.samples.test_tm\build\android\bin\assets\Resources\appicon.png
```

15.3.4. Ошибка: Error generating R.java from manifest

В файле build.log вы можете обнаружить сообщение:

[ERROR] Error generating R.java from manifest

Причина, скорее всего, связана с именем проекта. Имя проекта должно быть в формате: `com.имя_компании.имя_приложения`.

Неправильный формат имени проекта — наиболее вероятная причина такой ошибки. С другими причинами вы можете ознакомиться на форуме разработчиков Titanium: **http://developer.appcelerator.com**.

15.3.5. Что дальше?

Мы только что рассмотрели основные этапы начала разработки RIA-приложений для Android: загрузку, установку и настройку Titanium Studio, а также запуск тестового проекта. Как выяснилось, нажать кнопку **Run** недостаточно...

Что дальше? К сожалению, платформа Titanium Mobile не будет описана в этой книге полностью, поскольку она заслуживает отдельной книги, и рассмотрение ее в двух-трех главах было бы настоящим кощунством с моей стороны. Если вы заинтересованы в создании RIA-приложений, тогда начало уже положено — у вас на компьютере есть рабочая Titanium Studio, готовая к разработке приложений. Далее советую изучить руководство разработчика — без него никак:

http://developer.appcelerator.com/doc/mobile/guides

ГЛАВА 16

Взаимодействие с аппаратными средствами

16.1. Получение информации об устройстве

Уверен, что эта глава будет интересна большинству читателей, поскольку в ней мы поговорим о программировании аппаратных средств вашего устройства (смартфона или планшета). Сначала мы узнаем, как с помощью класса `TelephonyManager` получить информацию об устройстве, определить его состояние и набрать телефонный номер абонента. Потом мы рассмотрим, как использовать датчики (сенсоры), камеру и Bluetooth-адаптер вашего устройства. В отличие от других глав, в этой главе желательно тестировать программы не в эмуляторе, а на физическом устройстве — настоящем смартфоне/планшете, поскольку возможности эмулятора ограниченны, и полностью протестировать программы из этой главы в нем не получится.

Начнем мы с получения информации об устройстве. Для этого мы воспользуемся классом `TelephonyManager`. Прежде чем приступить к написанию кода, нужно добавить в файл манифеста следующую строку (рис. 16.1) — она разрешает приложению читать состояние устройства:

```
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

Файл разметки проекта по умолчанию подойдет для нашего приложения почти без изменений. Нужно только добавить свойство `id` для элемента `TextView` (листинг 16.1).

Листинг 16.1. Файл разметки TM/res/layout/main.xml для проекта TM

```
<?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:id="@+id/info"
    android:layout_width="fill_parent"
```

```
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
</LinearLayout>
```

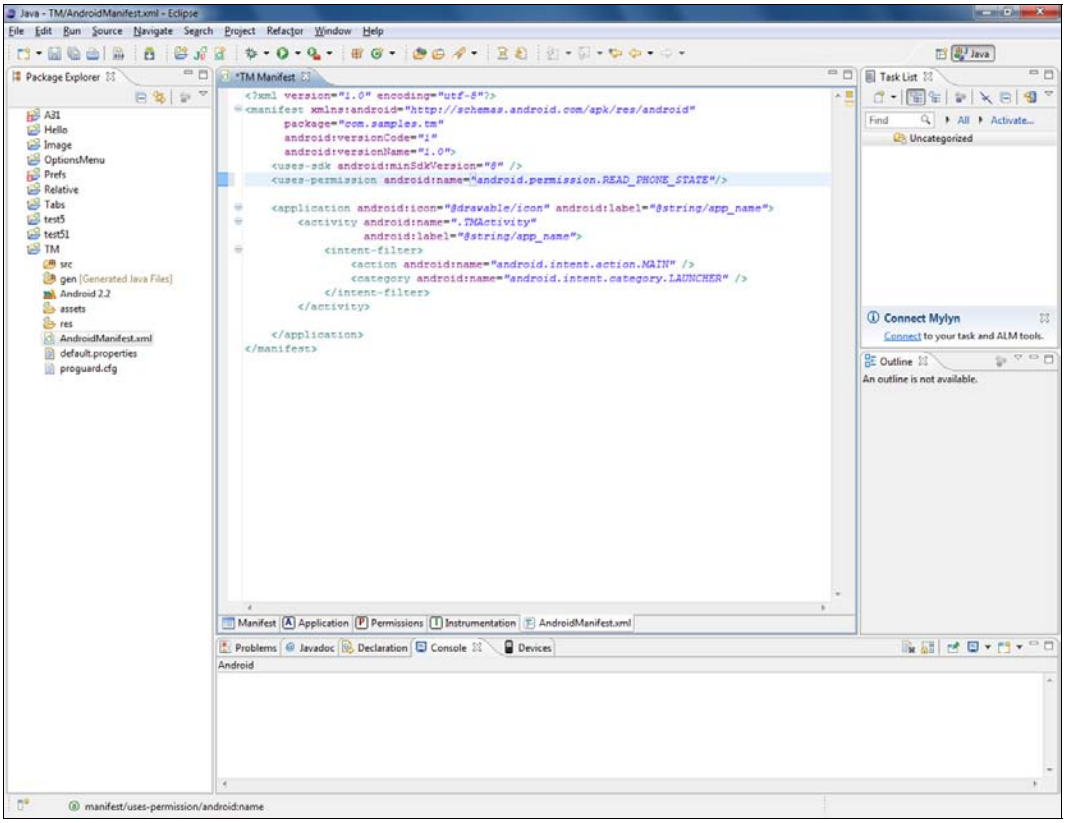


Рис. 16.1. Редактирование файла манифеста приложения

Java-код приложения с комментариями приведен в листинге 16.2.

Листинг 16.2. Java-код приложения TM

```
package com.samples.tm;

import android.app.Activity;
import android.os.Bundle;
import android.telephony.TelephonyManager;
import android.widget.TextView;

public class TMActivity extends Activity {
       TextView info;              // Текстовая область TextView
       TelephonyManager tm;        // для информации об устройстве
```

```
@Override
public void onCreate(Bundle savedInstanceState) {
      String EOL = "\n";

      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

      // Находим текстовую область в разметке
      info =(TextView) findViewById(R.id.info);
      // Создаем объект tm для получения информации об устройстве
      tm = (TelephonyManager)getSystemService(TELEPHONY_SERVICE);
      // Буфер строк
      StringBuilder sb = new StringBuilder();

      // Общая информация об устройстве
      sb.append("Общая информация:\n\n");
      sb.append("ID устройства :")
      .append(tm.getDeviceId()).append(EOL);
      sb.append("Версия ПО: ")
      .append(tm.getDeviceSoftwareVersion()).append(EOL);
      sb.append("Номер телефона: ")
      .append(tm.getLine1Number()).append(EOL);

      // Информация об операторе, выдавшем SIM-карту
      sb.append("\nОператор:\n\n");
      sb.append("Код страны (ISO): ")
      .append(tm.getSimCountryIso()).append(EOL);
      sb.append("Оператор: ")
      .append(tm.getSimOperator()).append(EOL);
      sb.append("Название оператора: ")
      .append(tm.getSimOperatorName()).append(EOL);
      sb.append("Серийный номер SIM-карты: ")
      .append(tm.getSimSerialNumber()).append(EOL);

      // Информация о текущей сети
      sb.append("\nСеть:\n\n");
      sb.append("Код страны (ISO): ")
      .append(tm.getNetworkCountryIso()).append(EOL);
      sb.append("Оператор сети: ")
      .append(tm.getNetworkOperator()).append(EOL);
      sb.append("Название оператора сети: ")
      .append(tm.getNetworkOperatorName()).append(EOL);

      // Голосовая почта и другая информация
      sb.append("\nДругая информация:\n\n");
      sb.append("ID подписчика: ")
      .append(tm.getSubscriberId()).append(EOL);
```

```
        sb.append("Альфа-тег голосовой почты: ")
        .append(tm.getVoiceMailAlphaTag()).append(EOL);
        sb.append("Номер голосового почтового ящика: ")
        .append(tm.getVoiceMailNumber()).append(EOL);

        // Выводим содержимое буфера строк в текстовую область
        info.setText(sb.toString());
        }
}
```

Результат работы нашего приложения приведен на рис. 16.2.

Рис. 16.2. Полная информация об устройстве

16.2. Прослушивание состояния устройства

Иногда нужно выполнить какие-то действия только при определенном состоянии устройства — скажем, при входящем звонке или при завершении звонка. Например, вы хотите создать программу записи входящих (или исходящих) звонков. Тогда вам нужно прослушивать состояние устройства — как только будет обнаружен входящий (исходящий) звонок, вам надо начать запись разговора. Кстати, именно такую программу мы создавать не будем, поскольку есть уже готовая бесплатная программа vRecorder. Зато мы разберемся, как прослушать состояние устройства, а при желании вы сможете самостоятельно написать собственный аналог vRecorder.

Для прослушивания состояния устройства с целью ожидания какого-либо события используются «прослушки», подробно описанные на странице руководства разработчика Android:

http://developer.android.com/reference/android/telephony/PhoneStateListener.html

Мы рассмотрим только наиболее часто используемую «прослушку» `PhoneStateListener.LISTEN_CALL_STATE`, позволяющую определить номер входящего звонка (и, соответственно, выполнить определенные действия при входящем звонке). Возможны три состояния звонка:

- `CALL_STATE_IDLE` — устройство не используется для телефонного звонка (не принимается входящий звонок и не устанавливается исходящий);
- `CALL_STATE_RINGING` — устройство принимает входящий звонок;
- `CALL_STATE_OFFHOOK` — пользователь разговаривает с абонентом.

Сейчас мы напишем программу, которая лишь реагирует на все три состояния звонка и более ничего не делает, — требуемые действия вы сможете определить сами. Такое решение принято, чтобы не захламлять код. А что делать, решать вам — можно, например, при получении звонка вывести соответствующее уведомление. Чтобы вы могли определить собственные действия при изменении состояния звонка, мы переопределим метод `onCallStateChanged()`. Код нашей программы приведен в листинге 16.3.

Листинг 16.3. Реакция на изменение состояния звонка

```
package com.samples.callstate;

import android.app.Activity;
import android.os.Bundle;
import android.telephony.PhoneStateListener;
import android.telephony.TelephonyManager;
import android.widget.TextView;

public class HardwareTelephony extends Activity {
TextView info;              // Сюда можно выводить информацию о звонке
TelephonyManager tm;

@Override
public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.main);

       info =(TextView) findViewById(R.id.info);
       // Создаем объект класса TelephonyManager
       tm = (TelephonyManager)getSystemService(TELEPHONY_SERVICE);
       // Устанавливаем "прослушку" для LISTEN_CALL_STATE
       tm.listen(new TelListener(),PhoneStateListener.LISTEN_CALL_STATE);
}
```

```
private class TelListener extends PhoneStateListener {
       public void onCallStateChanged(int state, String incomingNumber) {
              super.onCallStateChanged(state, incomingNumber);
       switch (state) {
              case TelephonyManager.CALL_STATE_IDLE:
                info.setText("IDLE");
                break;
              case TelephonyManager.CALL_STATE_OFFHOOK:
                info.SetText("OFFHOOK, Вход. номер:" +incomingNumber);
                break;
              case TelephonyManager.CALL_STATE_RINGING:
                info.SetText("RINGING, Вход. номер:" +incomingNumber);
                break;
              default:
                break;
         } // switch
     } // onCallStateChanged
   }
}
```

Наше приложение выводит в текстовую область (`TextView`) с именем `info` состояние устройства и номер входящего звонка, если таковой имеется. Файл разметки для этого приложения будет таким же, как и для предыдущего (см. листинг 16.1). Чтобы приложение работало корректно, в файл манифеста (см. рис. 16.1) нужно добавить следующую строку:

```
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

16.3. Набор номера

Только что мы узнали, как определить номер входящего звонка. А сейчас разберемся, как набрать номер, т. е. создать исходящий звонок.

Первым делом в файл манифеста нужно добавить разрешение на исходящий звонок:

```
<uses-permission android:name="android.permission.CALL_PHONE" />
```

Далее вы можете использовать одну из операций: или `ACTION_CALL`, или `ACTION_DIAL`. Первая операция отобразит диалоговое окно с набираемым номером (как при обычном наборе номера вручную), вторая — наберет номер без показа какого-либо интерфейса пользователя. Пример использования этих двух операций:

```
startActivity(new Intent(Intent.ACTION_CALL, Uri.parse("tel:номер")));
startActivity(new Intent(Intent.ACTION_DIAL, Uri.parse("tel:номер")));
```

16.4. Виброзвонок

Иногда для привлечения внимания к уведомлению или диалоговому окну программы нужно использовать вибрацию. Для управления виброзвонком надо добавить в файл манифеста следующее разрешение:

```
<uses-permission android:name="android.permission.VIBRATE" />
```

Далее следует использовать класс `Vibrator`:

```
Vibrator Vib = (Vibrator) getSystemService(Context.VIBRATOR_SERVICE);
Vib.vibrate(3000); // Вибрировать 3 секунды
```

Для преждевременной отмены вибрации (например, если вы установили вибрацию 3 секунды, а пользователь отреагировал раньше) служит метод `cancel()`:

```
Vib.cancel();
```

16.5. Датчики мобильного устройства

Современный мобильный телефон давно превратился в сложное электронное устройство, выполняющее в том числе и функции телефона. Да, именно так. Прежде у мобильного телефона существовала одна основная функция — телефония, второстепенной функцией была возможность отправки и приема текстовых сообщений (SMS). Сейчас же сложно назвать, какая функция мобильного телефона является основной. Да и сами телефоны стали называться иначе — смартфонами. Большинство пользователей покупают смартфоны именно из-за функций, которые для обычного телефона являются необязательными: доступа к Интернету, GPS-навигации и т. п.

Чтобы смартфон был конкурентоспособным, он должен предоставлять много разных функций, для реализации которых необходимы всевозможные датчики (сенсоры, от англ. sensor).

Наиболее популярным сенсором является камера, но управление ею заслуживает отдельного разговора, — мы обязательно разберемся с управлением камерой, но чуть позже. А пока рассмотрим другие датчики, которыми может быть оснащено мобильное устройство:

- акселерометр (`TYPE_ACCELEROMETER`) — позволяет определить ускорение мобильного устройства при его перемещении. Таким датчиком оснащаются не все смартфоны, преимущественно акселерометр можно найти на смартфонах, оснащенных функцией GPS (хотя это необязательно — все зависит от производителя устройства);
- датчик света (`TYPE_LIGHT`) — очень полезная штука. С его помощью можно управлять подсветкой дисплея — например, увеличить яркость, когда стало темно. В конечном счете такой датчик помогает экономить заряд аккумулятора;
- температурный датчик (`TYPE_TEMPERATURE`);
- датчик атмосферного давления (`TYPE_PRESSURE`).

В вашем устройстве могут иметься и дополнительные датчики. Получить список датчиков можно методом `getSensorList()` класса `SensorManager`.

В этом разделе мы рассмотрим чтение показаний датчика температуры. Примеры чтения других датчиков вы можете найти в документации разработчика Android.

Для чтения датчиков нужно подключить следующие пакеты:

```
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
```

Далее определить объекты класса `SensorManager`:

```
private SensorManager myManager = null;
myManager = (SensorManager)getSystemService(SENSOR_SERVICE);
myManager.registerListener(tempSensorListener,
myManager.getDefaultSensor(Sensor.TYPE_TEMPERATURE),
SensorManager.SENSOR_DELAY_GAME);
...
```

Методу `registerListener()` нужно передать три параметра. Первый — название обработчика датчика температуры. В нашем случае — это `tempListener`, который будет определен позже. Второй параметр — датчик по умолчанию, в нашем случае — датчик температуры. Третий параметр задает время обновления показаний датчика. Для наиболее быстрого обновления используйте `SENSOR_DELAY_GAME`, для обычного обновления — `SENSOR_DELAY_NORMAL`.

Следующий код определяет «прослушку» `tempListener`. Наша задача — переопределить методы `onAccuracyChanged()` и `onSensorChanged()`. Первый метод нам не нужен, поэтому мы определим его как пустой метод. А второй метод будет устанавливать текст области `info` (элемент `TextView` в разметке) — в ней мы станем показывать температуру:

```
private final SensorEventListener tempListener = new SensorEventListener(){

       @Override
       public void onAccuracyChanged(Sensor sensor, int accuracy) {}

       @Override
       public void onSensorChanged(SensorEvent event) {
              if(event.sensor.getType()==Sensor.TYPE_TEMPERATURE){
                info.setText("Температура: "+event.values[0]);
               }
       }
};
```

Дополнительную информацию можно получить по адресу:

http://developer.android.com/reference/android/hardware/SensorManager.html

16.6. Доступ к камере

Есть два способа доступа к камере: первый заключается в вызове стандартного интерфейса управления камерой, второй — в использовании класса `Camera`. Все зависит от того, что вам нужно. Если просто отобразить интерфейс управления камерой, чтобы пользователь мог создать фотографию (потом он может выбрать файл с только что созданным изображением и что-то с ним сотворить), тогда вам лучше выбрать первый способ. Второй способ подойдет, если вы сразу хотите заполучить снимок, созданный камерой. Понятно, что второй способ более удобен для пользователя, но более сложен для реализации программистом.

Начнем с первого способа. Для отображения стандартного интерфейса управления камерой служит следующий код:

```
Intent intent = new Intent("android.media.action.IMAGE_CAPTURE");
startActivity(intent);
```

Просто? Да. Представим, что вы создаете графический редактор. При запуске редактор просит указать имя файла на SD-карте. При этом у пользователя есть две кнопки: **Камера** и **Файл**. При нажатии первой кнопки появляется интерфейс получения снимка, при нажатии второй — диалоговое окно выбора файла. Далее все зависит от нашей программы. Простой графический редактор по нажатию кнопки **Камера** вызовет интерфейс управления камерой и позволит пользователю сделать снимок. После закрытия интерфейса получения снимка пользователю придется нажать кнопку **Файл** и выбрать для дальнейшей обработки только что созданный файл. Сложный графический редактор сразу приступит к обработке созданного снимка.

Почему диалоговое окно выбора файла не было описано в *главе 6*? Да потому что изначально в Android... нет такого диалогового окна. Вот, казалось бы — стандартная вещь, но ее приходится создавать отдельно.

К счастью для программиста, диалоговое окно выбора файла уже создано Александром Пономаревым. Получить информацию и скачать код этого диалогового окна можно по адресам:

http://code.google.com/p/android-file-dialog/

http://code.google.com/p/android-file-dialog/source/browse/trunk/FileExplorer/?r=3

А мы же вернемся к работе с камерой. Если вы — настоящий программист, вас интересует второй, более сложный способ. Перед использованием класса `Camera` нужно добавить в файл манифеста соответствующее разрешение:

```
<uses-permission android:name="android.permission.CAMERA" />
```

Для работы с камерой нам понадобятся следующие классы:

- `Camera` — доступ к «железу» (непосредственно к самой камере);
- `Camera.Parameters` — позволяет указать параметры камеры, например: размер изображения, его качество и т. д.;

- `SurfaceView` — позволяет выделить поверхность, которая будет использоваться в качестве области предварительного просмотра для камеры.

Создайте новый проект `MyCamera` (имя пакета `com.samples.mycamera`). Основной файл разметки res/layout/main.xml для нашего проекта приведен в листинге 16.4. Элемент `SurfaceView` будет использоваться в качестве области предварительного просмотра камеры.

Листинг 16.4. Основной файл разметки main.xml

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
       android:layout_width="fill_parent"
       android:layout_height="fill_parent"
       android:orientation="vertical">
<SurfaceView android:id="@+id/surface"
       android:layout_width="fill_parent"
       android:layout_height="fill_parent">
</SurfaceView>
</LinearLayout>
```

Управляющий камерой интерфейс будет описан в отдельном файле разметки. Назовите его cameraui.xml и поместите в каталог res/layout. Код этого файла приведен в листинге 16.5.

Листинг 16.5. Файл cameraui.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
       android:layout_width="fill_parent"
       android:layout_height="fill_parent"
       android:orientation="vertical"
       android:gravity="bottom"
       android:layout_gravity="bottom">
<LinearLayout
       xmlns:android="http://schemas.android.com/apk/res/android"
       android:layout_width="fill_parent"
       android:layout_height="wrap_content"
       android:orientation="horizontal"
       android:gravity="center_horizontal">
<Button
       android:id="@+id/button"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="Получить фото"
/>
</LinearLayout>
</LinearLayout>
```

Теперь приступаем к коду. Метод `takePicture()`, используемый для получения фото, требует указания трех методов:

- `SurfaceCallback()` — служит для управления поверхностью, может также использоваться для применения различных графических эффектов к полученному фото;
- `PictureCallback()` — служит для управления памятью: что делать с фото, если не хватило аппаратной памяти;
- `CompressionCallback()` — служит для сжатия фото.

Для управления поверхностью (surface) используется интерфейс `SurfaceHolder.Callback`. Нам нужно переопределить три его метода:

- `surfaceCreated()` — вызывается во время создания поверхности, применяется для инициализации объектов;
- `surfaceChanged()` — вызывается после создания поверхности и когда изменены параметры поверхности — например, ее размер;
- `surfaceDestroyed()` — вызывается при удалении поверхности, служит для очистки памяти.

Полный код приложения с комментариями приведен в листинге 16.6.

Листинг 16.6. Java-код проекта `MyCamera`

```
package com.samples.mycamera;

import android.app.Activity;
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.PixelFormat;
import android.os.Bundle;
import android.provider.MediaStore.Images;
import android.view.LayoutInflater;
import android.view.View;
import android.view.Window;
import android.view.WindowManager;
import android.view.View.OnClickListener;
import android.view.ViewGroup.LayoutParams;
import android.widget.Button;
import android.widget.Toast;
// Подключаем самые необходимые пакеты
import android.hardware.Camera;
import android.hardware.Camera.PictureCallback;
import android.hardware.Camera.ShutterCallback;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
```

```
public class MyCamera extends Activity implements SurfaceHolder.Callback {
   private LayoutInflater mInflater = null;
   Camera MyCam;                    // Камера
   byte[] tempdata;                 // Массив для временных данных
   boolean mPreviewRunning = false;
   // Поверхность и владелец поверхности (SurfaceHolder)
   private SurfaceHolder PreviewHolder;
   private SurfaceView Preview;
   // Кнопка
   Button GetPicture;

   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       getWindow().setFormat(PixelFormat.TRANSLUCENT);
       requestWindowFeature(Window.FEATURE_NO_TITLE);
       getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
       WindowManager.LayoutParams.FLAG_FULLSCREEN);
       setContentView(R.layout.main);
       // Находим поверхность и устанавливаем SurfaceHolder
       Preview = (SurfaceView)findViewById(R.id.surface);
       PreviewHolder = Preview.getHolder();
       PreviewHolder.addCallback(this);
       PreviewHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
       mInflater = LayoutInflater.from(this);
       View overView = mInflater.inflate(R.layout.cameraoverlay, null);
       this.addContentView(overView,
                           new LayoutParams(LayoutParams.FILL_PARENT,
                           LayoutParams.FILL_PARENT));
       // Находим кнопку
       GetPicture = (Button) findViewById(R.id.button);
       // Устанавливаем обработчик нажатия кнопки
       GetPicture.setOnClickListener(new OnClickListener(){
       public void onClick(View view){
       // Вызываем метод takePicture() для получения картинки
              MyCam.takePicture(Shutter, PicCallback, compress);
       }
    });
}

// Пустая заглушка — ничего не делаем
ShutterCallback Shutter = new ShutterCallback(){
       @Override
       public void onShutter() {}
};
```

```
// Пустая заглушка — ничего не делаем
PictureCallback PicCallback = new PictureCallback() {
       public void onPictureTaken(byte[] data, Camera c) {}
};

// Заполняем массив с временными данными и вызываем
// функцию done()
PictureCallback compress = new PictureCallback() {
       public void onPictureTaken(byte[] data, Camera c) {
              if(data !=null) {
                     tempdata=data;
                     done();
              }
       }
};

void done() {
       // Получаем растровое изображение путем декодирования
       // массива tempdata
       Bitmap bm = BitmapFactory.decodeByteArray(tempdata,
                                          0, tempdata.length);
       String url = Images.Media.insertImage(getContentResolver(),
                                                 bm, null, null);
       bm.recycle();
       Bundle bundle = new Bundle();
       if(url!=null) {
              bundle.putString("url", url);
              Intent mIntent = new Intent();
              mIntent.putExtras(bundle);
              setResult(RESULT_OK, mIntent);
       } else {
              Toast.makeText(this, "Ошибка получения картинки",
              Toast.LENGTH_SHORT).show();
       }
       finish();
}

// Реакция на изменение поверхности
@Override
public void surfaceChanged(SurfaceHolder holder, int format,
int w, int h) {
       try {
              if (mPreviewRunning) {
                     MyCam.stopPreview();
                     mPreviewRunning = false;
              }
       // Получаем параметры камеры
       Camera.Parameters p = MyCam.getParameters();
```

```
        // Устанавливаем размер пред. просмотра
        p.setPreviewSize(w, h);
        // Устанавливаем параметры камеры
        MyCam.setParameters(p);
        // Устанавливаем владельца поверхности
        MyCam.setPreviewDisplay(holder);
        // Запускаем пред. просмотр
        MyCam.startPreview();
        // Флаг пред. просмотра
        mPreviewRunning = true;
        } catch(Exception e) {
        // Действие в случае исключения, для упрощения кода
        // оставлено незаполненным. Можно вывести номер
        // ошибки как уведомление с помощью метода
        // e.toString()
        }
}

@Override
public void surfaceCreated(SurfaceHolder holder) {
// Действие при создании поверхности — открываем камеру
        MyCam = Camera.open();
}
@Override
public void surfaceDestroyed(SurfaceHolder holder) {
        // Останавливаем пред. просмотр
        MyCam.stopPreview();
        // Сбрасываем флаг
        mPreviewRunning = false;
        // Освобождаем ресурсы
        MyCam.release();
        MyCam=null;
    }
}
```

16.7. Bluetooth

Думаю, не найдется ни одного читателя этой книги, который бы не знал, что такое Bluetooth. Именно поэтому здесь вы не найдете описания этой технологии — все-таки книга для программистов, а не для начинающих пользователей. Предлагаю сразу перейти к программированию.

Для передачи данных с использованием Bluetooth нужно:

1. Включить адаптер Bluetooth.
2. Найти доступные устройства (другие мобильные устройства с включенным Bluetooth-адаптером).

3. Подключиться к одному из устройств.
4. Произвести собственно обмен данными.

В файл манифеста Android-приложения, использующего Bluetooth, нужно добавить строки:

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
```

В пакете `android.bluetooth` определены следующие классы:

- `BluetoothAdapter` — представляет интерфейс обнаружения и установки Bluetooth-соединений;
- `BluetoothClass` — описывает общие характеристики Bluetooth-устройства;
- `BluetoothDevice` — представляет удаленное Bluetooth-устройство;
- `BluetoothSocket` — сокет или точка соединения для данных, которыми наша система обменивается с другим Bluetooth-устройством;
- `BluetoothServerSocket` — сокет для прослушивания входящих Bluetooth-соединений.

16.7.1. Включение Bluetooth-адаптера

Первым делом нужно получить адаптер по умолчанию:

```
BluetoothAdapter myBluetooth = BluetoothAdapter.getDefaultAdapter();
```

Активировать Bluetooth-адаптер можно с помощью следующего кода:

```
// Если Bluetooth-выключен
if(!myBluetooth.isEnabled()) {
// Создаем действие ACTION_REQUEST_ENABLE — запрашивает включение
// адаптера
Intent eIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
// Выполняем действие
startActivity(eIntent);
}
```

16.7.2. Обнаружение соседних устройств

Для обнаружения соседних устройств служит код, приведенный в листинге 16.7. Обнаруженные устройства попросту выводятся в журнал с помощью `Log.d()`.

Листинг 16.7. Поиск Bluetooth-устройств

```
import android.util.Log;
...
private final BroadcastReceiver myReceiver = new BroadcastReceiver() {
   public void onReceive(Context context, Intent intent) {
       String action = intent.getAction();
```

```
        // Когда найдено устройство
        if (BluetoothDevice.ACTION_FOUND.equals(action)) {
                // Получаем объект BluetoothDevice из Intent
                BluetoothDevice device = intent.getParcelableExtra(
                        BluetoothDevice.EXTRA_DEVICE);
                // Выводим сообщение в журнал (его можно будет просмотреть
                // в Eclipse при запуске приложения).
                Log.v("BlueTooth Discovery: ",device.getName() + "\n"
                + device.getAddress());
        }
    }
};
IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
registerReceiver(myReceiver, filter);
myBluetooth.startDiscovery();
```

16.7.3. Установка соединения с Bluetooth-устройством

Вы можете разработать как приложение-сервер, ожидающее входящих запросов, так и приложение-клиент, которое будет выдавать запрос на соединение с сервером. В листинге 16.8 приведен код, ожидающий соединения от программы-клиента.

Листинг 16.8. Ожидание запроса на подключение от клиента

```
// Класс AcceptBluetoothThread принимает входящие запросы
private class AcceptBluetoothThread extends Thread {

  private final BluetoothServerSocket myServerSocket;

  public AcceptThread() {
       // Используем временный объект, который позже
       // будет присвоен члену myServerSocket, поскольку
       // myServerSocket — финальный член класса, и потом уже
       // не может быть изменен
       BluetoothServerSocket tmp = null;
       try {
       // MY_UUID — идентификатор, также используемый клиентом
       tmp = mAdapter.listenUsingRfcommWithServiceRecord(NAME,MY_UUID);
       } catch (IOException e) { }
       // Присваиваем tmp члену класса myServerSocket
       myServerSocket = tmp;
  }
  public void run() {
       BluetoothSocket socket = null;
       // Прослушиваем соединения
       while (true) {
       try {  // Принимаем соединение
```

```
                socket = myServerSocket.accept();
                } catch (IOException e) {
                        break;
        }
        // Если соединение было принято
        if (socket != null) {
                // Производим обработку соединения — в отдельном потоке
                DoSomethingWith(socket);
                // После обработки соединения закрываем сокет
                myServerSocket.close();
                break;
        }
    }
}
/** Действие в случае отмены соединения */
  public void cancel() {
        try {  // Закрываем сокет
                myServerSocket.close();
        } catch (IOException e) { }
  }
}
```

Теперь осталось написать приложение-клиент, устанавливающее соединение с Bluetooth-сокетом. Пример класса, который можно использовать для установки соединения, приведен в листинге 16.9.

Листинг 16.9. Установка соединения с сервером (Bluetooth-сокетом)

```
private class ConnectThread extends Thread {
  private final BluetoothSocket mySocket;
  private final BluetoothDevice myDevice;

  public ConnectThread(BluetoothDevice device) {
    // Используем временный объект, который позже
    // будет присвоен члену mySocket, поскольку
    // mySocket — финальный член класса, и потом уже
    // не может быть изменен
    BluetoothSocket tmp = null;
    myDevice = device;
    // Получаем BluetoothSocket для соединения с BluetoothDevice
    try {
      // MY_UUID — идентификатор, такой же использует сервер
      tmp = device.createRfcommSocketToServiceRecord(MY_UUID);
    } catch (IOException e) { }
    mySocket = tmp;
    }
```

```
    public void run() {
    // Отключаем обнаружение устройств, поскольку оно замедляет
    // соединение
    mAdapter.cancelDiscovery();
    try {
       // Соединяемся с устройством через сокет
       mySocket.connect();
    } catch (IOException connectException) {
       // Невозможно подключиться, закрываем сокет
       try {
              mySocket.close();
       } catch (IOException closeException) { }
       return;
      }
  // Соединение установлено, производим его обработку
  // в отдельном потоке
  DoSomethingWith(mySocket);
}
// Отмена соединения, закрываем сокет
  public void cancel() {
       try {
              mySocket.close();
       } catch (IOException e) { }
   }
}
```

16.8. Дополнительное оборудование виртуального устройства

Полностью списывать со счета эмулятор не стоит. В *главе 2* (см. рис. 2.21) было показано, как можно добавить к виртуальному устройству имитируемое оборудование. Для платформы 2.2 вы можете добавить следующие устройства (рис. 16.3):

- SD-карту;
- DPad;
- акселерометр;
- трекбол;
- звуковое устройство;
- камеру;
- батарею (для управления питанием);
- микрофон (Audio recording support);
- тачскрин;

- GPS-адаптер;
- GSM-модем.

Для платформы 3.0 список дополнительного оборудования скуден, но только потому, что по умолчанию все доступное оборудование уже добавлено в список **Hardware** виртуального устройства.

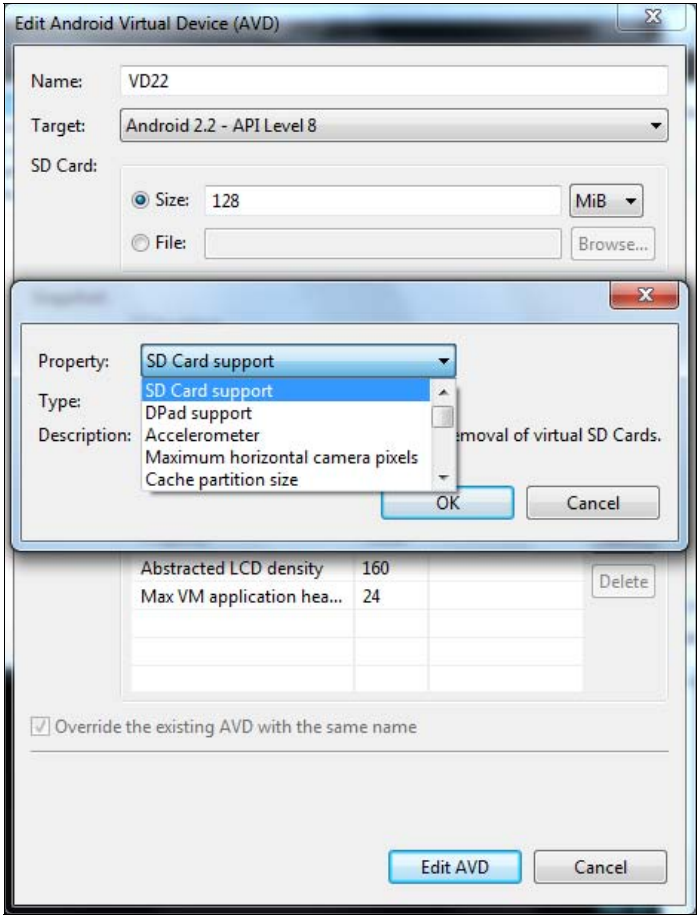


Рис. 16.3. Редактирование виртуального устройства: добавление дополнительного оборудования

ГЛАВА 17

Работа с Google Play Маркет

17.1. Что такое Play Маркет?

Создать хорошее приложение — мало. Нужно его еще распространить. Можно создать собственный сайт и заниматься раскруткой и сайта, и приложения. Такой подход приемлем для крупных компаний с большим ассортиментом приложений и с желанием потратить круглую сумму на хорошую рекламную кампанию для сайта. Ведь если о нем никто не узнает, то вы не сможете получить доход от созданных программ.

Кроме рекламной кампании нужно будет решить еще и ряд технических вопросов. Один из них — это борьба с пиратством. Согласитесь, некий Вася Пупкин может, единожды заплатив, скачать ваше приложение, а потом распространять его по всему Интернету бесплатно. Второй — механизм получения оплаты и загрузки приложения. Если вы не являетесь веб-разработчиком, придется воспользоваться услугами специальной компании. Для большой фирмы — это не проблема, в ее штате уже есть веб-разработчики. А для отдельного программиста создание такого сайта может вылиться в круглую сумму.

Небольшой компании или отдельному программисту гораздо выгоднее распространять свои Android-приложения через Play Маркет (**https://play.google.com**). В результате решается множество проблем, которые Play Маркет берет на себя, а именно:

- распространение программы;
- обновление программы (информацию о новой версии получат пользователи, уже купившие программу или просто ее загрузившие — для бесплатных программ);
- защиту от несанкционированного распространения;
- получение оплаты за программу.

Play Маркет — это хранилище и одновременно интернет-магазин Android-приложений. Приложения могут быть коммерческими (для их установки требуется оплата) или бесплатными (пользователь может свободно скачать приложение). В первом случае Play Маркет выступает в качестве посредника между разработчиком и пользователем. Во втором — Play Маркет представляет собой обычное

хранилище, куда разработчики могут загрузить приложения, а пользователи — скачать.

Забегая наперед, скажу, что Play Маркет — не панацея. Существуют и другие сайты, помогающие распространять ваши приложения, — например, сайт:

http://androidapplications.com/.

Получение оплаты за программу в случае с Play Маркет оставляет желать лучшего — используется система Google Checkout, через которую разработчики из России и Украины могут получать за продажи в Play Маркет банковские переводы. Но такие переводы осуществляются только на счета в долларах США[1], поэтому если у вас нет такого счета, то его придется открыть (в любом банке). Конечно, если для начала вы хотите распространять только бесплатные приложения, счет открывать не требуется.

Есть у Play Маркет и еще одна особенность. Далеко не все могут распространять/продавать/покупать приложения. В предыдущем издании этой книги приводилась даже отдельная табличка со странами, в которых можно было регистрировать аккаунты продавца/разработчика, чтобы покупать/продавать приложения. Теперь все это в прошлом — список стран расширен до такой степени, что нет смысла приводить его в книге. Поэтому остановимся только на странах, в которых может побывать эта книга. На Украине, в Беларуси и России разрешается скачивание как платных, так и бесплатных приложений. Однако для каждой страны есть барьер стоимости платных приложений. Для России — это от 30 до 6000 рублей, для Украины — от 8 до 1650 гривен, для Беларуси информация не приводится. Это означает, что в России нельзя продать приложение, дороже чем за 6000 рублей (на данный момент), а на Украине — за 1650 гривен.

Дополнительную информацию можно получить в справочном центре:

https://support.google.com/googleplay/android-developer/table/3541286?hl=ru

Что же касается аккаунтов разработчика и продавца (не всегда продавец является разработчиком), то их можно зарегистрировать для всех этих трех стран. Дополнительная информация доступна тут:

https://support.google.com/googleplay/android-developer/table/3539140?hl=ru

17.2. Правила размещения приложений на Play Маркет

При регистрации аккаунта разработчика вам придется принять условия соглашения The Play Market Developer Distribution Agreement. В нем довольно много правил и всевозможной информации, поэтому мы остановимся только на самых важных его моментах.

[1] Переводы для России и Беларуси осуществляются в долларах, для Украины — еще и в гривнах. Так что для украинских разработчиков подойдет и счет в гривнах.

- Вы должны полностью принять условия Соглашения. Нарушение правил Соглашения приведет к «бану» вашего аккаунта и удалению ваших продуктов из Play Маркет.
- Вы можете распространять продукты бесплатно или за деньги. Если вы продаете программы, то обязаны использовать только платежную систему Google Checkout. Никакие другие системы: ни WebMoney, ни популярную на Западе PayPal, вы не можете использовать для работы с Play Маркет. На всякий случай прочитайте о Google Checkout по адресу:

  http://ru.wikipedia.org/wiki/Google_Checkout.
- Ничего не дается даром. Принимая условия Play Маркет, вы соглашаетесь, что Google будет удерживать 30 % вашего дохода. Другими словами, если вы продаете программу стоимостью 10 долларов, то получите за нее только 7 долларов (3 доллара будет удержано).
- Но 30 % — это еще не все. Получая 7 долларов за каждое проданное приложение, вы не должны забывать, что с них нужно заплатить налоги (их размер и правила оплаты зависят от законов вашей страны). Я не силен в налоговом законодательстве, поэтому хороший бухгалтер или хотя бы консультация с налоговым инспектором вам не помешают.
- Вы можете распространять бесплатную демо-версию своего приложения с потенциальной разблокировкой всей функциональности программы после покупки приложения. Но в этом случае вы обязуетесь для получения оплаты за приложение использовать Play Market Payment Processor. Могу поспорить, что прочитал ваши мысли — половина читателей наверняка подумали: буду бесплатно распространять демо-версию, а полную версию можно будет купить на моем сайте, и все деньги без удержания 30 % получу лично я. Такая схема пройдет до того момента, пока вас не вычислят и, в результате, не заблокируют, а ваши продукты не удалят с Play Маркет, поэтому нечистые на руку разработчики много не заработают. Вы не имеете права направить пользователей демо-приложения на какую-то другую платежную систему, кроме Google Checkout.
- Если в течение 48 часов пользователь удалит приложение (если оно ему не понравится), ему будут возвращены деньги.
- Разработчикам запрещается собирать какие-либо личные данные пользователей, загрузивших (купивших) их приложения, в процессе использования приложения. Это правило можно обойти, если вы и пользователь заключите между собой специальное соглашение, но в таком случае вы обязаны явно (например, при первом запуске) предупредить пользователя о том, что ваше приложение будет собирать и обрабатывать его персональные данные.
- Google не запрещает распространять ваши Android-приложения через другие каналы (например, через собственный сайт). Но если вы распространяете конкретное приложение через Play Маркет, вам запрещено использовать другие каналы для продажи этого приложения.

- Старайтесь создавать качественные приложения. На страницах Play Маркет работает система рейтинга, и скачавшие вашу программу пользователи могут оценить ее по пятибалльной шкале. Если оценка будет слишком низкой, это отрицательно скажется на продажах приложения.
- Продавая свое приложение, вы предоставляете пользователю неэксклюзивное право использовать ваш продукт на его устройстве. Но если такого короткого соглашения мало, вы можете написать отдельное пользовательское соглашение (End User Licence Agreement, EULA).
- Вы не можете в названии приложения использовать слово «Android», также запрещено использовать логотип Android. Подробную информацию на этот счет можно получить по адресу **http://www.android.com/branding.html**.

17.3. Регистрация аккаунта разработчика

Чтобы зарегистрировать аккаунт разработчика Android, вам потребуются 25 долларов (именно столько стоит аккаунт с правом публикации приложений) и аккаунт Google (его можно получить, зарегистрировавшись на **www.gmail.com**).

> ***Варианты оплаты аккаунта***
>
> Чтобы заплатить 25 долларов, вам понадобится долларовая платежная карта с возможностью оплаты в Интернете и за границей — при открытии карточки укажите оба эти условия менеджеру банка. Если оба эти условия не соблюдены, перевести деньги вы не сможете. Можно поступить проще. Создайте QIWI-кошелек, а в нем — виртуальную карточку. При этом не нужно идти в банк, писать заявления и т. д. Да, я сейчас сэкономил вам кучу времени! Деньги в QIWI-кошелек можно или положить через терминал в ближайшем магазине, или перевести с другого электронного кошелька вроде Яндекс.Деньги или WebMoney.

Весь процесс создания аккаунта разработчика состоит из трех этапов: создания аккаунта, оплаты взноса в 25 долларов и принятия соглашения о распространении ПО через Play Маркет. Соглашение можно прочитать прямо сейчас:

https://play.google.com/about/developer-distribution-agreement.html

Итак, представим, что Google-аккаунт и заветные 25 долларов у вас имеются. Перейдите по ссылке **https://play.google.com/apps/publish/signup/** для регистрации аккаунта разработчика.

Процесс регистрации предельно прост — вам нужно указать свое имя, телефон и сайт. Все эти поля являются обязательными для заполнения. После ввода личных данных надо уплатить регистрационный взнос с помощью Google Checkout. Пока вы не заплатите 25 долларов, вы не получите доступ к консоли разработчика. Когда же взнос уплачен, открыть консоль управления можно здесь:

https://play.google.com/apps/publish/signup/

Интерфейс консоли управления предельно прост — вы разберетесь с ней без моих комментариев. Мы же лучше сконцентрируемся на подготовке ваших приложений к публикации на Play Маркет.

Дополнительная информация доступна по ссылкам:

https://support.google.com/googleplay/android-developer/answer/113469?hl=ru

http://indevices.ru/android/solved-troubles/razmestit-app-google-play-android-market.html

17.4. Телефон для разработчика: Android Developer Phone

Из консоли разработчика вы можете заказать Android Developer Phone (рис. 17.1). Android Developer Phone (ADP) — это полноценный телефон. Первоначально ADP строился на базе телефона T-Mobile G1, затем — на базе HTC Dream.

Рис. 17.1. Android Developer Phone

Стоит это чудо — 399 долларов без учета доставки и регистрационного взноса (25 долларов), который вы уже заплатили. Стоит ли покупать этот телефон? Давайте это выясним. С одной стороны, цена не так уж и велика, — примерно столько стоит любой другой смартфон среднего уровня (бюджетные модели из рассмотрения исключаем).

Взглянем на функции ADP. За названные деньги вы получаете полнофункциональное и полностью разблокированное устройство с камерой, слайдером, клавиатурой, GPS и картой памяти 1 Гбайт. В общем, вполне прилично. «Разблокированное» — означает, что устройство не привязано к какому-либо оператору, и вы можете использовать любую SIM-карту, а также загружать новые версии «прошивки» и операционной системы Android, чего нельзя делать с некоторыми обычными смартфонами, купленными в салонах мобильной связи.

С другой стороны, функциональность ADP оставляет желать лучшего. Если вы не планируете переустанавливать операционную систему, а желаете просто протестировать свои приложения на определенной платформе, лучше присмотритесь

к обычным смартфонам, которые можно купить в любом специализированном магазине. Дело в том, что программное обеспечение, поставляемое с ADP, является базовым, и для повседневной эксплуатации смартфона оно вряд ли подходит.

Отсюда можно сделать вывод: на начальном этапе, когда вы еще не заработали на своих приложениях ни цента, ADP вам не нужен. Но по мере вашего роста можно купить ADP как дань моде, ведь Android-разработчик без ADP — это как сапожник без сапог.

Вот только купить его могут не все, а избранные. ADP поставляется в США, Великобританию, Германию, Японию, Индию, Канаду, Францию, Тайвань, Испанию, Австралию, Сингапур, Швейцарию, Нидерланды, Австрию, Швецию, Финляндию, Польшу и Венгрию. России в этом списке нет, как и любой другой страны из бывшего СССР. При желании, конечно, заполучить ADP можно — попросить друзей из той страны, которая есть в заветном списке, купить ADP и переслать его вам. Но, чтобы на этом не спекулировали, Google... ограничила число телефонов, которое можно заказать с одного аккаунта, — на один аккаунт приходится всего один телефон. Так что регистрационный взнос придется платить, видимо, дважды. После всего этого хочется пойти в магазин, купить обычный смартфон и не иметь никакого дела с ADP...

17.5. Подготовка приложений к продаже

17.5.1. Тестирование на разных устройствах

Нужно убедиться, что ваше приложение нормально работает на разных смартфонах. Для этого понадобится несколько устройств различных производителей, желательно с разными версиями Android и разрешениями экрана. Где взять смартфоны? Один из вариантов — купить. Второй вариант — одолжить у друзей. Одно только можно сказать: не стоит выкладывать свое приложение на Play Маркет, если оно не протестировано хотя бы на 3–4 устройствах разных производителей.

17.5.2. Поддержка другого разрешения экрана

Не забудьте проверить, как выглядит ваше приложение на экране, размер которого отличается от размера экрана эмулятора по умолчанию. Здесь все намного проще: создайте несколько эмуляторов с разными размерами экрана и запустите приложение на каждом из них.

17.5.3. Локализация

Чтобы расширить аудиторию своего приложения и, следовательно, заработать на нем больше денег, нужно выполнить локализацию, т. е. перевод приложения на разные языки.

С технической точки зрения выполнить локализацию довольно просто. Создайте в каталоге res несколько каталогов values-*: например — values-en для английского языка, values-de — для немецкого и т. д. (рис. 17.2). Затем скопируйте в эти катало-

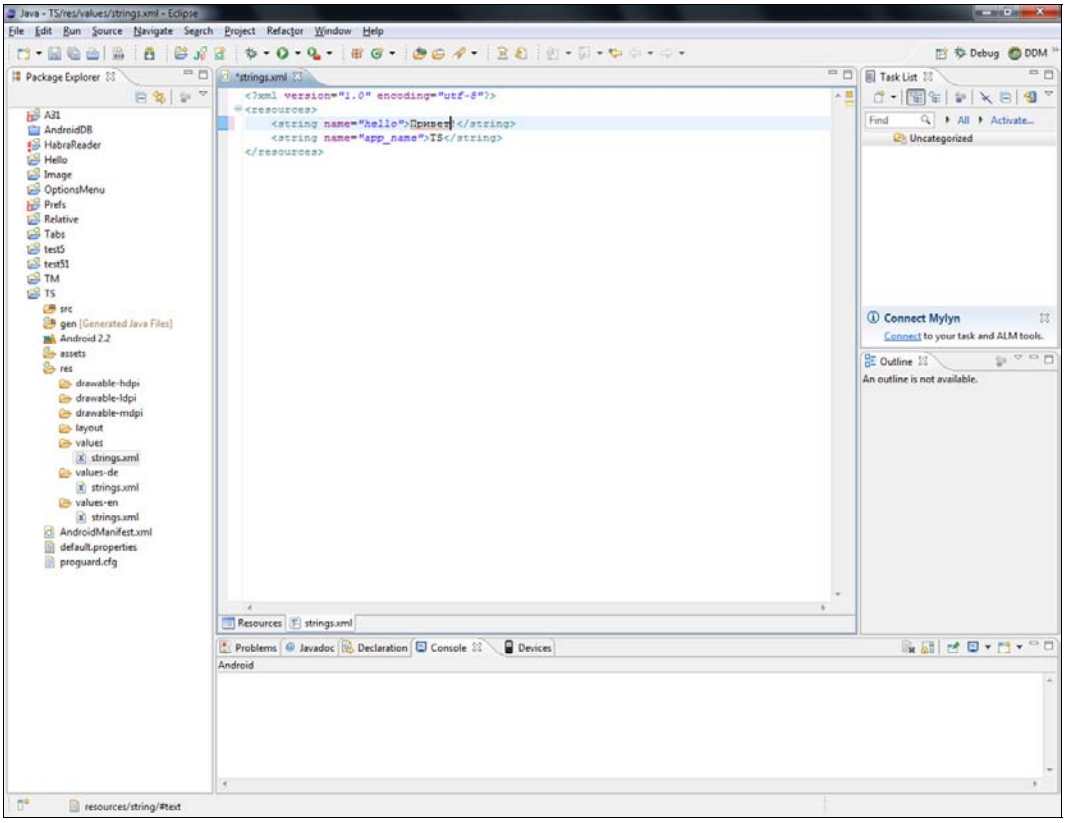


Рис. 17.2. Созданы отдельные ресурсные файлы для каждого поддерживаемого языка

ги оригинальный файл strings.xml из каталога res/values. Ясно, что локализацией нужно заниматься, когда приложение уже создано, все ошибки исправлены и больше не планируется добавлять в strings.xml новые строковые ресурсы.

Далее в настройки приложения добавляете параметр **Язык** (Language), позволяющий выбрать язык. Выбранный язык будет сохраняться с помощью предпочтений. При запуске приложения оно проверяет предпочтение и загружает соответствующие строковые ресурсы.

Кроме строковых ресурсов не забудьте также локализовать изображения (из каталогов res/drawable-*), если они содержат надписи, и меню приложения.

Вот только, чтобы не опростоволоситься, вам нужно нанять профессионального переводчика, который и выполнит перевод файлов strings.xml для каждого языка. Возможно, понадобится несколько переводчиков.

17.5.4. Пиктограмма приложения

Довольно часто начинающие разработчики забывают изменить пиктограмму приложения. Согласитесь, коммерческое приложение за 5–10 долларов с пиктограммой по умолчанию выглядит, по меньшей мере, смешно.

Изменить пиктограмму приложения очень просто. Подготовьте файл пиктограммы и поместите его в каталог res/drawable. Затем измените атрибут `android:icon` элемента `<application>` в файле манифеста:

```
<application anroid:icon="@drawable/my_icon" ...
```

17.5.5. Ссылки на магазин

Чтобы перенаправить пользователя на Play Маркет (например, для покупки полноценной версии программы или для покупки другой программы), используйте URI `market://`. Например:

```
Intent i = new Intent(Intent.ACTION_VIEW,
Uri.parse("market://search?q:имя_моей_программы"));
startActivity(i);
```

17.5.6. Подготовка APK-файла к загрузке

Прежде чем вы опубликуете свой APK-файл на Play Маркет, его нужно подписать, т. е. создать сертификат для вашего приложения. Проще всего это сделать с помощью утилиты keytool, которая находится в каталоге C:\Program Files\Java\jdk1.6.X_XX\bin (в подкаталоге bin каталога, в который вы установили JDK):

```
keytool -genkey -v -keystore c:\my_keys\app.keystore -alias my_android_app
-storepass 123456 -keypass 111456 -keyalg RSA -validity 9125
```

Параметры утилиты keytool приведены в табл. 17.1 в порядке их указания в команде, представленной ранее.

Таблица 17.1. *Параметры утилиты keytool*

Параметр	Описание
`genkey`	Генерирует пару ключей — публичный и приватный
`v`	Выводит подробные сообщения, обычно этот параметр можно опустить
`keystore`	Путь к хранилищу ключей, в нашем случае мы используем файл c:\my_keys\app.keystore
`alias`	Псевдоним в базе данных ключей. Чтобы вам было понятнее, используйте в качестве псевдонима название приложения (без пробелов, разумеется)
`storepass`	Пароль для хранилища
`keypass`	Пароль для приватного ключа. Укажите более сложные пароли, чем мои. Пароли для хранилища и приватного ключа могут совпадать, но лучше, когда они разные
`keyalg`	Алгоритм ключа, обычно используется RSA
`validity`	Период проверки в днях, в нашем случае 9125 дней, т. е. 25 лет. Двадцать пять лет — это не много. Если вы планируете публиковать приложение на Android, срок действия сертификата у него должен быть до 22 октября 2033 года. Сейчас 2014 год, следовательно, минимально необходимый сертификат — на 19 лет. Исходя из документации по Android, 25 лет — рекомендуемый минимальный срок действия сертификата. Вы же можете создать сертификат на 50 лет — этого более чем достаточно

Утилита keytool задаст ряд вопросов (рис. 17.3), а затем вы увидите сообщение о том, что сертификат сгенерирован:

```
Generating 1,024 bit RSA key pair and self-signed certificate (SHA1withRSA)
with a validity of 9125 days
        for: CN=Denis, OU=Dev, O=Home Ltd, L=Russia, ST=Unknown, C=RU
[Storing c:\my_keys\app.keystore]
```

Теперь у вас есть цифровой сертификат, которым вы можете подписать ваш APK-файл. Для этого вам нужно использовать утилиту jarsigner:

```
jarsigner -keystore c:\my_keys\app.keystore -storepass 123456
-keypass 111456 путь_к_apk_файлу my_android_app
```

Давайте разберемся, что здесь что. Параметр `keystore` указывает путь к хранилищу ключей. Мы договорились, что будем использовать файл c:\my_keys\app.keystore.

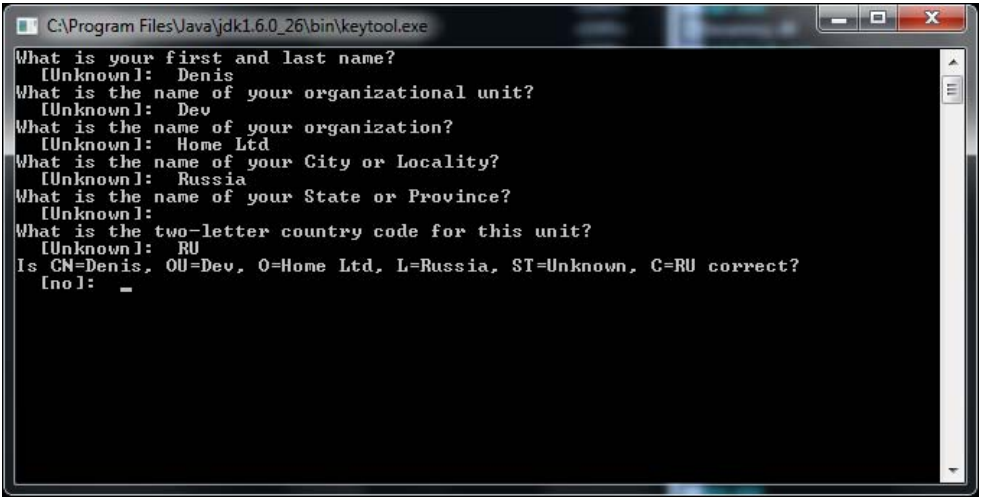


Рис. 17.3. Создание пары ключей утилитой keytool

Затем указываются пароли к хранилищу и к приватному ключу. Следующий параметр — это путь к APK-файлу вашего приложения, последний параметр — псевдоним, указанный при создании ключа.

Уже подписанный APK-файл лучше всего передать программе zipalign для приведения несжатых данных в соответствие с границами памяти, что позволит оптимизировать программу — она будет быстрее выполняться. У процессоров Android-устройств четырехбайтная граница памяти, поэтому команда `zipalign` будет выглядеть так:

```
zipalign -v 4 signed.apk optimized.apk
```

Как вы уже догадались, здесь: `signed.apk` — это подписанный APK-файл, а `optimized.apk` — это имя нового, оптимизированного, APK-файла, который полностью готов для публикации на Play Маркет.

Если командная строка вас пугает или кажется вам слишком архаичной, то щелкните в Eclipse правой кнопкой мыши на проекте и выберите команду меню **Android**

Tools | Export Signed Application Package. После этого откроется окно, в котором разберется даже ребенок. Сначала нужно ввести имя файла ключа и пароль к нему (рис. 17.4), далее — информацию о разработчике (рис. 17.5), а затем — путь к подписанному APK-файлу (рис. 17.6). После нажатия кнопки **Finish** будет создан ключ, а ваше приложение — подписано этим ключом и помещено в каталог, указанный на последнем этапе экспорта.

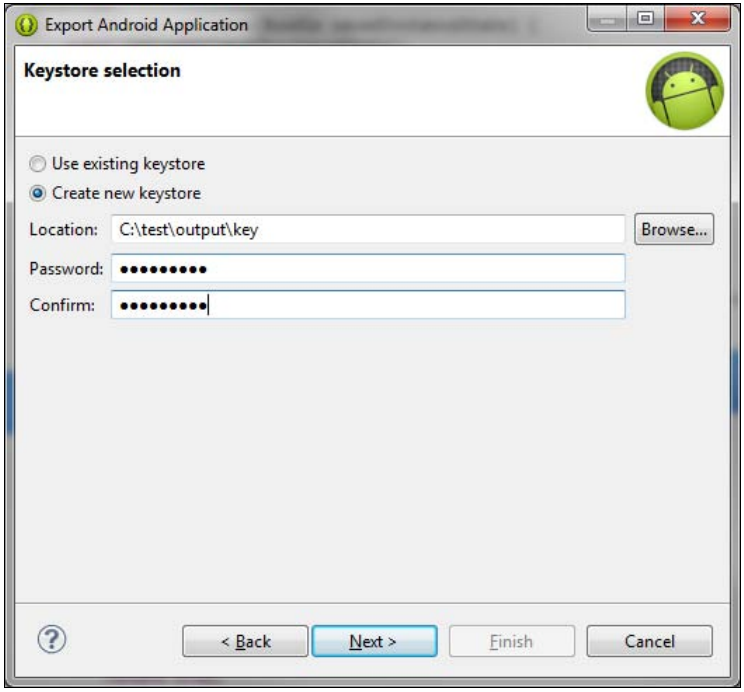


Рис. 17.4. Информация о создаваемом ключе

Но загружать файл пока рано. Попробуйте его установить вручную в эмулятор или физическое устройство. Для этого или запустите эмулятор, или подключите Android-устройство к компьютеру, а затем введите команду:

```
adb install <имя_apk_файла>
```

Запустите и протестируйте приложение. Затем попытайтесь удалить его:

```
adb uninstall <имя_пакета>
```

Например:

```
adb uninstall com.samples.my_app;
```

Если все прошло успешно, тогда перейдите в консоль разработчика и выберите **Загрузить приложение** — вы готовы загрузить свое приложение на Play Маркет.

Еще раз отмечу, что Play Маркет — это не единственное хранилище приложений. Вы также можете распространять свои приложения на следующих ресурсах: **http://soc.io/**, **http://slideme.org**, **http://www.androidgear.com**. Уверен, что в Интернете вы найдете и другие сайты.

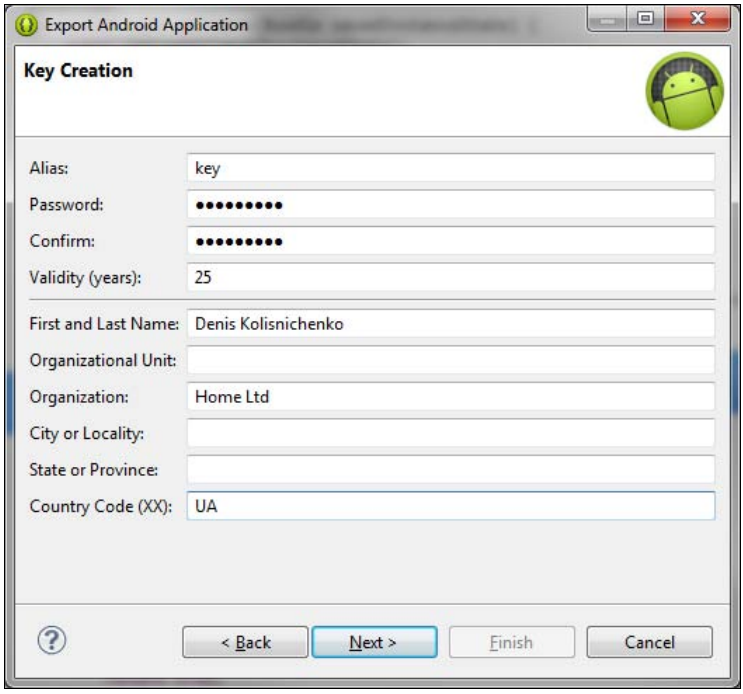


Рис. 17.5. Информация о разработчике

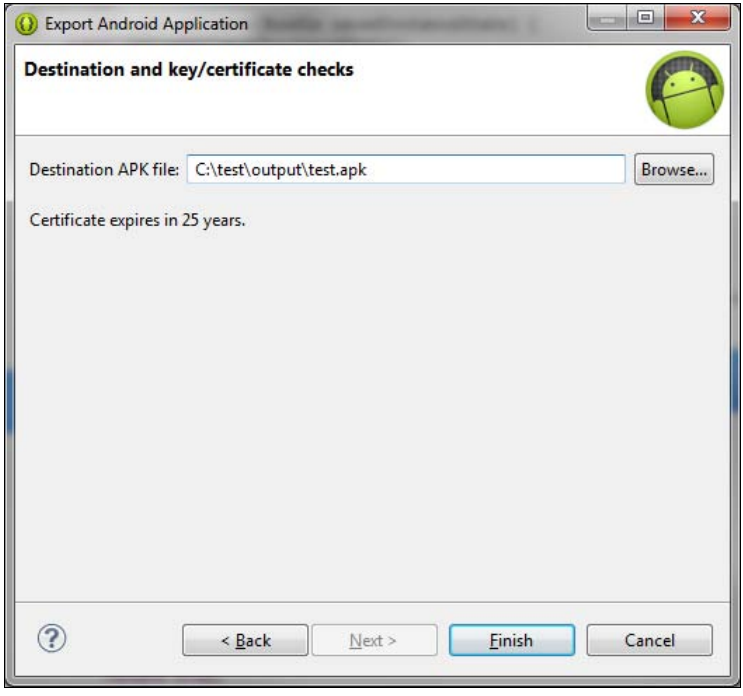


Рис. 17.6. Путь к APK-файлу

ГЛАВА 18

Отладка приложений

18.1. Средства среды Eclipse

Среда Eclipse содержит довольно мощные средства отладки приложений. Но в дополнение к ним вы можете использовать утилиты отладки из Android SDK, адаптированные для отладки именно Android-приложений. Начнем со стандартных средств Eclipse, а затем рассмотрим утилиты отладки из Android SDK. Некоторые моменты уже были вкратце рассмотрены ранее (когда в этом была необходимость). Сейчас же мы рассмотрим их более подробно.

18.1.1. Выбор конфигурации запуска

Из меню **Run | Run as** вы можете выбрать один из профилей запуска приложения. Можно запустить программу как Android-приложение, как Java-апплет, как Java-приложение и т. д. Все зависит от того, какую программу мы разрабатываем. Современные версии Eclipse не помещают в это подменю действия, не соответствующие типу проекта.

Для редактирования конфигураций запуска используется команда меню **Run | Run Configurations**. Для редактирования конфигураций отладки: **Run | Debug Configurations**.

Вы можете изменить конфигурацию запуска отдельно для каждого проекта. Параметры каждой конфигурации задаются на трех вкладках.

- На вкладке **Android** (рис. 18.1) вы можете выбрать запускаемый проект. Как правило, имя проекта здесь соответствует тому, которое выбрано на панели слева. Вы можете выбрать и другой проект, но зачем?
- На вкладке **Target** (рис. 18.2) можно выбрать виртуальное устройство, на котором будет запущено приложение. Обычно это происходит автоматически — выбирается устройство, соответствующее платформе запускаемого приложения. Но вы можете установить переключатель **Always prompt to pick device** и выбрать устройство вручную. Это особенно полезно, если вы создали два устройства для одной и той же платформы, но с разными параметрами.

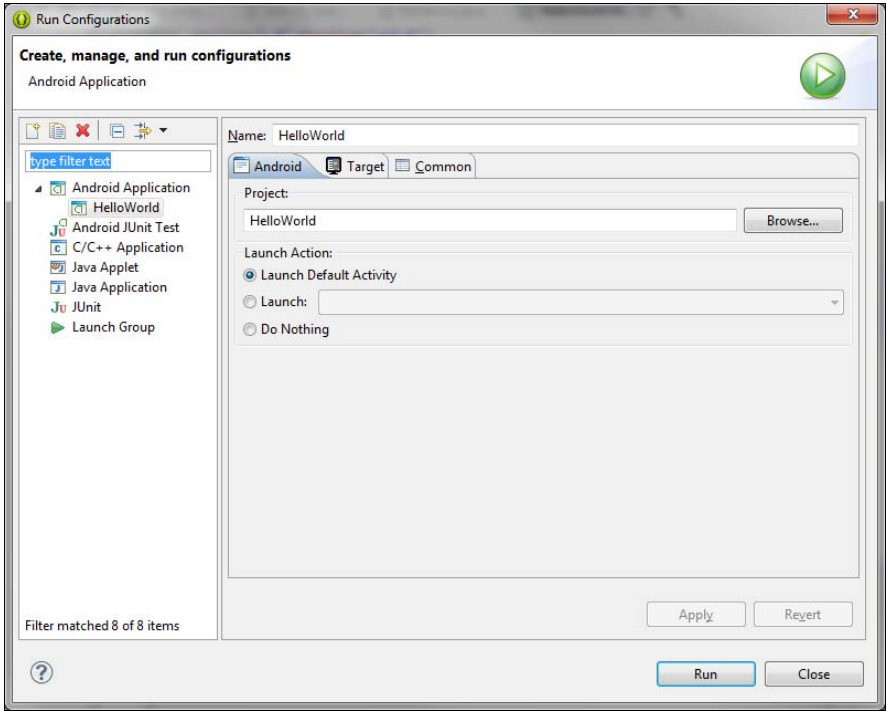


Рис. 18.1. Вкладка **Android**

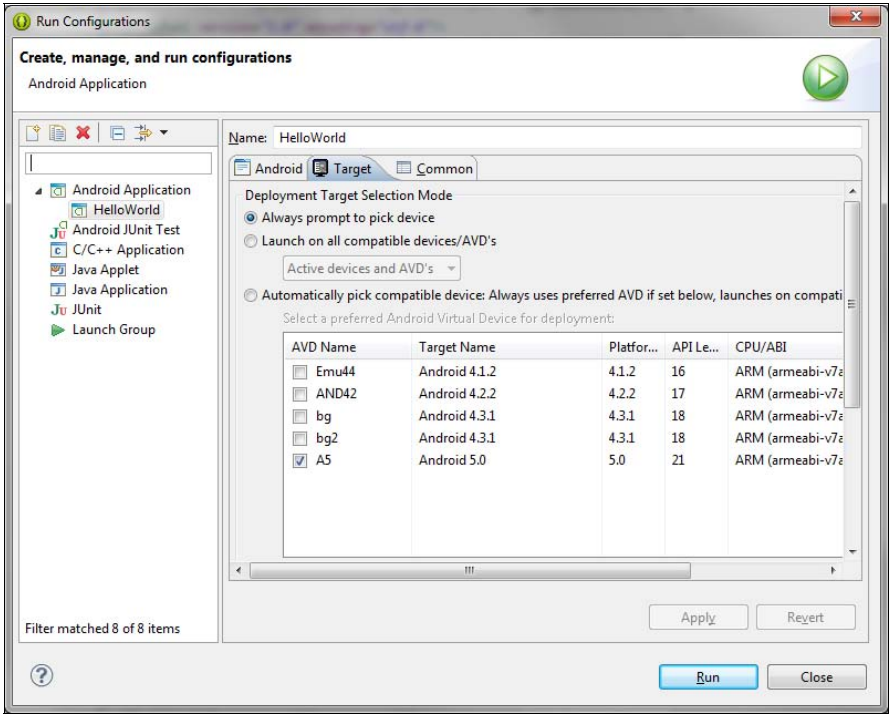


Рис. 18.2. Вкладка **Target**

- Параметры вкладки **Common** (рис. 18.3) больше относятся к самому проекту и настройкам Eclipse, нежели к процессу запуска. Вы можете выбрать, где сохранить проект, установить кодировку по умолчанию и другие параметры.

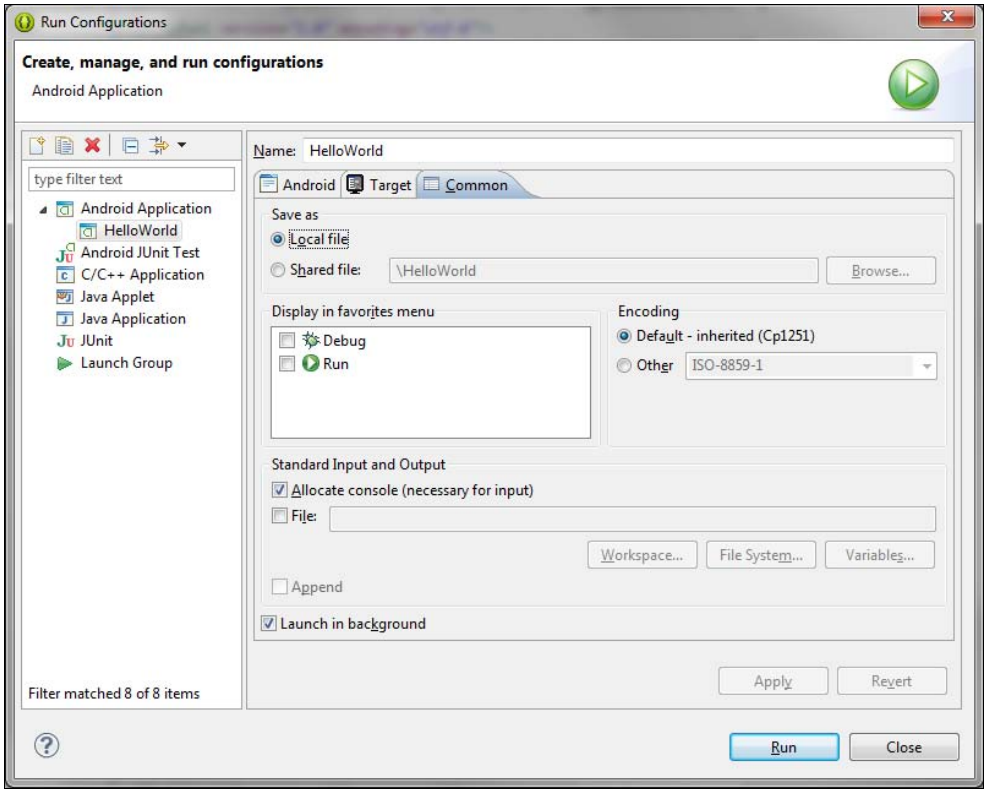


Рис. 18.3. Вкладка **Common**

18.1.2. Использование DDMS

Запустив приложение на выбранном виртуальном (или физическом) устройстве, вы можете запустить Dalvik Debug Monitoring Service (DDMS) для получения информации о состоянии устройства. Чтобы открыть DDMS, выполните команду меню **Window | Open Perspective | Other**, а из открывшегося окна (рис. 18.4) выберите **DDMS** (можно, конечно, запустить DDMS и с помощью командной строки).

Кстати, с помощью DDMS можно просматривать файловую систему физического/виртуального устройства — просто перейдите на вкладку **File Explorer** (рис. 18.5).

18.1.3. Перспектива Debug

Многие программисты привыкли к традиционному способу отладки, который заключается в установке точек прерывания (breakpoints). Среда Eclipse позволяет

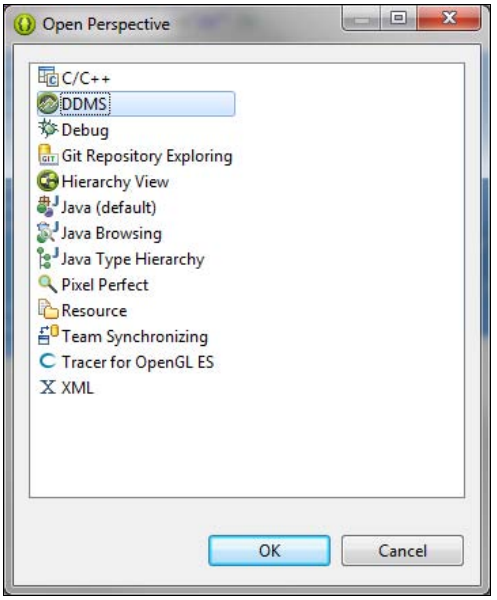


Рис. 18.4. Окно **Open Perspective**

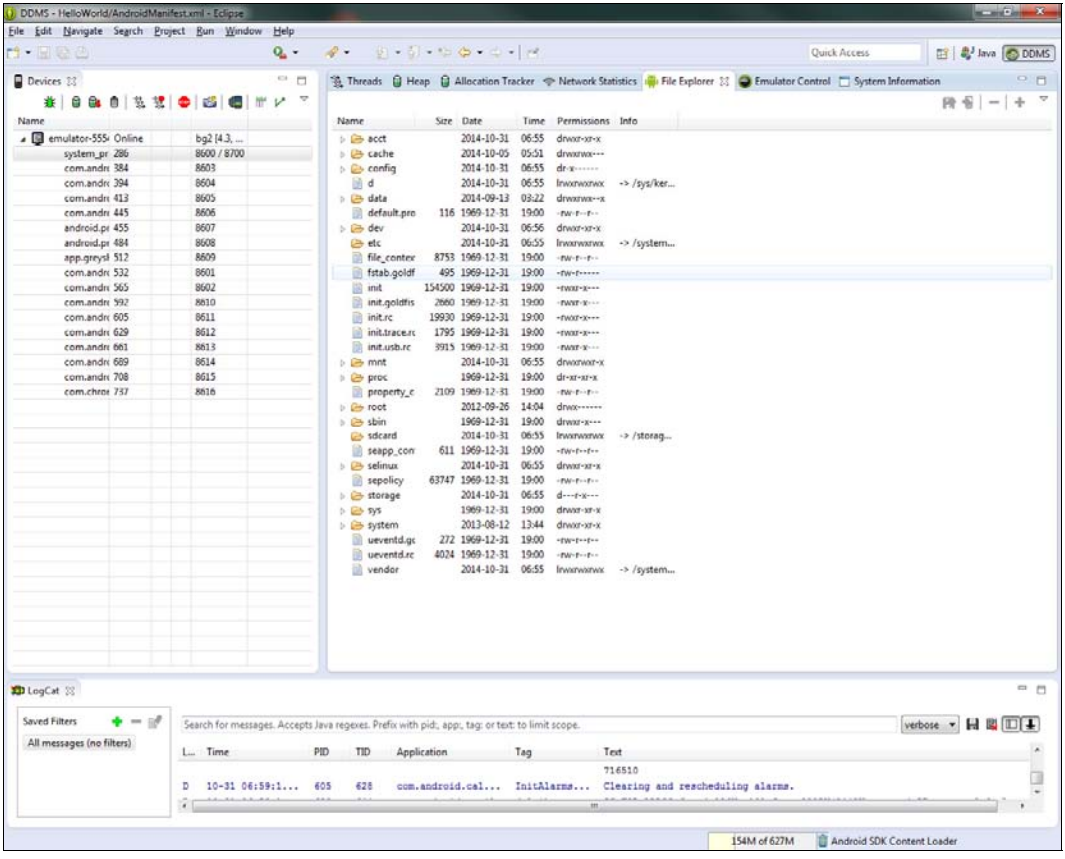


Рис. 18.5. Окно **DDMS**: просмотр содержимого файловой системы эмулятора

выполнять и такую отладку. Для этого откройте перспективу Debug (команда **Window | Open Perspective | Debug**).

> ***Перспектива окна Eclipse***
>
> Перспектива окна Eclipse определяет доступный инструментарий и внешний вид среды. Например, перспектива Debug используется для отладки приложений, перспектива Hierarchy View — для просмотра иерархии и т. д. Выбрать перспективу можно с помощью команды меню **Window | Open Perspective**.

После этого окно Eclipse будет выглядеть так, как показано на рис. 18.6. В верхней части присутствует вкладка **Debug** (станет активной, как только вы запустите процесс отладки командой **Run | Debug**). Справа от нее — две вкладки: **Variables** (отображает переменные приложения) и **Breakpoints** (отображает точки останова).

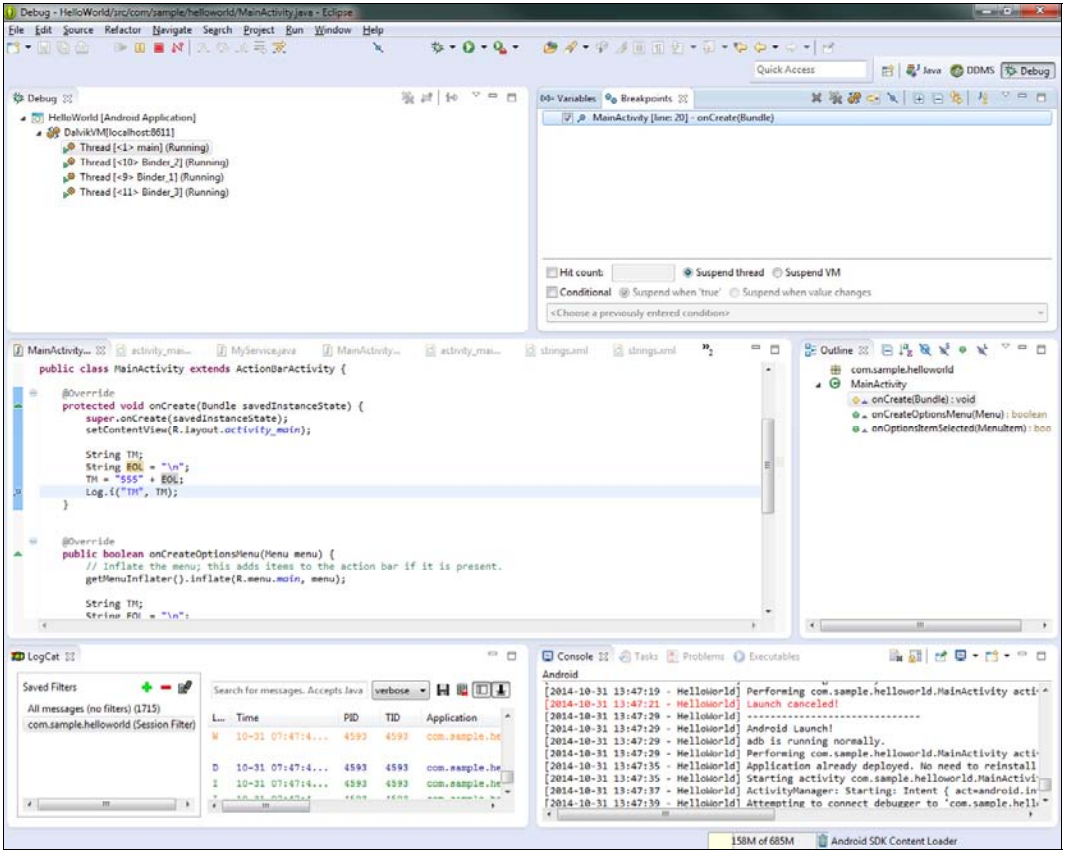


Рис. 18.6. Перспектива Debug: выполнение программы приостановлено, т. к. достигнута точка останова

Под этими вкладками находятся вкладка с исходным Java-кодом, вкладка с файлом манифеста и уже знакомая вкладка **Outline**. В нижней части окна расположены вкладки **Console** (консоль Android, обычно — журнал запуска программы), **Tasks** (список задач) и **LogCat** (журнал виртуального устройства).

Для установки точки останова просто щелкните двойным щелчком в месте будущей точки останова на полоске слева от кода — напротив строки кода. Точка останова сразу будет отображена на вкладке **Breakpoints**.

Затем запустите приложение на отладку командой **Run | Debug**. Вы увидите, как будет заполняться информацией вкладка **Debug**. Выполнение программы дойдет до первой точки останова и будет приостановлено. Далее вы можете управлять выполнением программы с помощью команд меню **Run**:

- **Resume** — продолжить выполнение (до следующей точки останова);
- **Suspend** — приостановить выполнение программы;
- **Terminate** — завершить выполнение программы;
- **Step Into** — когда выполнение программы дойдет до оператора вызова функции, операторы функции будут выполнены пошагово — по одному за одно нажатие клавиши <F5> (именно эта клавиша вызывает команду **Step Into**);
- **Step Over** — выполнение всей функции будет произведено за один шаг;
- **Run to Line** — запустить программу до достижения определенной строки кода. Как только выполнение достигнет заданной строки, оно будет приостановлено;
- **Skip All Breakpoints** — пропустить все последующие точки останова;

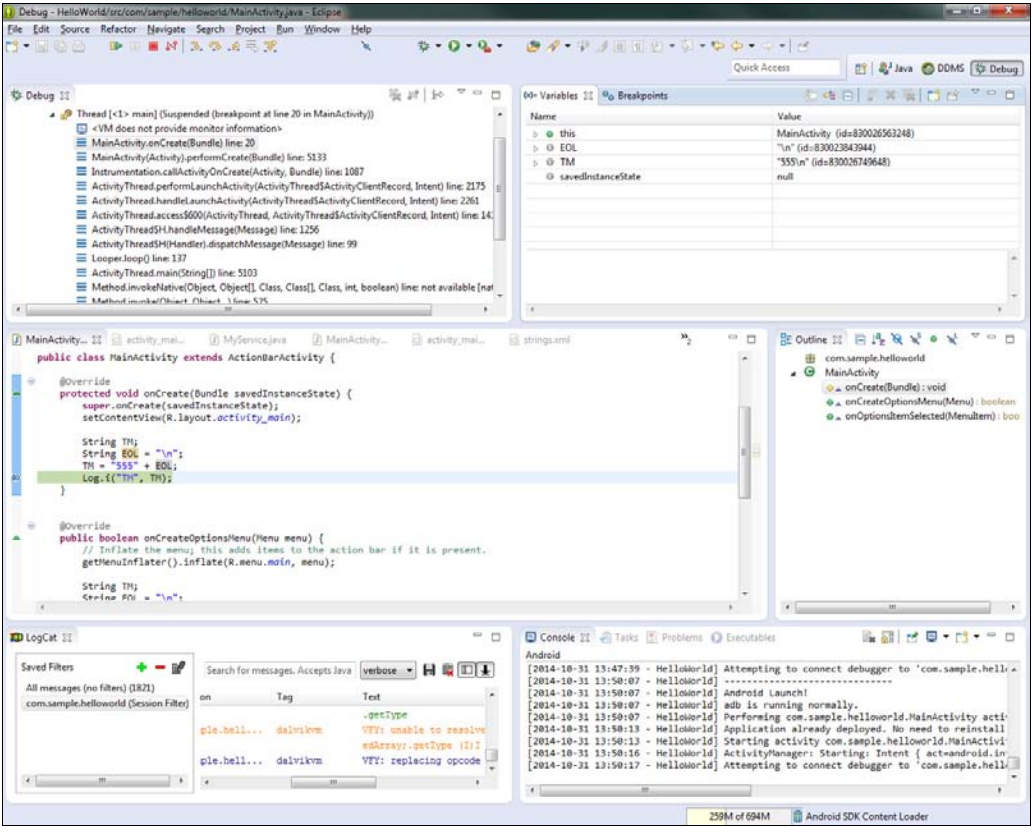


Рис. 18.7. Вкладка **Variables**: значения переменных

- **Remove All Breakpoints** — удалить все точки останова;
- **Watch —** просмотреть значения переменных программы (рис. 18.7).

18.2. Утилиты отладки из Android SDK

18.2.1. Android Debug Bridge

Android Debug Bridge (ADB) предоставляет способ управления состоянием эмулятора или подсоединенного к USB-порту компьютера физического устройства. ADB состоит из трех частей: клиента, сервера и демона. Клиент инициализируется с помощью сценария на машине разработчика. Сервер запускается как фоновый процесс и может быть запущен или остановлен командами:

```
adb start-server
adb kill-server
```

Демон тоже запускается как фоновый процесс, но в самом эмуляторе или на физическом устройстве.

Обычно перезапускать сервер не нужно — это делает менеджер SDK автоматически в случае необходимости, но теперь вы знаете, как осуществить перезапуск ADB вручную, если понадобится. Исполнимый файл adb.exe находится в каталоге platform-tools.

18.2.2. Использование LogCat

LogCat — утилита журналирования реального времени, доступная для операционной системы Android. Она собирает все сообщения системы и приложения, которые можно просмотреть и отфильтровать. Утилиту LogCat можно запускать как отдельно — с помощью команды `adb logcat`, так и воспользоваться вкладкой **LogCat** перспективы DDMS. Использовать вкладку **LogCat** намного удобнее — вы можете просматривать сообщения системы и приложения, не закрывая окна Eclipse (рис. 18.8).

Формат сообщений в журнале **LogCat** следующий:

- **Time** — время записи сообщения в журнал;
- **Type** — тип сообщения, может быть:
  - **V** (от Verbose) — подробное сообщение (наинизший приоритет);
  - **D** (от Debug) — отладочное сообщение;

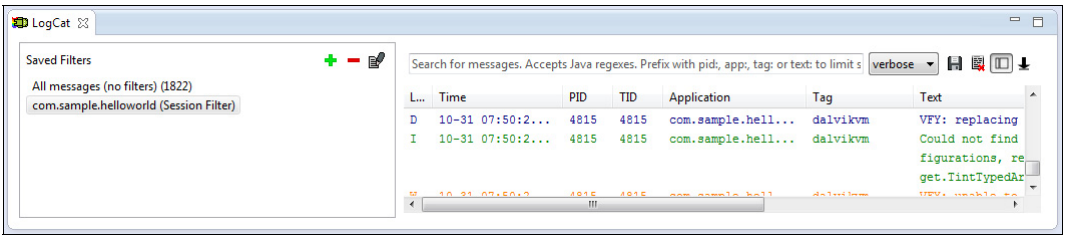


Рис. 18.8. Вкладка **LogCat**

- **I** (от Info) — информационное сообщение;
- **W** (от Warning) — предупреждение;
- **E** (от Error) — ошибка;
- **F** (от Fatal) — фатальная ошибка;
- **S** (от Silent) — наивысший приоритет, но при этом ничего не выводится;

❒ **pid** — номер процесса, сгенерировавшего сообщение;

❒ **tag** — тег сообщения (для фильтра);

❒ **Message** — само сообщение.

С помощью кнопок в верхней части окна **LogCat** вы можете отфильтровать сообщения определенного типа — например, выбрав **error** из списка, вы отобразите только ошибки (рис. 18.9).

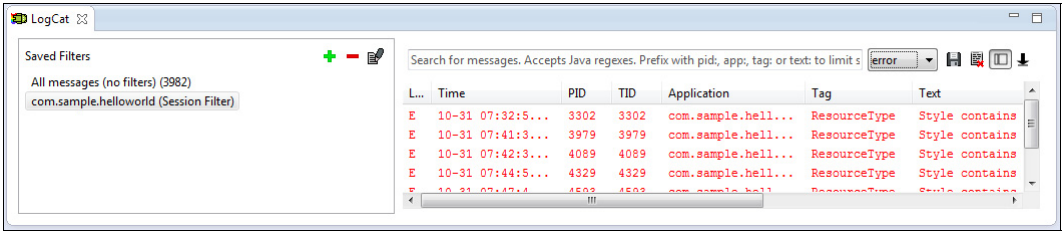


Рис. 18.9. Отображаем только ошибки

Иногда из соображений отладки полезно вывести какое-то сообщение в область **LogCat**. Это можно сделать так:

```
import android.util.Log;
...
Log.i("Тег", "Сообщение");
```

Например:

```
Log.i("TM", "555");
```

Наше сообщение будет отображено в области **LogCat** (рис. 18.10).

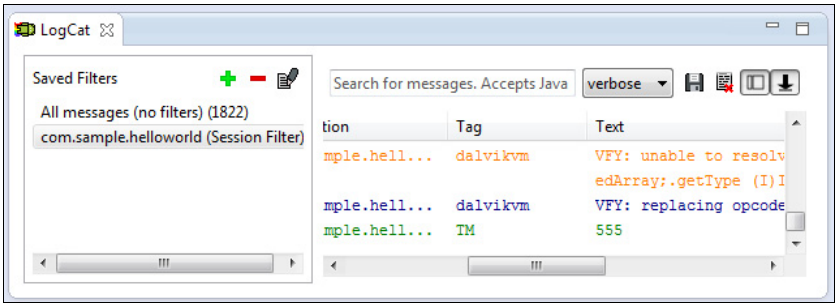


Рис. 18.10. Отладка с помощью **LogCat**

18.2.3. Системные утилиты отладки

Важно понимать, что Android — это почти Linux. Вы можете использовать на виртуальном (или реальном) устройстве любые стандартные Linux-утилиты. Для их запуска служит утилита adb:

```
adb shell <Linux-команда>
```

Например, в Linux для просмотра запущенных процессов служит команда `top`. Чтобы выполнить эту команду на текущем устройстве (виртуальном или реальном — без разницы), используется следующая команда (рис. 18.11):

```
adb shell top
```

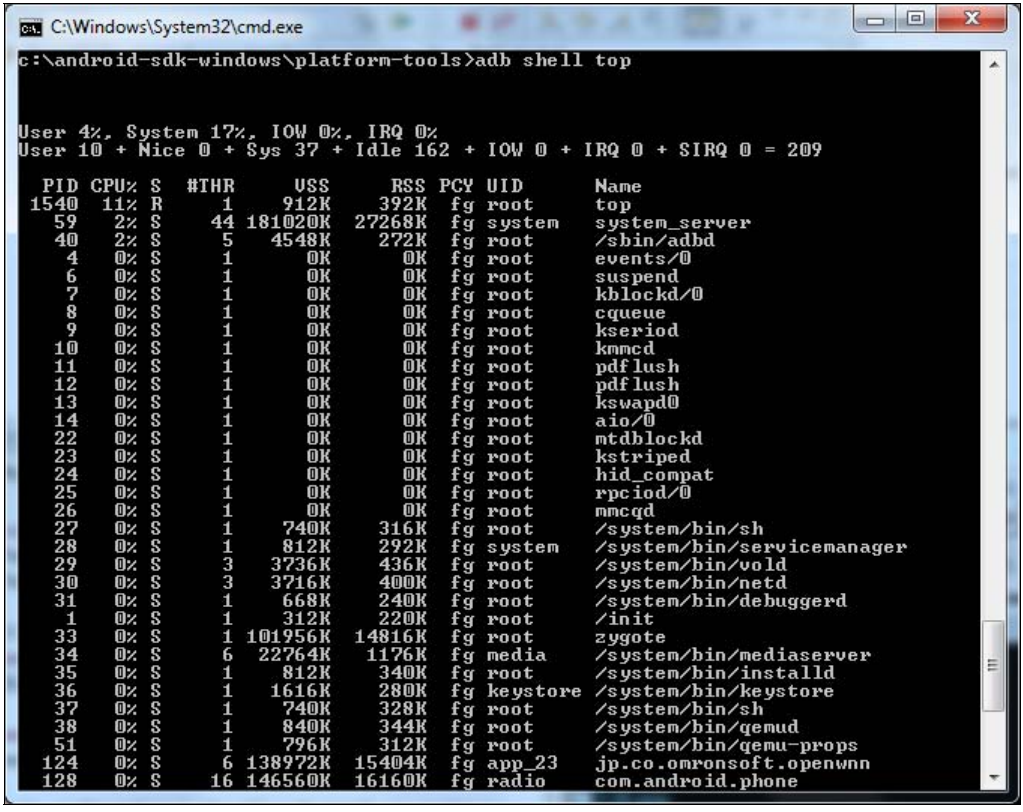


Рис. 18.11. Команда `top`

Можно просмотреть содержимое корневого каталога (рис. 18.12):

```
adb shell ls
```

Чтобы просмотреть содержимое другого каталога, нужно сначала в него перейти, а затем вызвать команду `ls`, например:

```
adb shell cd /etc; ls
```

Впрочем, намного удобнее просматривать файловую систему устройства с помощью перспективы DDMS.

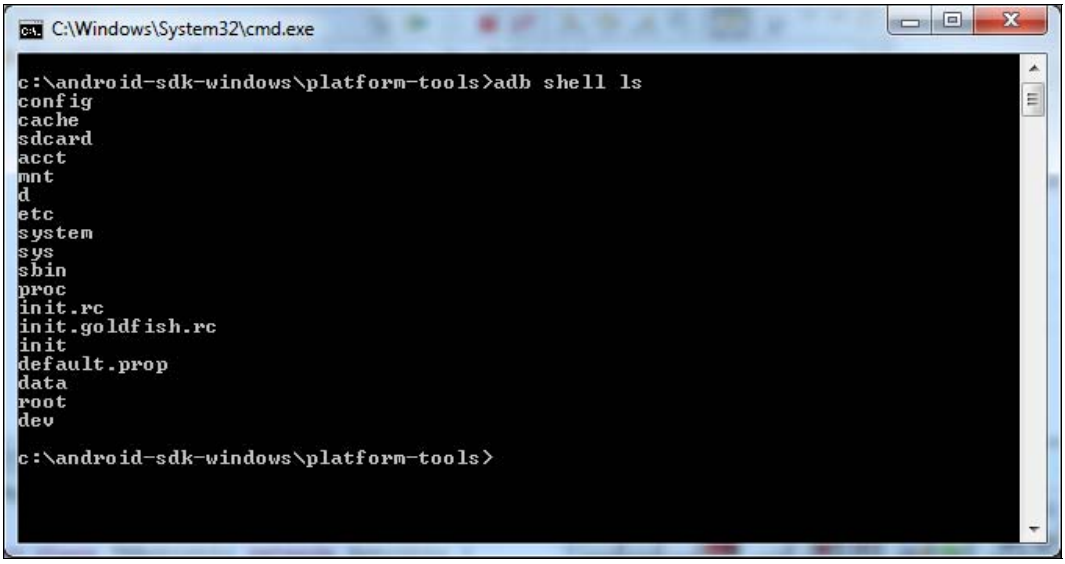


Рис. 18.12. Команда `ls`

Просмотреть распределение памяти можно с помощью команды `dumpsys`:

```
adb shell dumpsys meminfo <имя пакета>
```

Вывод этой команды для пакета `com.samples.tm` приведен на рис. 18.13.

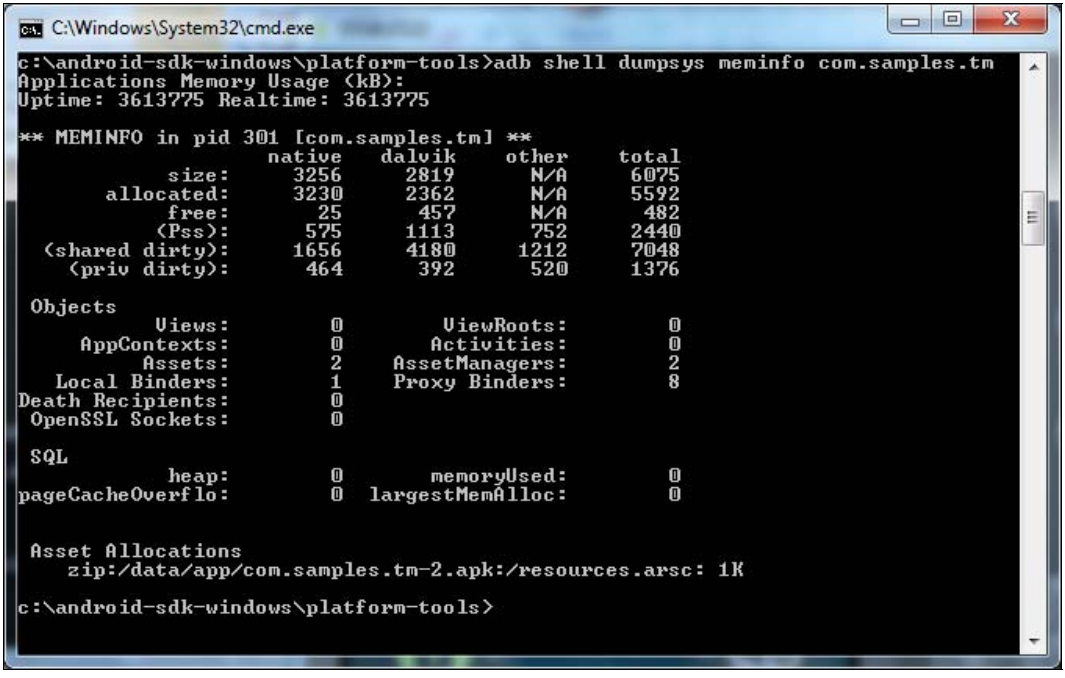


Рис. 18.13. Команда `dumpsys meminfo`

18.2.4. Отладчик gdb и Android-приложения

Если вы ранее программировали в Linux, то наверняка знакомы с отладчиком. Некоторые программисты не представляют себе жизни без него.

В Android вы тоже можете установить отладчик gdb. Для этого введите команды:

```
> adb shell
> adb /data/
> mkdir myfolder
> exit
> adb push gdbserver /data/myfolder
```

Теперь осталось запустить gdb:

```
> adb shell /data/myfolder/gdbserver 10.1.1.1:1234 myprogram
```

Здесь `10.1.1.1` — IP-адрес Android-устройства, а `1234` — номер порта. Последний аргумент — отлаживаемая программа.

Далее вы можете работать с gdb как обычно, используя его встроенные команды.

Вместо заключения

Разработка приложений для Android — процесс довольно интересный и перспективный. Интересный — хотя бы потому, что пишешь программу не для компьютера (чем сейчас никого не удивишь), а для мобильного устройства. Перспективный — потому, что разработчиков для мобильных устройств на наших просторах не так уж и много.

Ясное дело, что в одной книге полностью не раскроешь всех аспектов программирования для Android. Но книг на русском языке откровенно мало. Дополнительную информацию можно получить в книге А. Голощапова «Google Android: программирование для мобильных устройств»:

http://www.bhv.ru/books/book.php?id=187679

Связаться со мной и задать вопросы, если таковые появятся в ходе чтения книги, можно здесь:

http://www.dkws.org.ua

Специально для разработчиков приложений для Android я создал на своем форуме отдельный раздел, где вы найдете много интересного:

http://www.dkws.org.ua/phpbb2/viewforum.php?f=60

ПРИЛОЖЕНИЕ

App Inventor — среда быстрой разработки приложений

П.1. Что такое App Inventor?

App Inventor — это среда визуальной разработки Android-приложений, не требующая, как утверждается, навыков программирования. Первоначально она была разработана для Google Labs, а затем передана Массачусетскому технологическому институту (другими словами, сейчас сама компания Google не имеет никакого отношения к этой среде).

Разработка приложений в App Inventor осуществляется с помощью визуального языка программирования, похожего на язык Scratch, но если вы не имеете отношения к программированию, то это мало вам что скажет.

Итак, давайте разберемся, что же представляет собой App Inventor. Разработчики Android уверяют, что с помощью App Inventor даже домохозяйка сможет разработать свое приложение без минимальных навыков программирования. Вам в это верится? Мне — нет. Однако, используя App Inventor, у вас все-таки получится создать приложение без знаний языка Java и, вообще, без знания какого-либо языка программирования. Приложения в App Inventor создаются по принципу конструктора: расставил кнопки и другие элементы управления в окне «смартфона», собрал выполняемые программой действия и, вуаля — программа готова. Ошибиться в действиях как бы тоже невозможно, поскольку все компоненты приложения собираются по принципу пазла.

В Интернете есть видео, где девушка, якобы не имеющая никакого отношения к программированию, создает за пару минут приложение. Приложение простое: нажимаешь на изображение кота — слышишь звук его мява:

http://www.youtube.com/watch?v=8ADwPLSFeY8

Думаю, общее впечатление о том, что такое App Inventor у вас сформировалось. Теперь — мое отношение к этой среде. Если хочешь что-то выучить, то нужно учиться, разбираться, пробовать. Ничего просто так не дается. Да, App Inventor — хорошая попытка создать RAD-систему (систему быстрой разработки) для Android, но, на мой взгляд, попытка удалась не вполне.

В App Inventor мне не понравилось две особенности. Первая — довольно скудный функционал самой среды. Сложное Android-приложение вряд ли вы сможете построить (именно построить, а не разработать) с помощью App Inventor. Когда вы пишете приложение на Java, то ваши возможности ограничены, по сути, только вашими навыками в программировании. Если же в App Inventor не окажется нужного «пазла», то приложение создать не получится. Вторая — с помощью App Inventor простенькое приложение вы создадите, но размер APK-файла будет неприлично большим, я бы даже сказал — огромным.

Надо, впрочем, отметить, что большая часть моих претензий к среде App Inventor относится к ее первой версии. Вторая версия среды (App Inventor Version 2) по сравнению с первой стала значительно лучше. Во-первых, больше нет мучительной процедуры установки. Среда якобы была рассчитана на домохозяек, но для ее запуска на компьютере домохозяйки требовался опытный системный администратор. Сейчас надо лишь зайти на сайт среды и нажать кнопку **Create** — далее все будет работать, как и должно, без каких-либо танцев с бубном. Во-вторых, улучшена функциональность самой среды, появились дополнительные пазлы, создавать приложения стало удобнее. Но все равно вы остаетесь ограничены имеющейся функциональностью. Приведу небольшой пример. В Android нет диалогового окна выбора файла, который есть в API Windows. Вы не замечали, что у всех приложений Windows диалоговые окна открытия и сохранения файлов одинаковые? Это потому, что они реализованы на уровне API операционной системы. В Android такой реализации нет. Однако, используя Java и стандартные виджеты Android, вы можете реализовать необходимые диалоговые окна. А вот в App Inventor подобная реализация отсутствует. Следовательно, создать Android программу с диалоговым окном открытия файла с помощью App Inventor у вас не получится.

П.2. Начало работы с App Inventor

Для работы с App Inventor нужно установить виртуальную машину Java. Последнюю версию можно получить по адресу: **http://www.java.com/ru/download/**.

Загрузив и установив виртуальную машину Java, перейдите на сайт MIT App Inventor (рис. П.1) по адресу: **http://appinventor.mit.edu/explore/**.

На этом сайте находится много вспомогательных материалов, которые помогут разобраться со средой. Обязательно изучите их на досуге. А сейчас нажмите кнопку **Create** в верхнем правом углу — вы увидите запрос на доступ к вашему Google-аккаунту. Разрешите приложению MIT AppInventor Version 2 такой доступ.

Далее среда предложит вам просмотреть краткий обзор (рис. П.2). Вы можете нажать кнопку **Take Survey Now**, чтобы просмотреть обзор прямо сейчас, кнопку **Take Survey Later**, чтобы просмотреть обзор позже и, наконец, кнопку **Never Take Survey**, чтобы вовсе отказаться от просмотра. Пока же, чтобы не отвлекаться от чтения книги, нажмите кнопку **Take Survey Later** — среда подскажет, как подключить и настроить Android-устройство или эмулятор (рис. П.3).

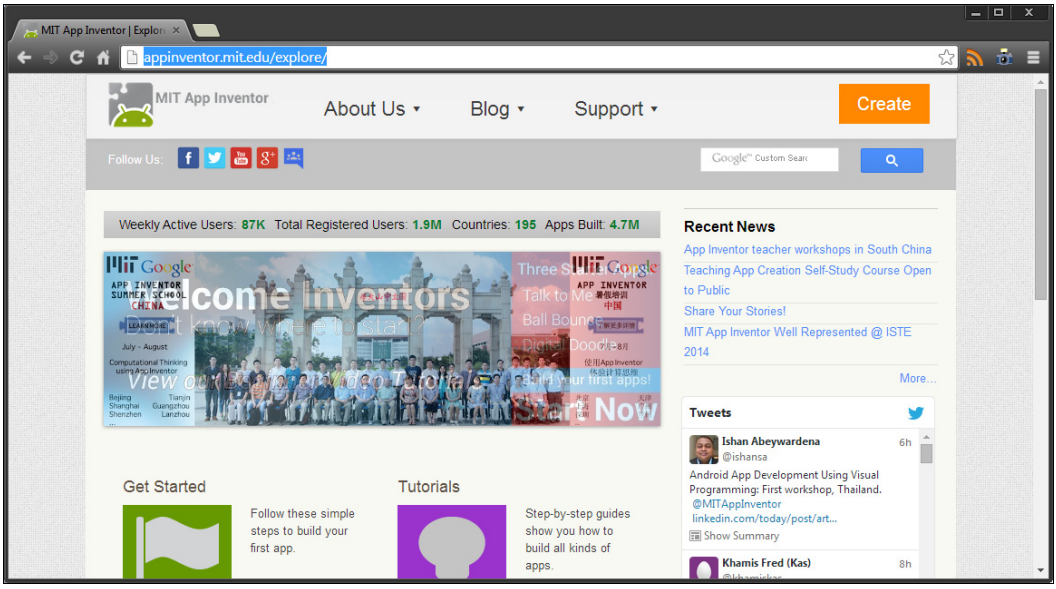


Рис. П.1. Сайт **appinventor.mit.edu**

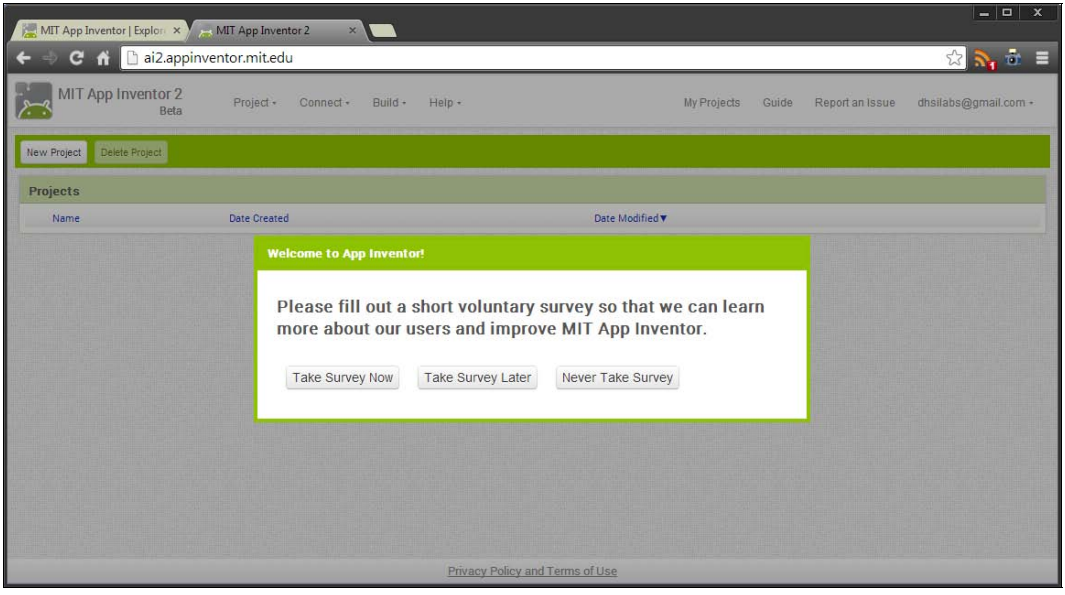


Рис. П.2. Просмотреть обзор?

Ознакомьтесь с приведенными ссылками (откройте их в новом окне или новой вкладке) и нажмите кнопку **Continue** — вы увидите список созданных вами проектов. Понятное дело, список поначалу будет пуст, поскольку вы еще не успели ничего создать (рис. П.4).

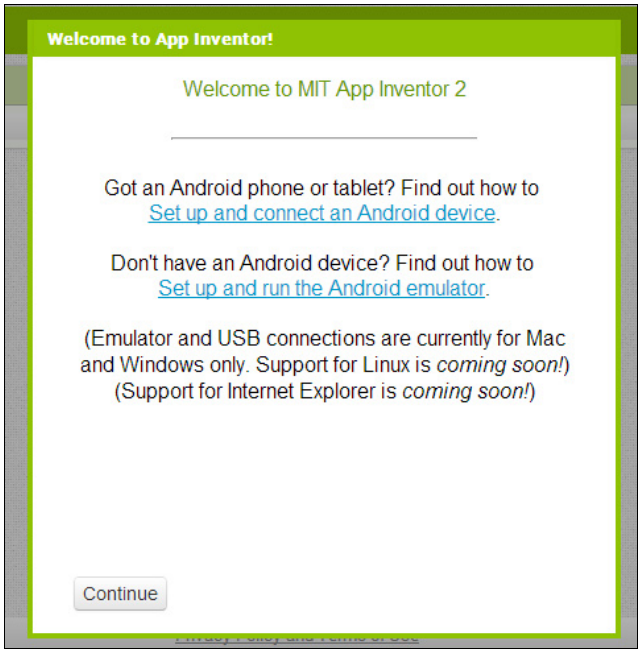


Рис. П.3. Нажмите кнопку **Continue**

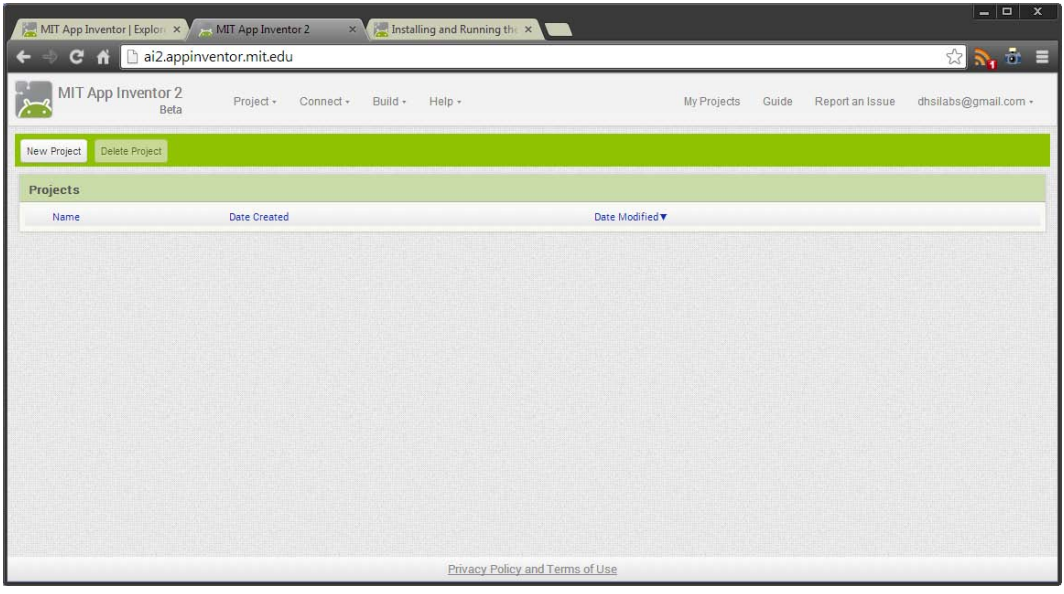


Рис. П.4. Пустой список проектов

Нажмите кнопку **New Project** для создания нового проекта. Введите название проекта и нажмите кнопку **OK** (рис. П.5). Имя проекта не должно содержать пробелов и символов национальных алфавитов, должно начинаться с буквы, может содержать цифры и знак подчеркивания.

Рис. П.5. Создание нового проекта

Далее вы увидите саму среду (рис. П.6), описание которой приводится в следующем разделе.

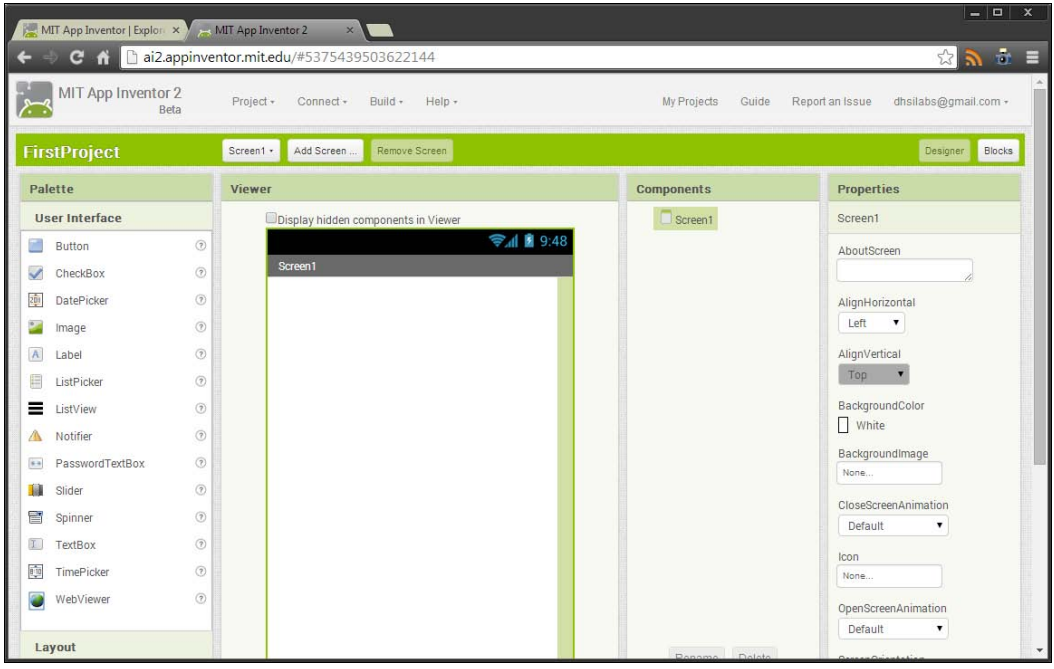


Рис. П.6. Среда App Inventor

П.3. Основной экран App Inventor

Прежде, чем приступить к созданию первого приложения, давайте исследуем меню среды App Inventor.

❒ В меню **Project** сосредоточены основные команды управления проектом:

- **My Projects** — выводит список ваших проектов.
- **Start new project** — создает новый проект.
- **Save project** — сохраняет проект.
- **Save project as** — сохраняет проект под другим именем.

- Меню **Connect** содержит команды подключения к устройству (**USB**) или к эмулятору (**Emulator**).
- В меню **Build** находятся команды построения проекта. Так, команда **App (save .apk to my computer)** сохраняет APK-файл созданного приложения на ваш компьютер. Далее вы можете запустить его в эмуляторе (см. *главу 2*) или установить в своем смартфоне.

Весь экран в режиме проектирования разделен на пять областей:

- **Palette** — отображает палитру компонентов (элементов графического интерфейса): кнопок, надписей, полей ввода и т. д.;
- **Viewer** — область предварительного просмотра и проектирования графического интерфейса пользователя. Посмотрите на рис. П.6. Сейчас у нас там есть один экран — **Screen1**. Если вы добавите второй экран (с помощью кнопки **Add Screen**), то кнопка для переключения на редактирование этого второго экрана появится рядом с кнопкой **Screen1**. Переключатель **Display hidden components in Viewer** позволяет отобразить в области **Viewer** невидимые компоненты;
- **Components** — отображает все компоненты, добавленные на экран. Даже если компонент не отображается в области **Viewer**, он все равно будет присутствовать в области **Components**. На рис. П.6 видно, что на экран 1 (**Screen1**) не добавлено компонентов (есть только **Screen1**);
- **Media** — эта область отображает медиафайлы;
- **Properties** — свойства выбранного компонента. Например, на рис. П1.6 изображены свойства экрана.

П.4. Проектирование приложения

Добавьте в область **Viewer** два компонента: **Button** (кнопку) и **Label** (надпись), чтобы компоненты в областях **Viewer** и **Components** отображались, как показано на рис. П.7.

Затем щелкните на надписи и в области **Properties** очистите свойство **Text** — это текст надписи. После этого надпись исчезнет из **Viewer** — точнее, она там останется, но так как вы очистили ее текст, она станет не видна. Чтобы вновь редактировать ее свойства, вам нужно будет щелкнуть по компоненту **Label1** в области **Components**.

Итак, вы научились изменять свойства компонента! Теперь опять выберите компонент **Label1** и присвойте свойству **Text** какой-нибудь текст, — например, `Hello!` Ведь по традиции первая программа должна содержать это слово!

Что ж, полдела сделано — мы расположили компоненты на экране. Теперь приступим к описанию действий программы. Наша программа не будет слишком оригинальной — при запуске на экране будут видны кнопка и надпись. По нажатию на кнопку надпись с экрана исчезнет. Пусть это и не самое сложное Android-приложение, но как первый опыт — вполне сгодится.

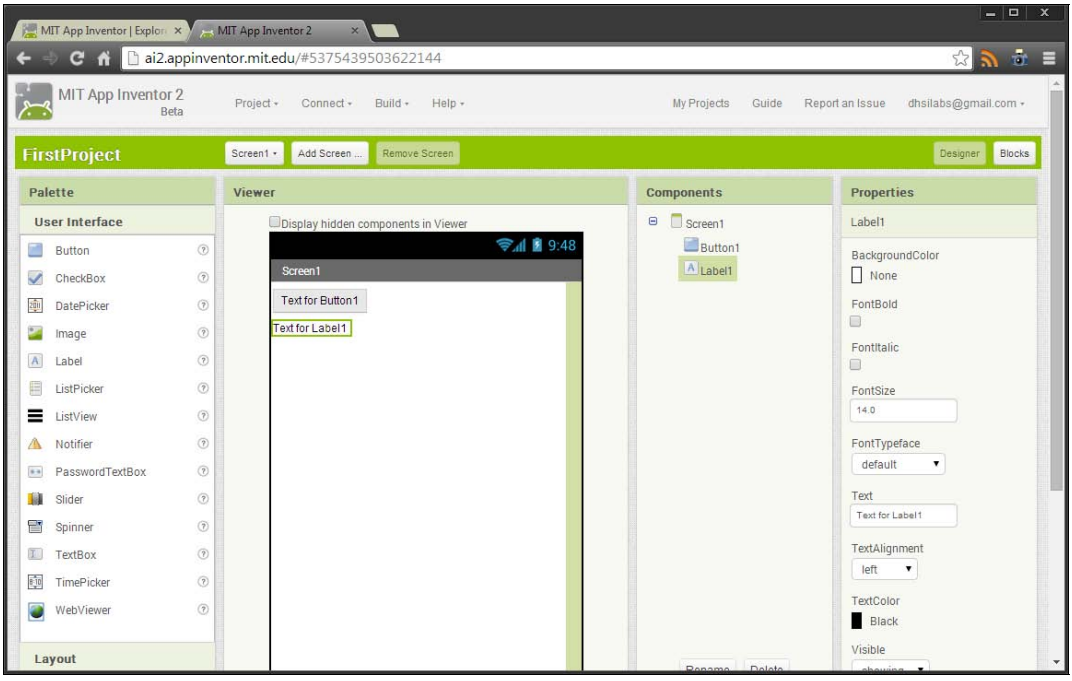


Рис. П.7. Добавленные компоненты

Нажмите кнопку **Blocks** для открытия редактора блоков. Начнем собирать пазлы! Сначала область редактора блоков будет пуста. Подумаем, что должно происходить? Мы же нажимаем на кнопку — следовательно, нам нужно действие-обработчик нажатия на кнопку.

Ваши компоненты отображаются в секции **Screen1** (или **Screen2**, **Screen3** — в зависимости от экрана). Выберите **Button1** — вы увидите все действия, доступные для выбранного компонента, т. е. для кнопки (рис. П.8).

Выберите пазл **when Button1.Click do** и добавьте его в свободную область редактора блоков (рис. П.9). Поясню, что мы сделали, — выбрали обработчик нажатия кнопки **Button1**. Теперь осталось определить, что произойдет, когда будет нажата эта кнопка. Нам ведь надо скрыть надпись. Значит, первым делом в **Screen1** нужно выбрать надпись **Label1**, чтобы посмотреть, что можно с ней сделать.

Выберите действие **set Label1 Visible to**. Это действие устанавливает свойство **Visible** надписи **Label1** в определенное значение. У вас должна получиться конструкция, изображенная на рис. П.10.

Чтобы скрыть надпись, нужно установить ее свойство **Visible** в **false**. Вот и последний пазл нашей программы — из группы **Built-in** выбрать группу **Logic**, а из нее — значение **false**. У вас должна получиться конструкция, изображенная на рис. П.11.

Если нужно удалить какой-то блок, просто перетащите его в корзину. По сути, наша первая программа готова. Выберите команду меню **App (save .apk to my computer)** из меню **Build**, а затем — каталог, в который нужно сохранить APK-

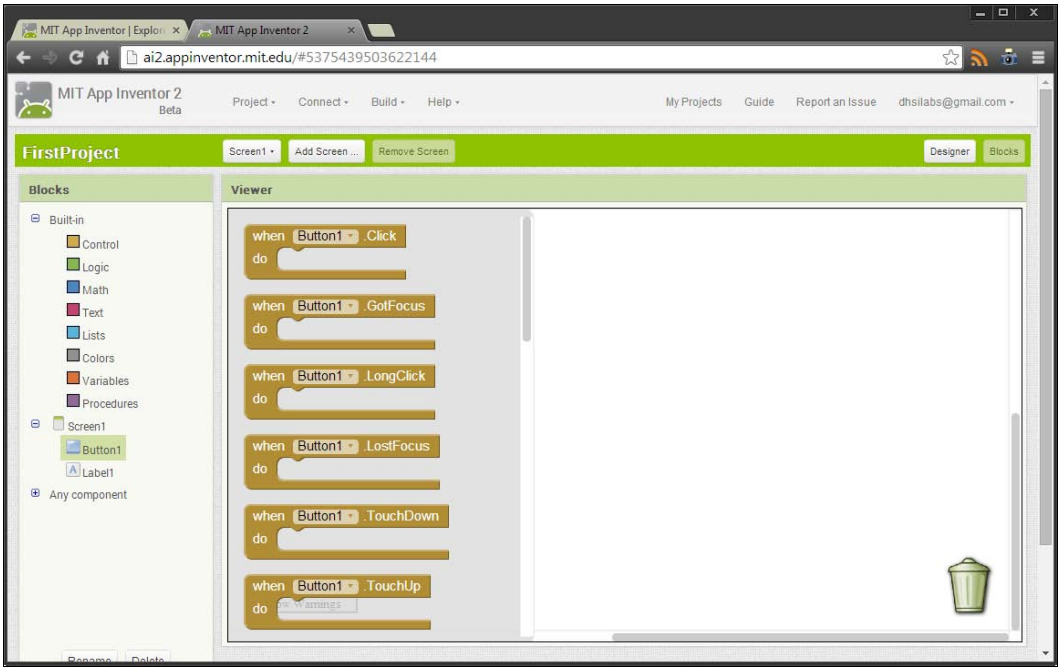


Рис. П.8. Редактор блоков

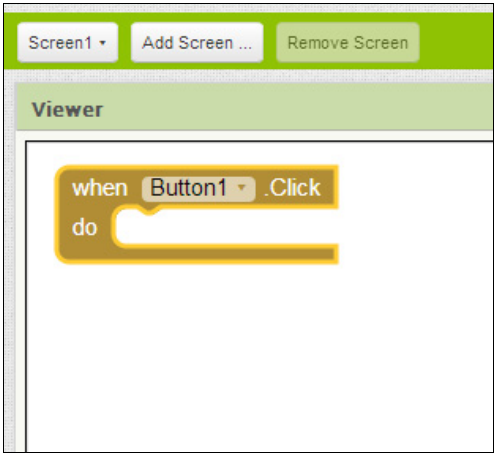


Рис. П.9. Создание обработчика для кнопки **Button1**

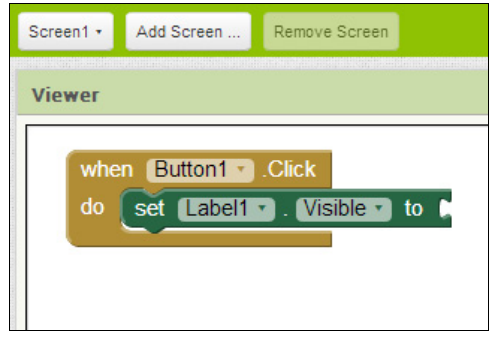


Рис. П.10. Определение действия для кнопки **Button1**

файл. Далее или протестируйте его в эмуляторе, или установите в смартфоне. Напомню, что для тестирования программы в эмуляторе нужно открыть командную строку, перейти в каталог platform-tools и ввести команду:

```
adb install путь\FirstProject.apk
```

На рис. П.12 приведена наша первая программа, созданная средствами App Inventor.

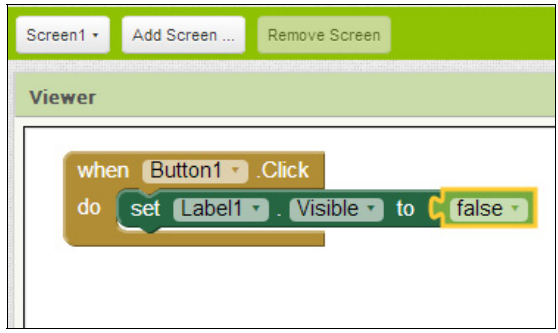


Рис. П.11. Наша программа

Рис. П.12. Приложение, созданное в App Inventor

Как видите, создать программу с помощью App Inventor довольно просто. Но посмотрите на объем полученного APK-файла — 1,3 Мбайт. И это простое приложение с одной кнопкой и надписью. Столько же «весит» полноценное приложение, написанное на Java в среде Eclipse...

Предметный указатель

A

B

K

M

П

Р

С

У

Ф

Ш

Э