Основы программирования для мобильных устройств

1. Установка и настройка среды программирования ADT Bundle **Цель лабораторной**

работы:

Установка и настройка среды программирования ADT Bundle

Задачи лабораторной работы: Установка и настройка среду программирования ADT

- Создать первое приложение **Hello, world!**
- Научиться запускать приложение на эмуляторе мобильного устройства

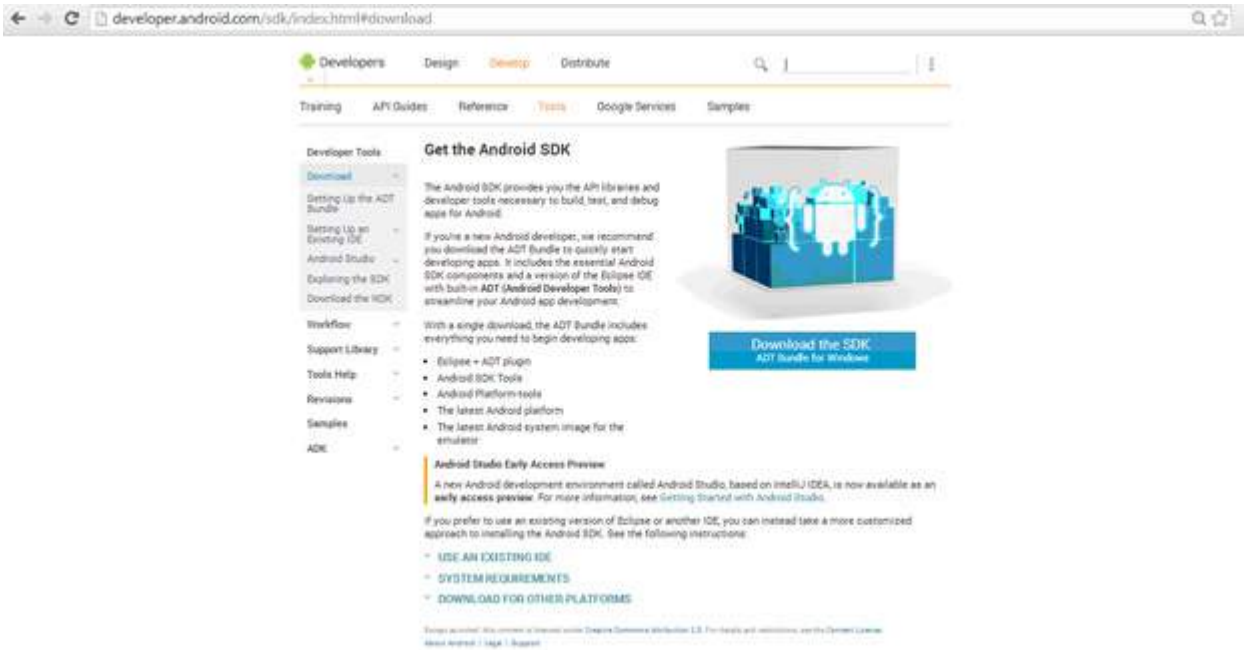
2.1 Введение

Лабораторная работа посвящена описанию работы в среде ADT Bundle (AndroidIDE). В работе рассказывается о программном обеспечении, которое необходимо скачать с официальных сайтов, установить и настроить, создании и запуске простейшего приложения на эмуляторе и мобильном устройстве. В связи с тем, что отладка на устройствах сопряжена с рядом трудностей, приводится подробная инструкция по настройке устройств и операционной системы Windows (версия не ниже 7).

2.2 Установка среды

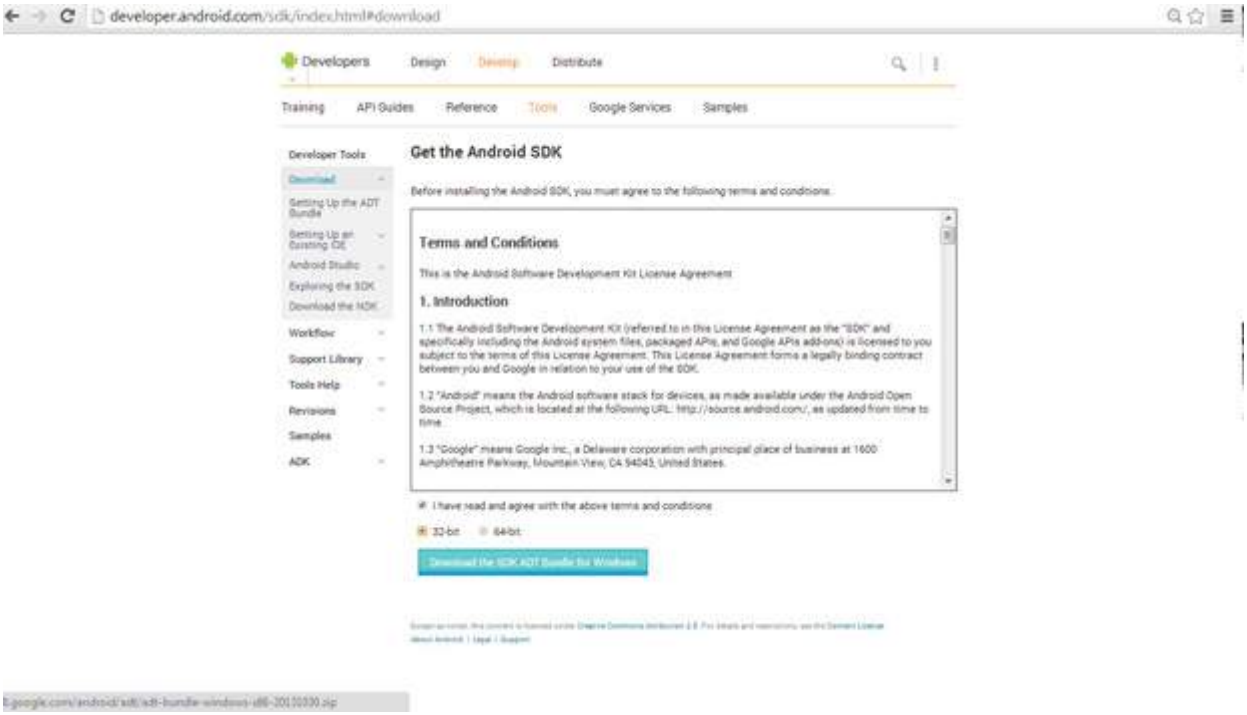
Большинство приложений для OS Android написано на Java. Одной из самых популярных сред разработки является Eclipse (для неё также необходим JDK) с установленным плагином ADT и AndroidSDK. Раньше приходилось ставить все компоненты отдельно. Сейчас появилась версия среды Eclipse с уже настроенными дополнениями - ADT Bundle. Здесь есть минимум инструментов, необходимый для разработки приложений. С этой версией мы и будем работать. Однако в ней есть далеко не всё, поэтому, если при разработке какого-либо проекта вам потребуются инструменты, не входящие в ADT Bundle, вы можете скачать их с сайта разработчиков и дополнить свою среду.

Скачать среду можно с сайта для разработчиков Android (http://developer.android.com/sdk/index.html).



увеличить изображение
Рис. 2.1. Сайт разработчика

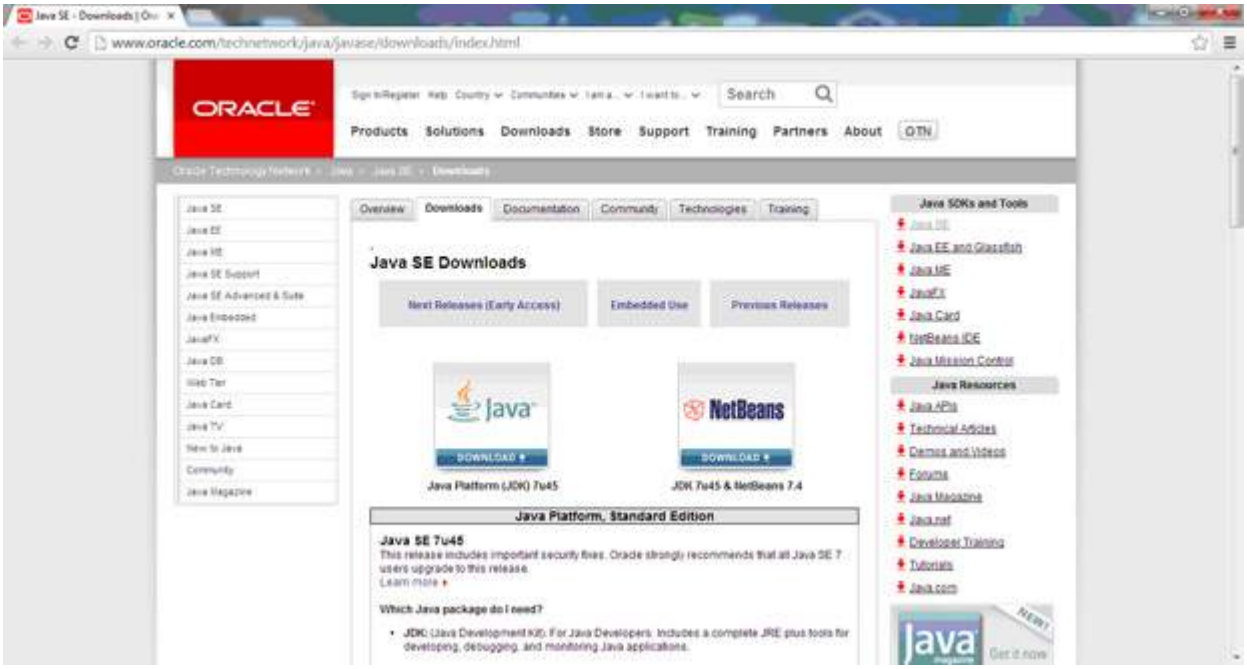
Для того, чтобы скачать среду необходимо принять условия лицензионного соглашения и выбрать вашу версию Windows (32-bit или 64-bit).



увеличить изображение
Рис. 2.2. Скачивание среды

После скачивания распакуйте архив в ту папку, где собираетесь работать (среда не требует специальной установки). После распаковки зайдите в папку и запустите Eclipse. Здесь возможна небольшая проблема: если у вас не установлен JDK, среда не запустится и потребует указать путь к папке с JDK или установить его.

Скачать JDK можно с сайта Oracle (http://www.oracle.com)



[увеличить изображение]
Рис. 2.3. Сайт компании Oracle

Чтобы скачать JDK нужно сначала принять условия лицензионного соглашения, а затем выбрать нужную версию.

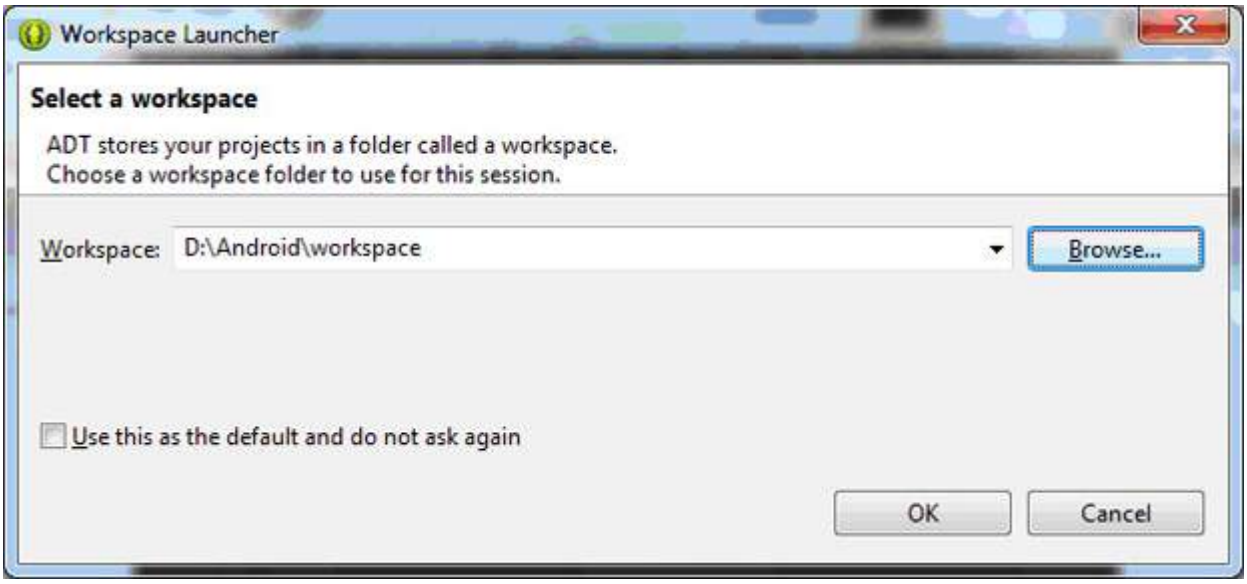
[увеличить изображение]
Рис. 2.4. Скачивание JDK

После скачивания запустите setup-файл и установите JDK.

увеличить изображение
Рис. 2.5. Установка JDK

После установки JDK среда должна запуститься.

Далее вам необходимо выбрать (или создать новое) рабочее пространство, т.е. место, где будут находиться ваши проекты. Если поставить галочку, то это рабочее пространство будет выбираться по умолчанию, а противном случае это окно будет появляться при каждом запуске Eclipse.



увеличить изображение
Рис. 2.6. Выбор рабочего пространства

Затем появляется окно, в котором разработчики предлагают отправлять статистику для дальнейшего улучшения SDK. Вы можете согласиться или отказаться.

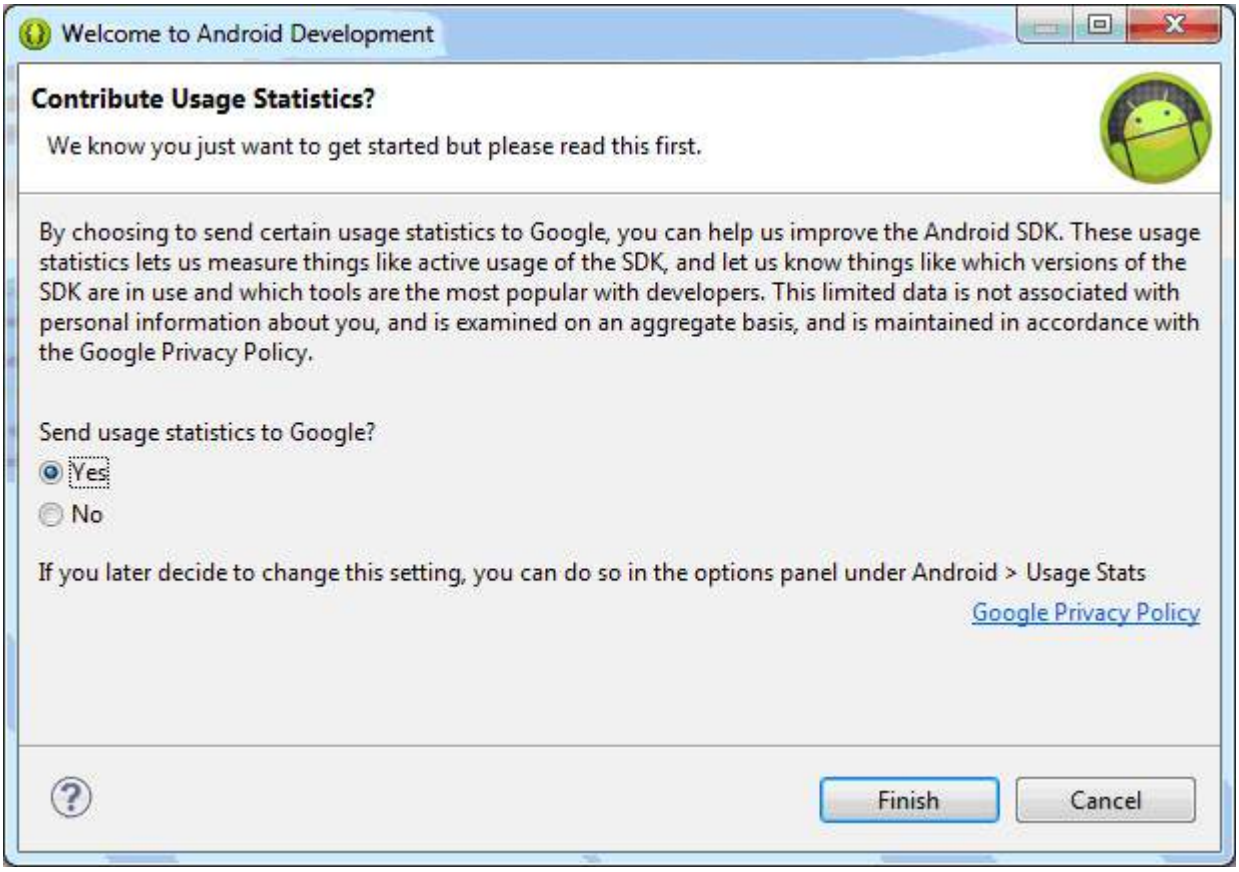
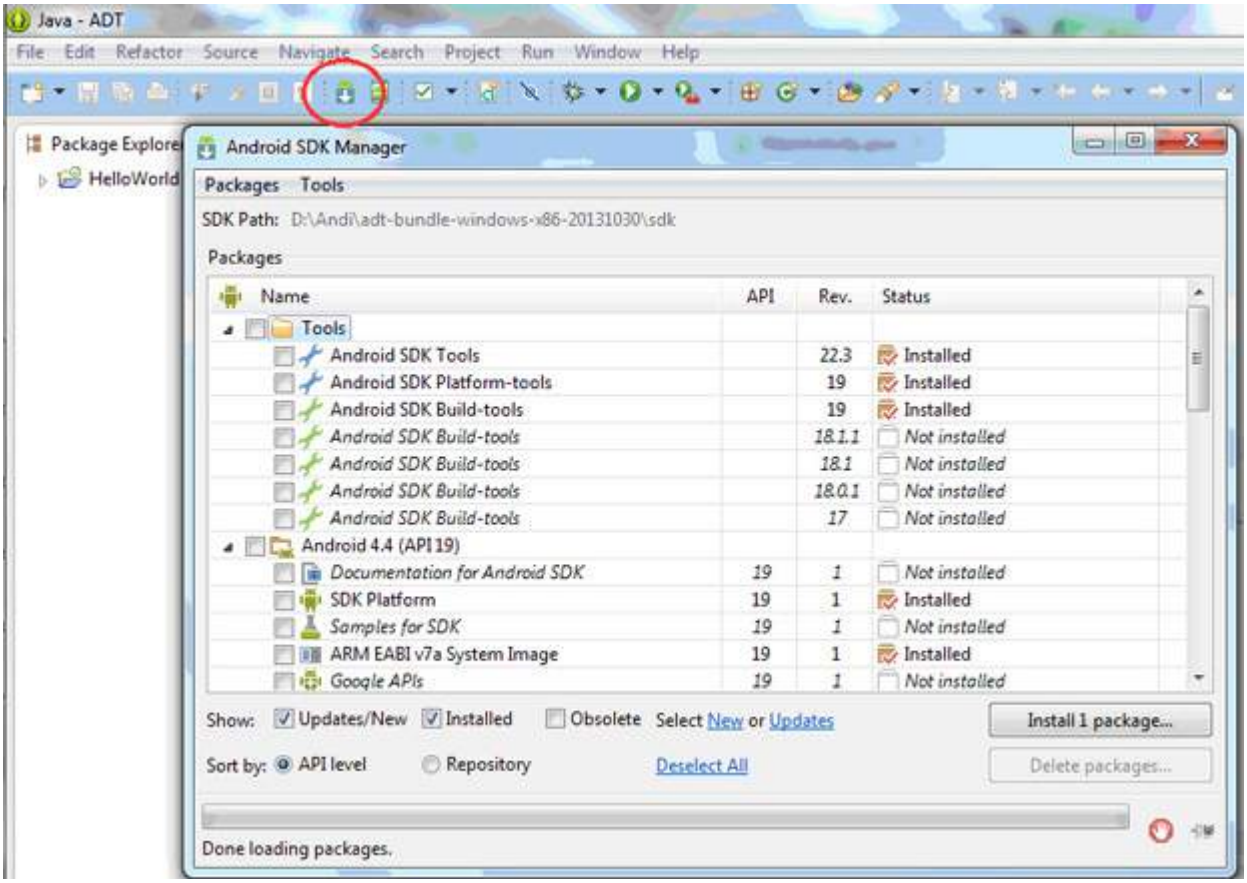


Рис. 2.7. Отправка статистики

Обратите внимание на значок AndroidSDK Manager, находящийся на панели инструментов (его также можно найти в меню Window). С его помощью вы сможете добавлять в свою среду новые инструменты.

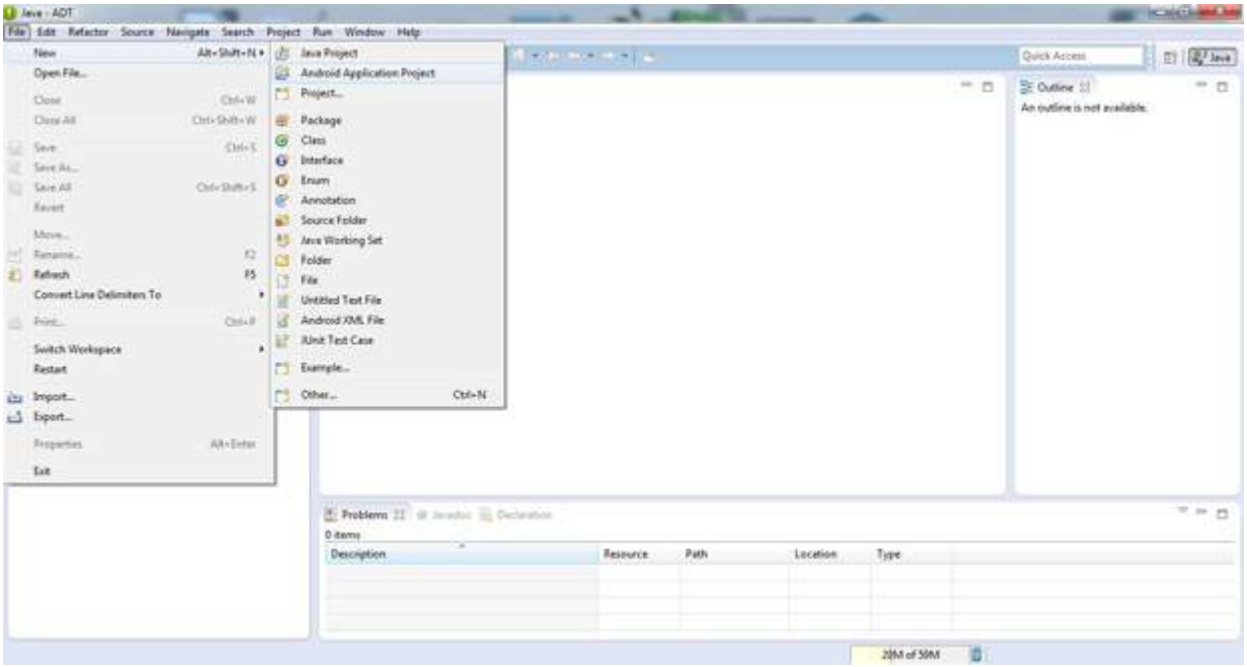


[увеличить изображение]
Рис. 2.8. Android SDK Manager

2.3 Создание проекта

Итак, наконец, мы подошли к самому главному - созданию проекта.

Чтобы создать проект, зайдите в меню **File->New->Android Application Project**.



увеличить изображение
Рис. 2.9. Создание проекта

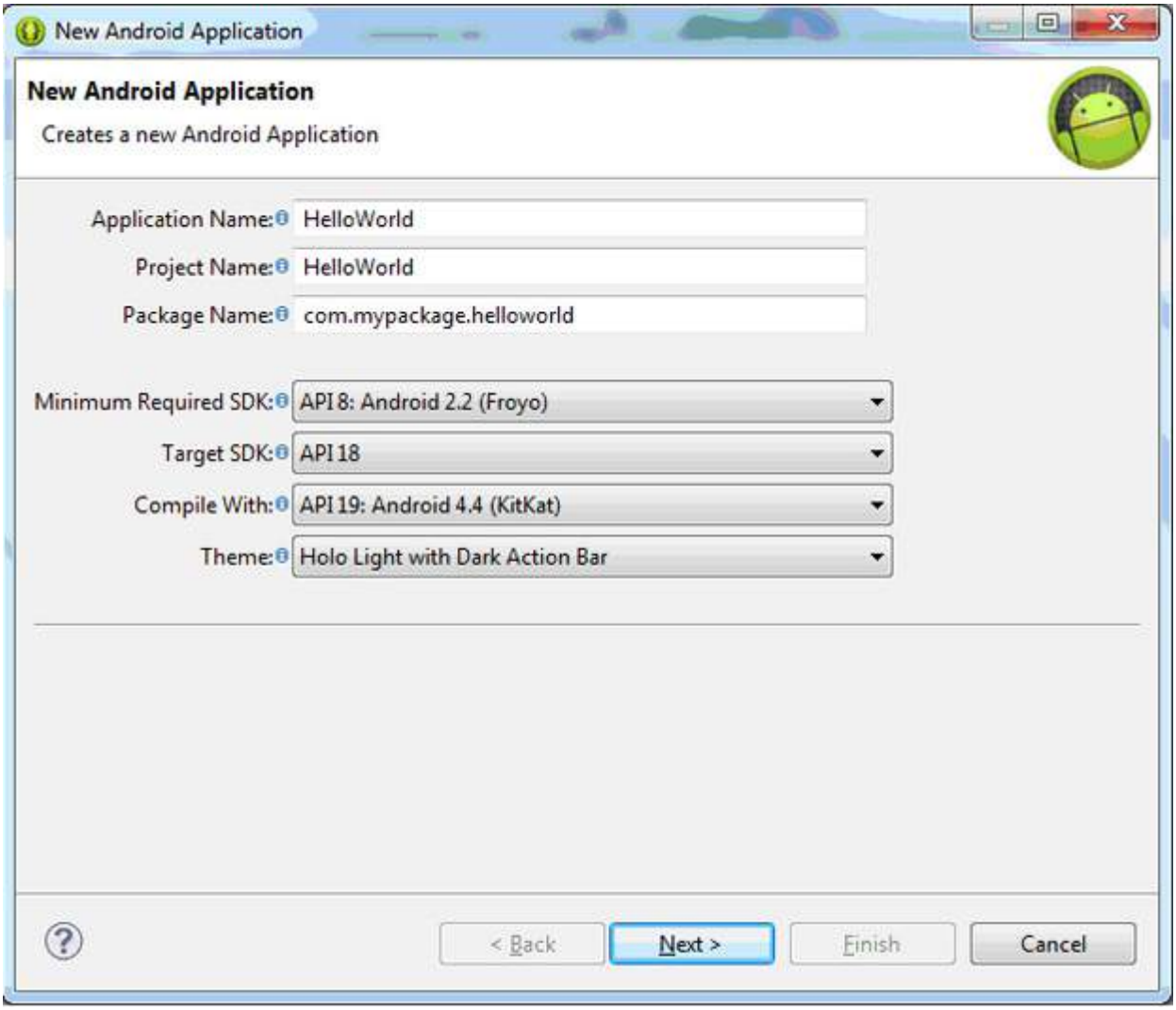
В появившемся окне обязательно нужно прописать имя приложения, имя проекта, а также имя пакета (package). Лучше не оставлять его именем example, т.к. пакет с таким именем нельзя разместить в GooglePlay. Конечно, учебные приложения туда не загружают, однако, следует иметь это в виду на будущее.

MinimumRequired SDK - минимальная версия Android, которую будет поддерживать приложение. Чаще всего по умолчанию указывается версия 2.2, чтобы поддерживать как можно больше устройств. Если определенная функция вашего приложения работает только на более новых версиях Android, и это не является критическим для основного набора функций приложения, вы можете включить ее в качестве опции на версиях, которые поддерживают его.

Target SDK - версия Android, под которую будет написано ваше приложение; определяет максимальную версию Android, на которой вы тестировали приложение. Это нужно для режимов совместимости.

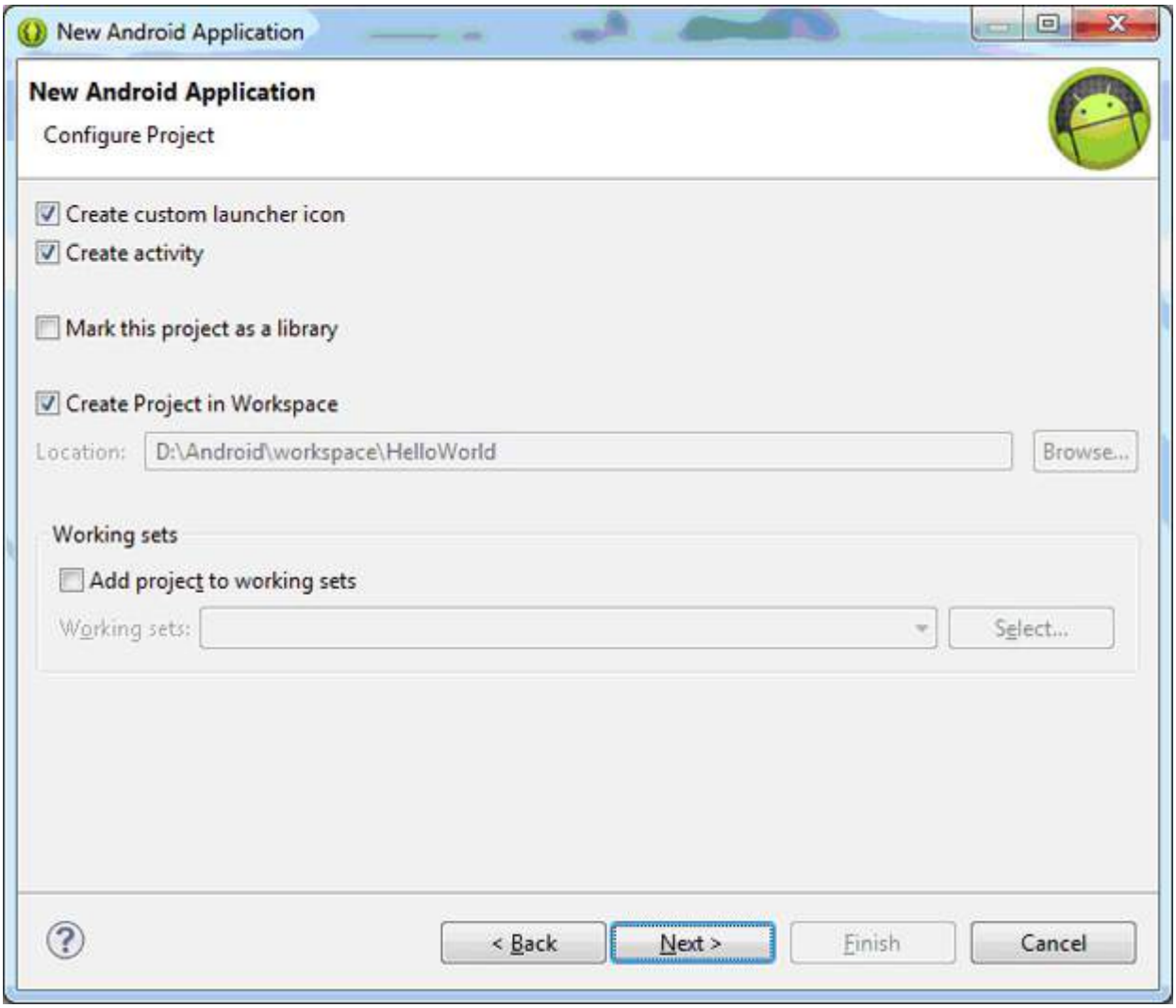
CompileWith определяет, возможности какой версии Android будет использовать приложение.

Оставьте пока установки, заданные по умолчанию в качестве значений для этого проекта.



увеличить изображение
Рис. 2.10. Наименование проекта

Следующее окно можно пропустить без изменений.



увеличить изображение
Рис. 2.11. Конфигурация проекта

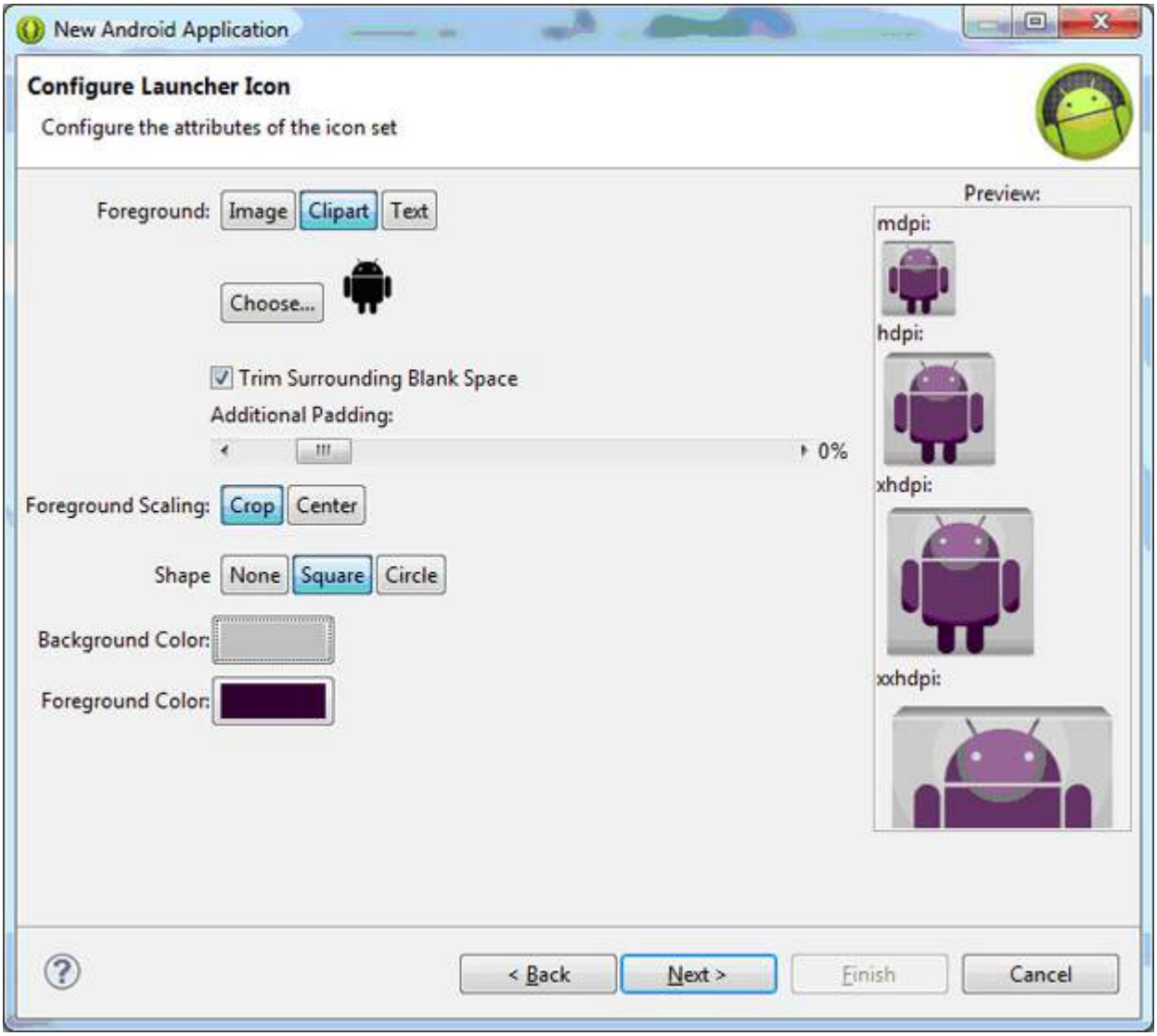
Createcustomlauncherіcon - создать значок приложения.

Createactivity - создать Activity (активность, деятельность).

Mark this project as library - создатьпроект, какбиблиотеку. Сейчас в этом нет необходимости, наше приложение в других проектах использоваться не будет.

Create Project in Workspace - создатьпроектвпапке Workspace. В этой папке будут храниться все наши проекты.

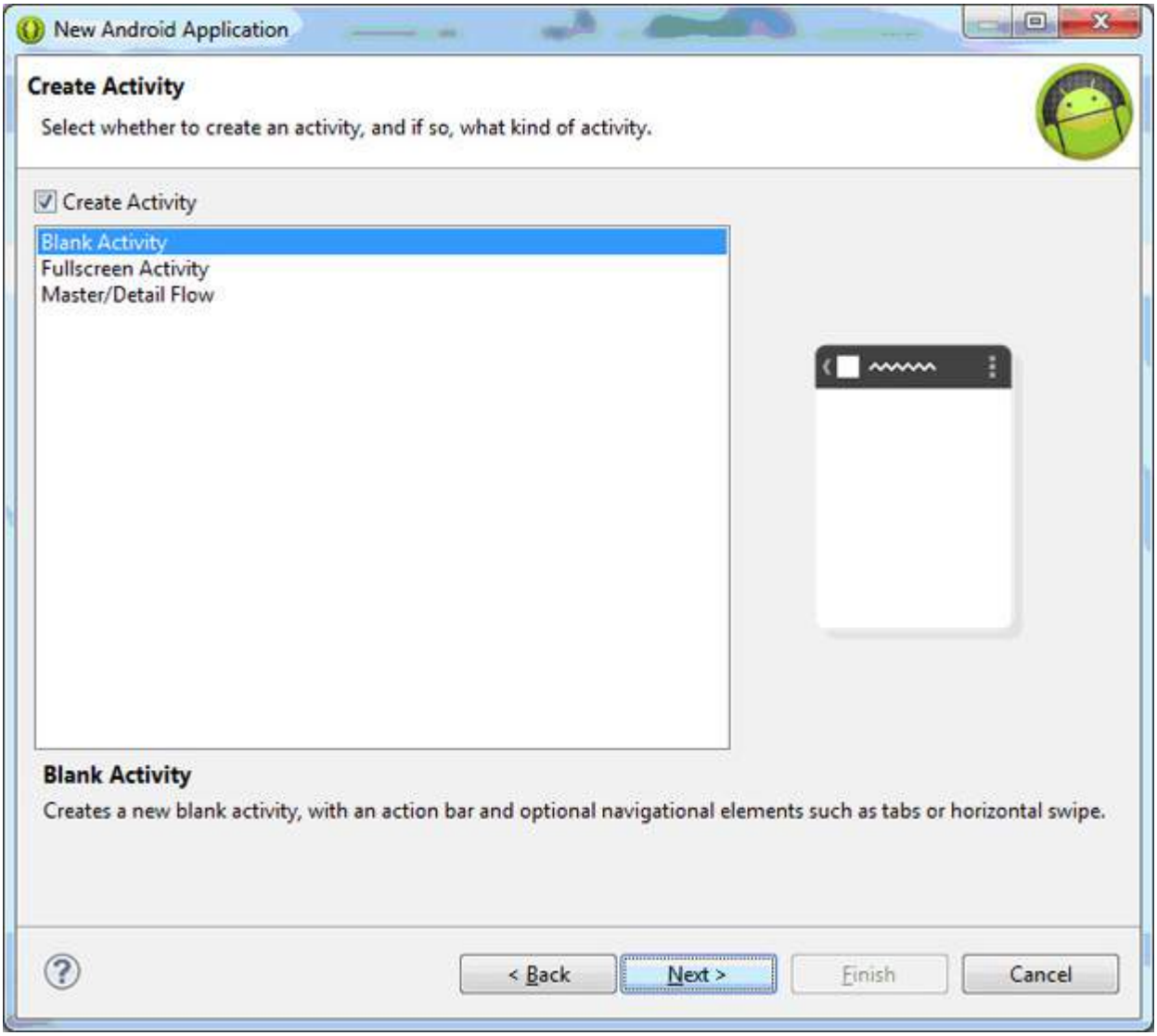
Следующий этап - создание иконки. Можете оставить стандартную или создать свою собственную. В нашем примере изменена цветовая гамма, форма, а также выбрана фигурка из клипарта.



увеличить изображение
Рис. 2.12. Создание иконки приложения

Большинство приложений на Android имеют свой экран (форму, окно), которое называется активностью или деятельностью (Activity).

Следующее два окна создают пустую активность. В первом ничего пока менять не нужно. Во втором вы можете переименовать свою активность (в приложении их может быть несколько).

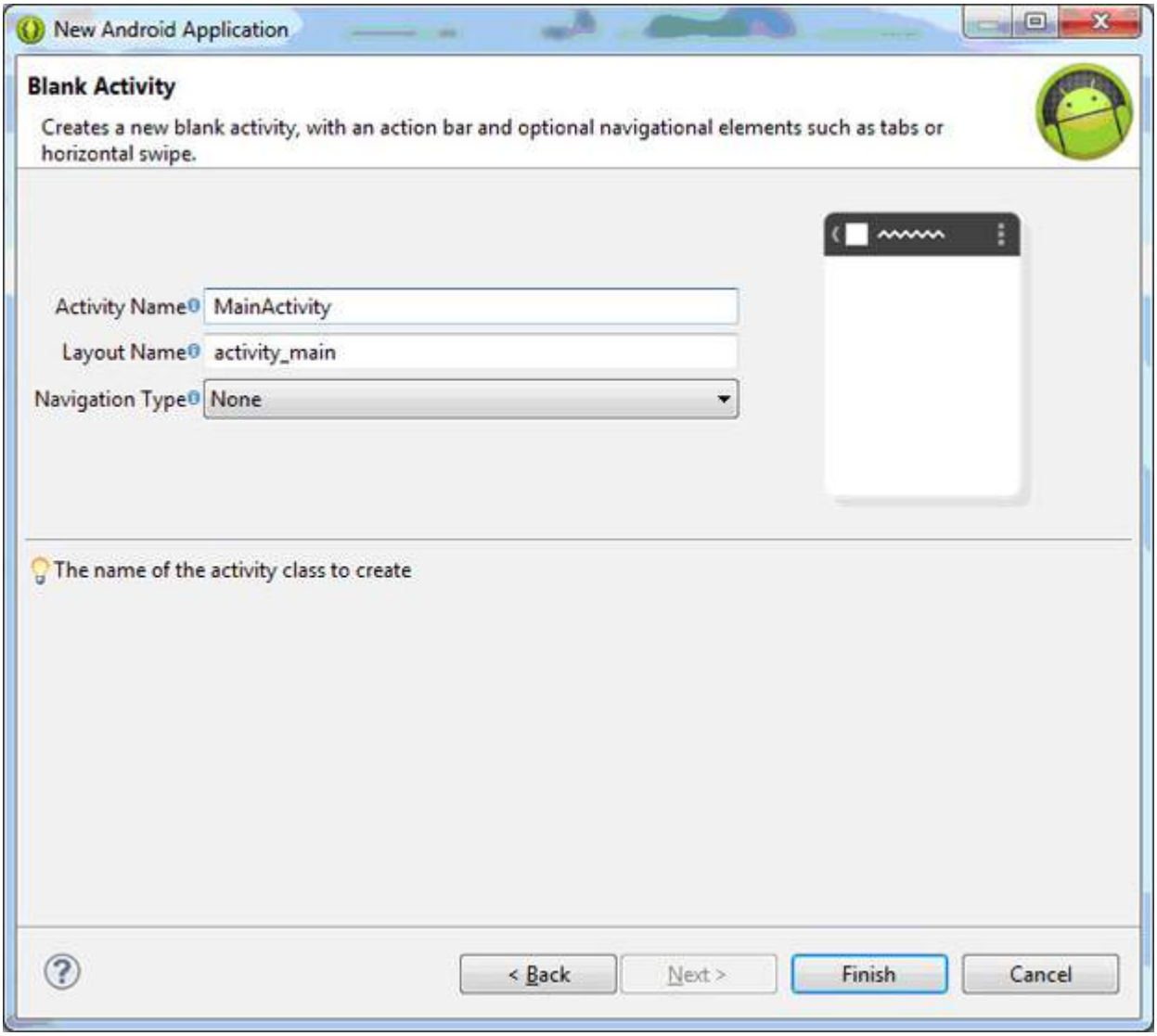


увеличить изображение
Рис. 2.13. Создание активности

BlankActivity - шаблон, предназначенный для мобильных телефонов.

FullscreenActivity - шаблон, позволяющий растянуть приложение на весь экран (без навигационной панели и статус-бара).

Master/DetailFlow - шаблон, предназначенный для планшетных компьютеров.

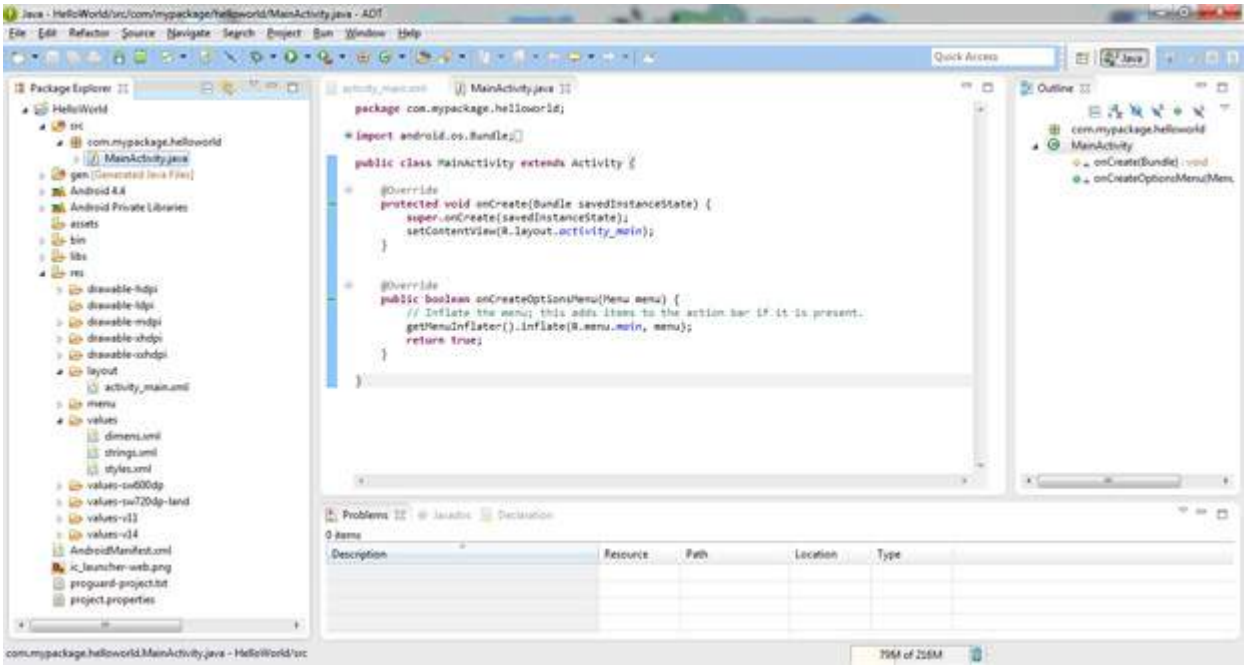


[увеличить изображение]
Рис. 2.14. Переименование активности

Итак, вы создали свой первый проект. Конечно, это всего лишь встроенное в среду приложение для проверки корректной установки инструментария, однако множество приложений создаются именно из него.

Посмотрим на его структуру. Она показана в области слева.

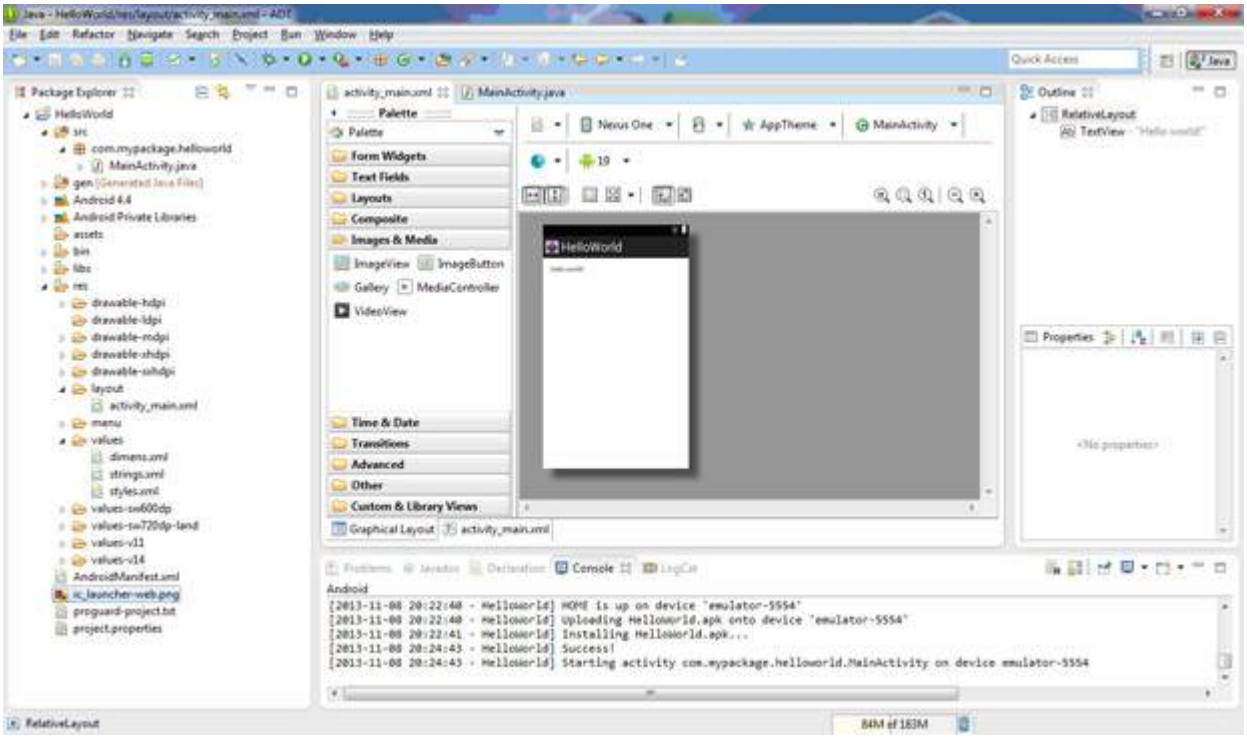
В первую очередь нас интересует файл активности. Он находится в папке **src** в вашем пакете. Он имеет расширение .java.



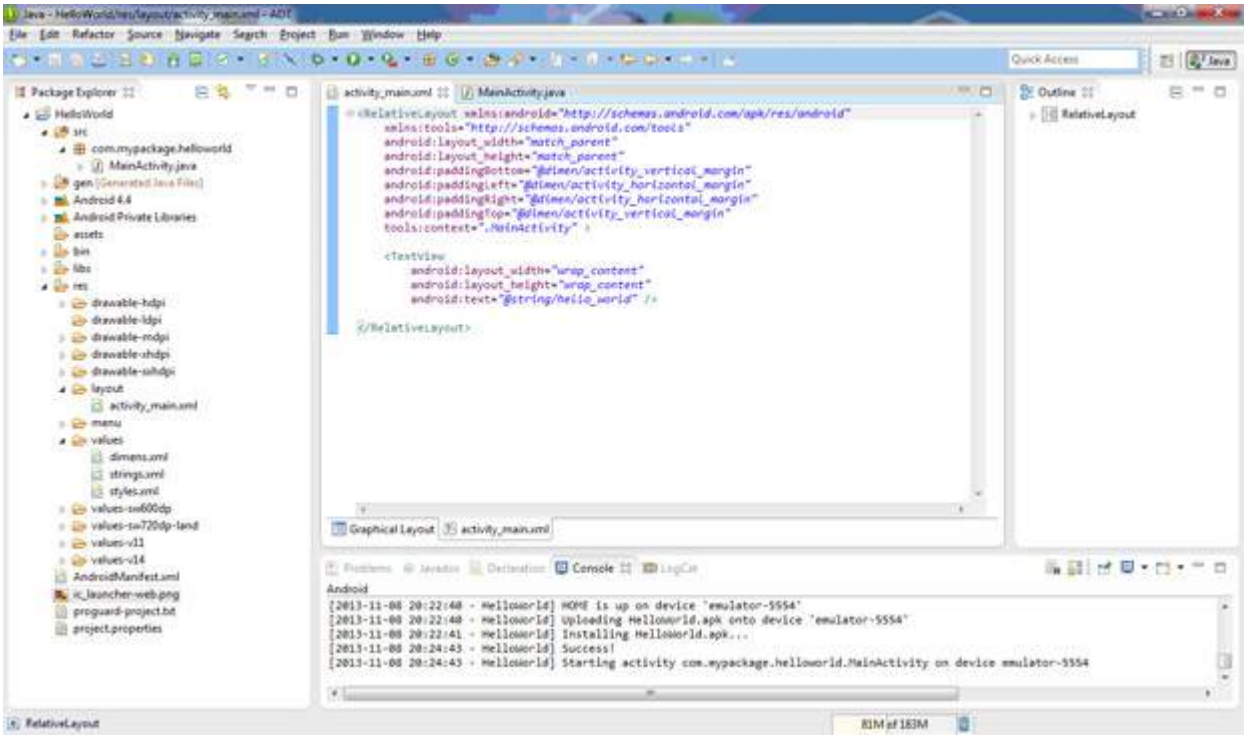
увеличить изображение
Рис. 2.15. Активность

В папке **res** в подпапке **layout** находится xml-файл, который является оболочкой нашей активности. Именно этот файл будет виден на экране устройства.

С xml-файлами можно работать как в режиме графического редактора, так и непосредственно редактировать код.



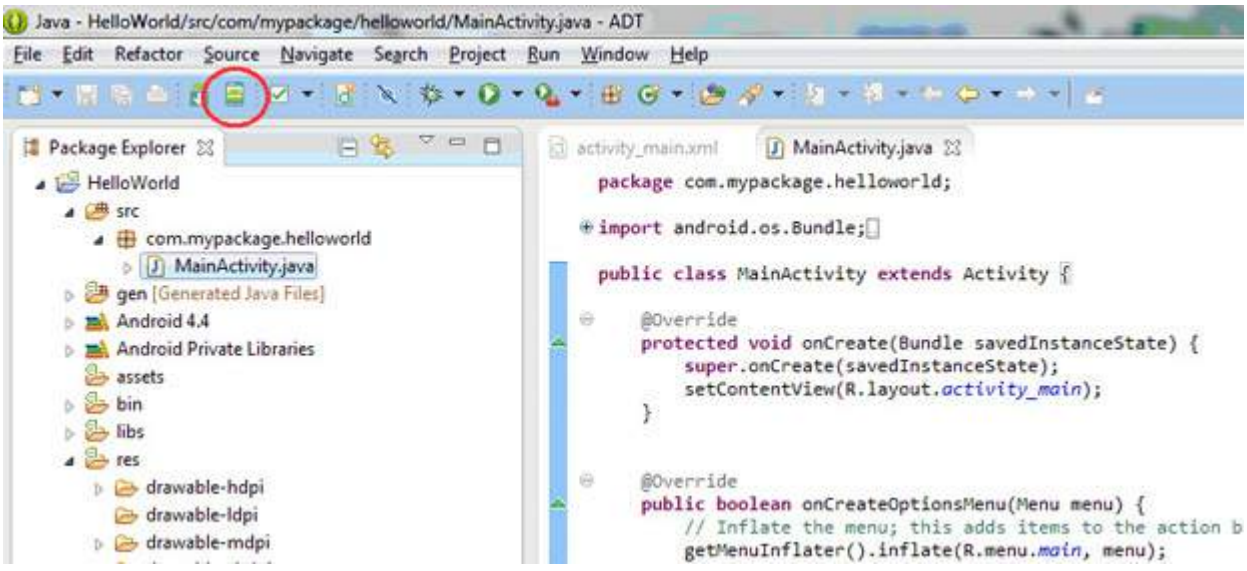
увеличить изображение
Рис. 2.16. Xml-файл. Графический редактор



увеличить изображение
Рис. 2.17. Xml-файл

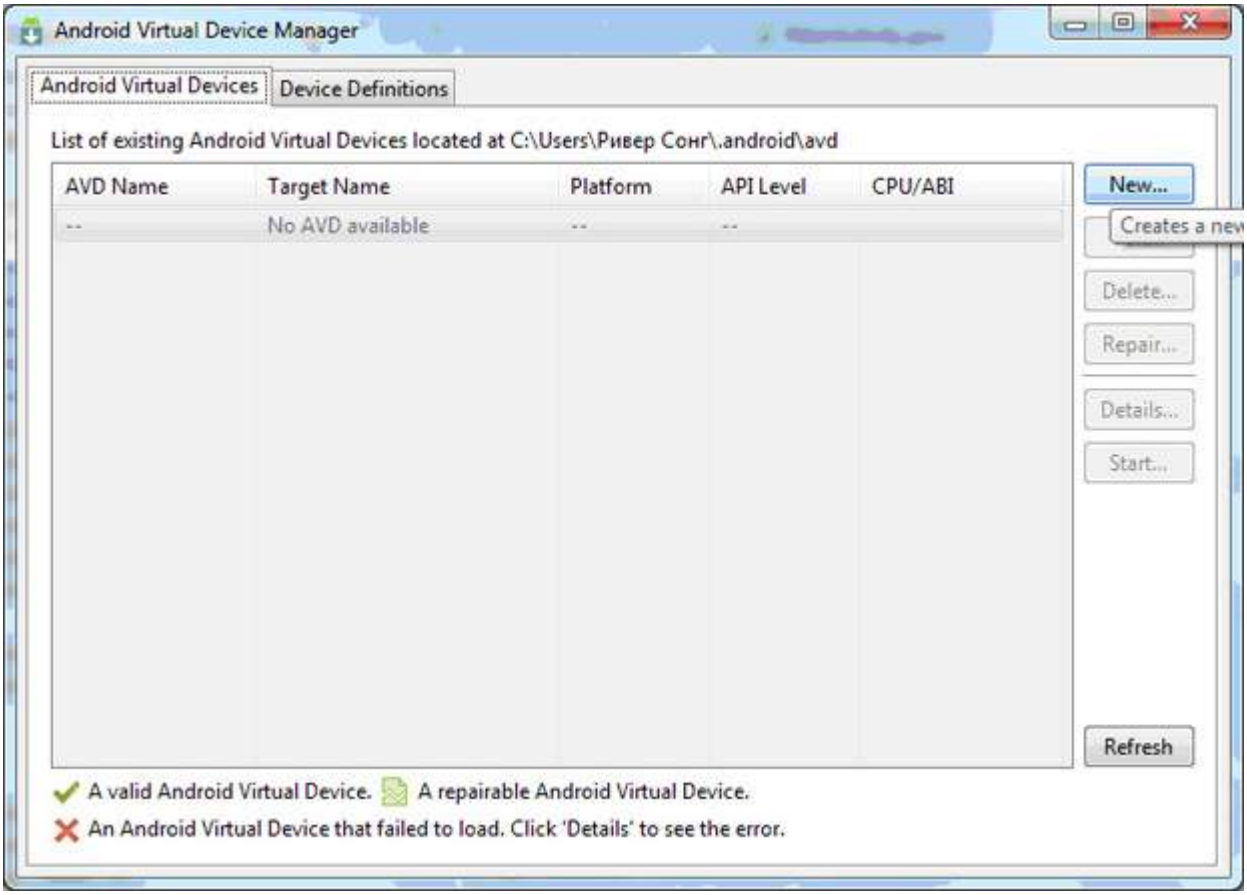
2.4 Запуск проекта на эмуляторе устройства

В первую очередь нужно создать эмулятор устройства. Это можно сделать, нажав на кнопку на панели инструментов, изображающую смартфон. Если кнопки нет на панели, ее можно найти в меню **Window**.



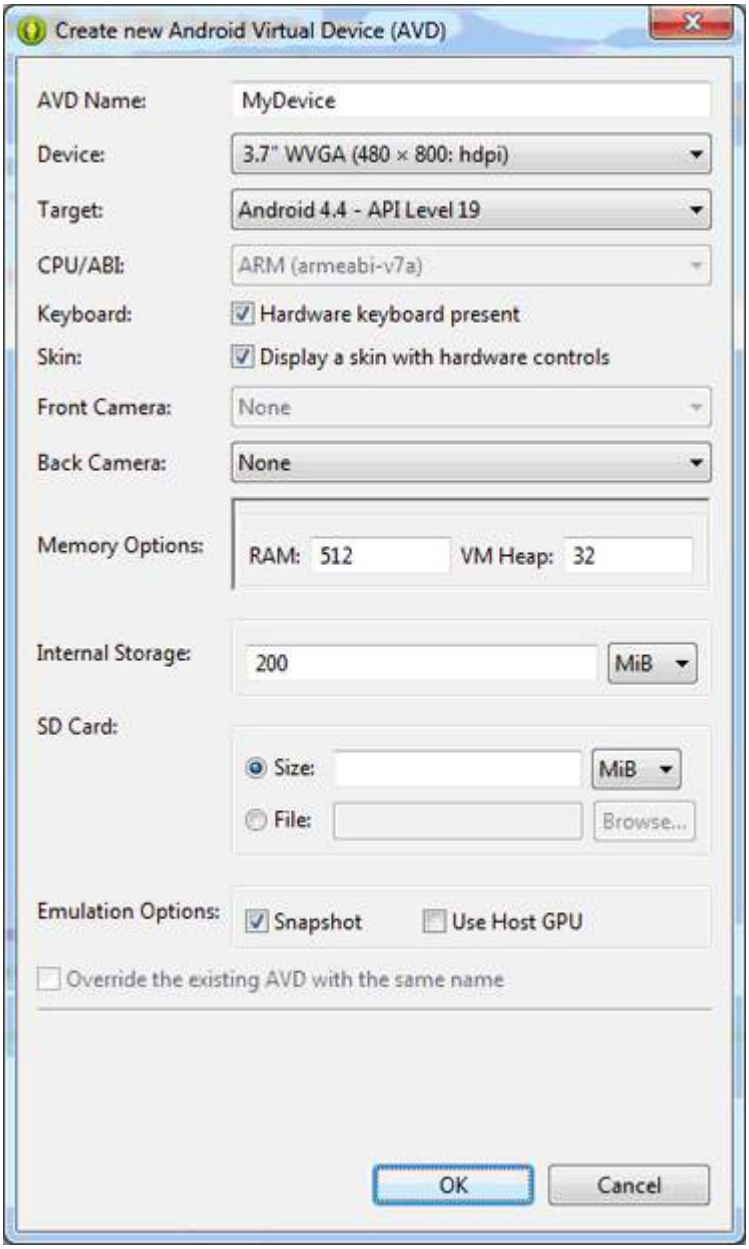
увеличить изображение
Рис. 2.18. Запуск проекта

Откроется AndroidVirtualDeviceManager. Пока в нем нет ни одного виртуального устройства.



увеличитьизображение
Рис. 2.19. Android Virtual Device Manager

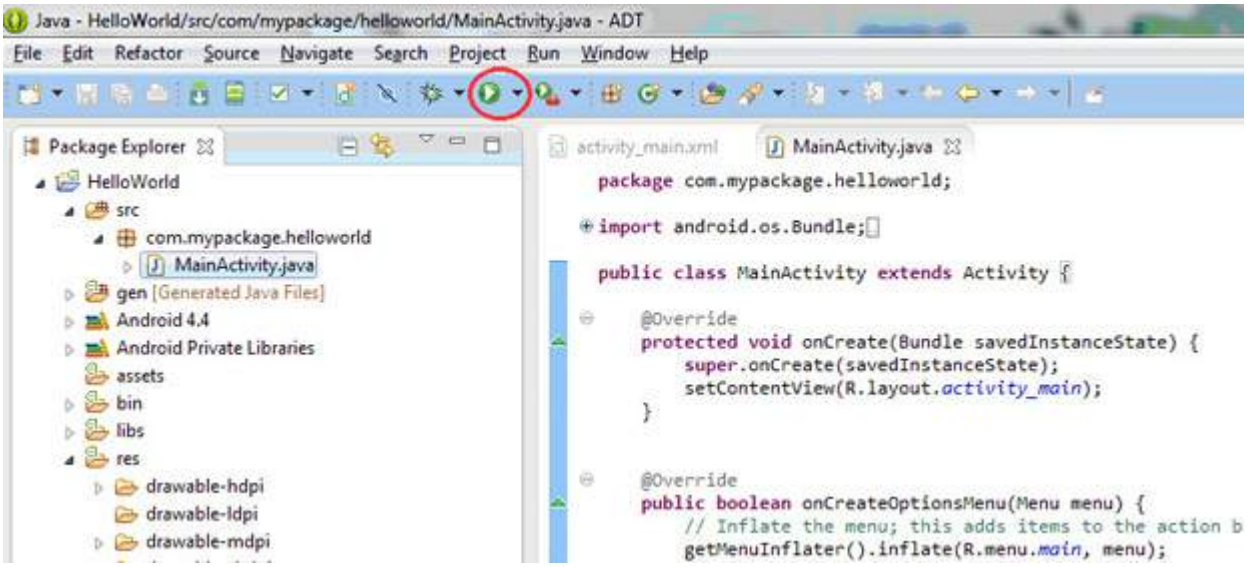
Чтобы создать виртуальное устройство, нажмите кнопку **New**. Появится окно создания. Вам нужно назвать устройство и выбрать обязательные характеристики: **Device** - модель вашего устройства, и **Target** - версия Android. Также можно изменять дополнительные параметры: размер sd-карты, встроенной памяти и т.п.



увеличить изображение
Рис. 2.20. Создание AVD

Теперь можно запускать приложение. Для этого нужно нажать на кнопку **Run** (белый треугольник в зеленом кружочке) на панели инструментов. Проблемы с запуском можно отследить в консоли.

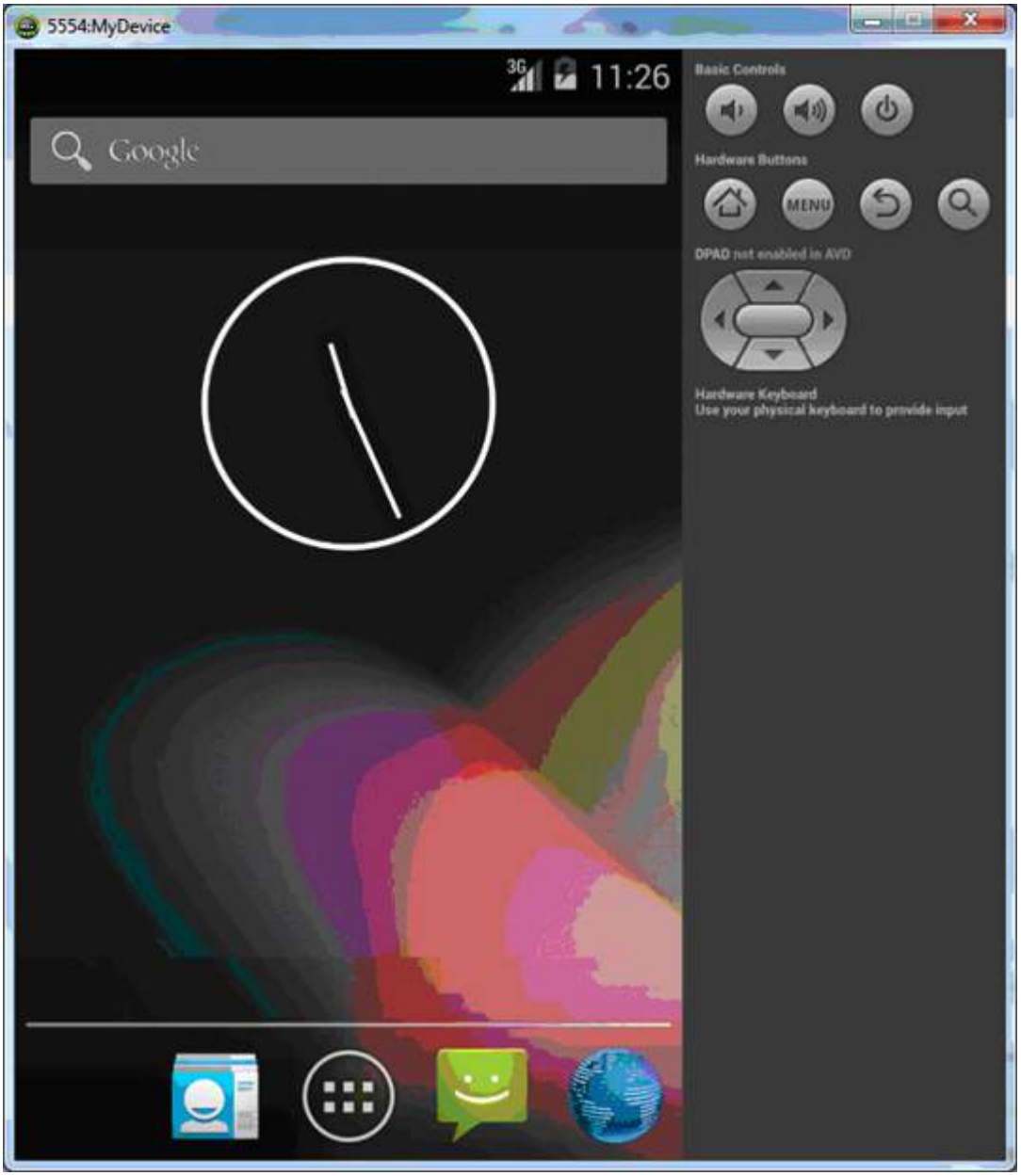
Если приложение не запускается, попробуйте нажать на черный треугольник справа от кнопки **Run**, выбрать **RunConfigurations**, затем во вкладке **Target** выбрать созданное устройство и запустить проект снова.



увеличить изображение
Рис. 2.21. Запуск приложения

Если все сделано правильно, должен запуститься эмулятор. Время запуска зависит от размера оперативной памяти на вашем компьютере. В дальнейшем эмулятор можно не закрывать, приложения будут запускаться в работающем.

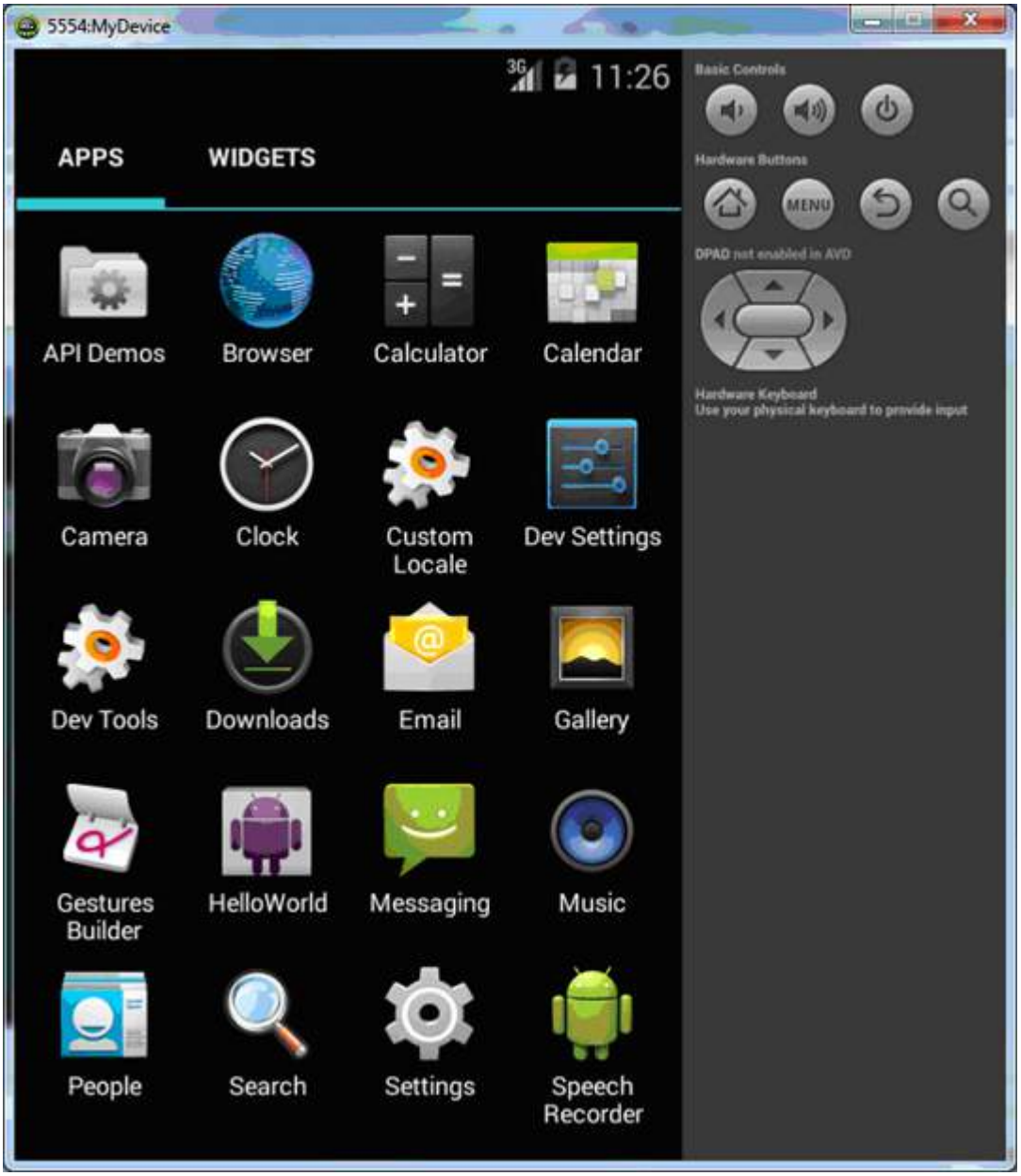
увеличить изображение
Рис. 2.22. Запуск эмулятора



увеличить изображение
Рис. 2.23. Запущенный эмулятор

увеличить изображение
Рис. 2.24. Приложение Hello, world!

Если ваше приложение сразу не запустилось, его можно найти в меню приложений устройства.



увеличить изображение
Рис. 2.25. Меню приложений

2.5 Запуск проекта на устройстве

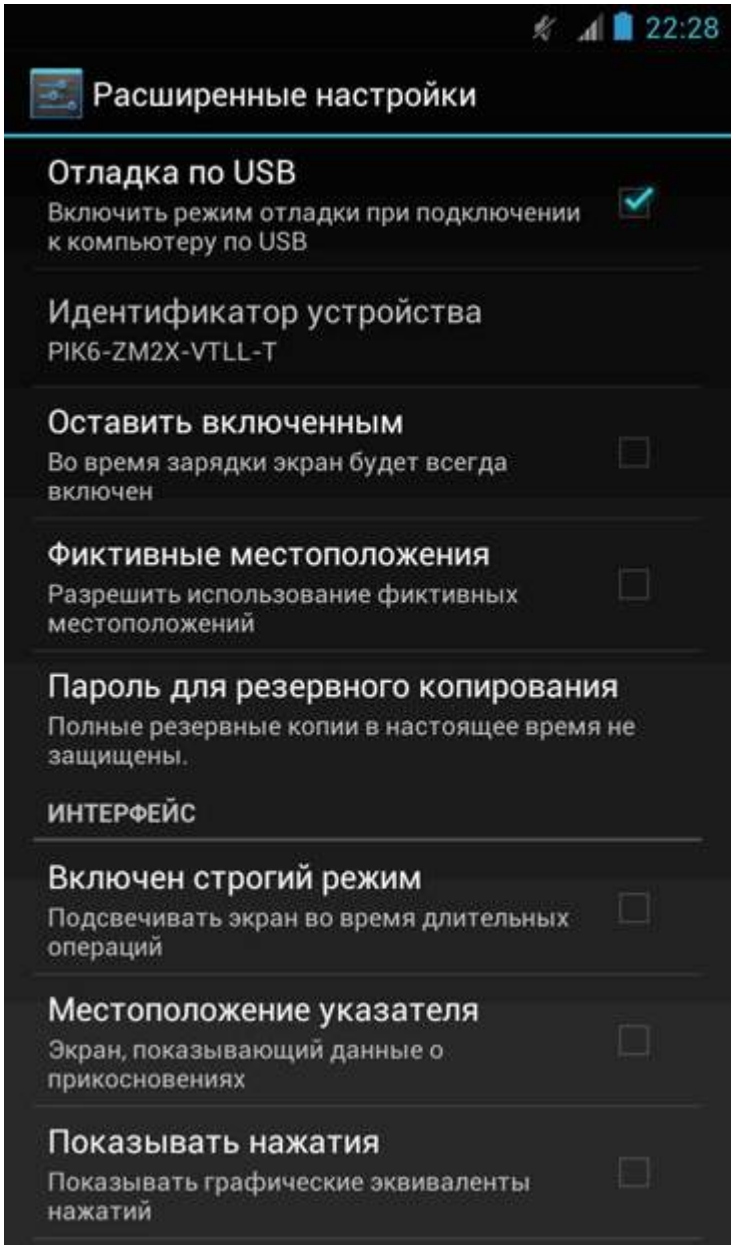
Прежде чем запустить проект на реальном устройстве, необходимо:

- Настроить устройство
- Настроить компьютер
- Настроить среду

2.5.1 Настройка устройства

Настройка устройства состоит в следующем:

1. Включить режим отладки по USB.
2. Для запуска файлов с расширением *.apk, полученных не из магазина приложений GooglePlay, необходимо разрешить установку приложений из альтернативных источников.



увеличить изображение
Рис. 2.26. Настройка устройства

2.5.2 Настройка компьютера

Чтобы настроить компьютер необходимо выполнить следующие действия:

1. Зайти в Диспетчер устройств (**Пуск->Панель управления->Система и безопасность->Система->Диспетчер устройств**) и найти ваше устройство

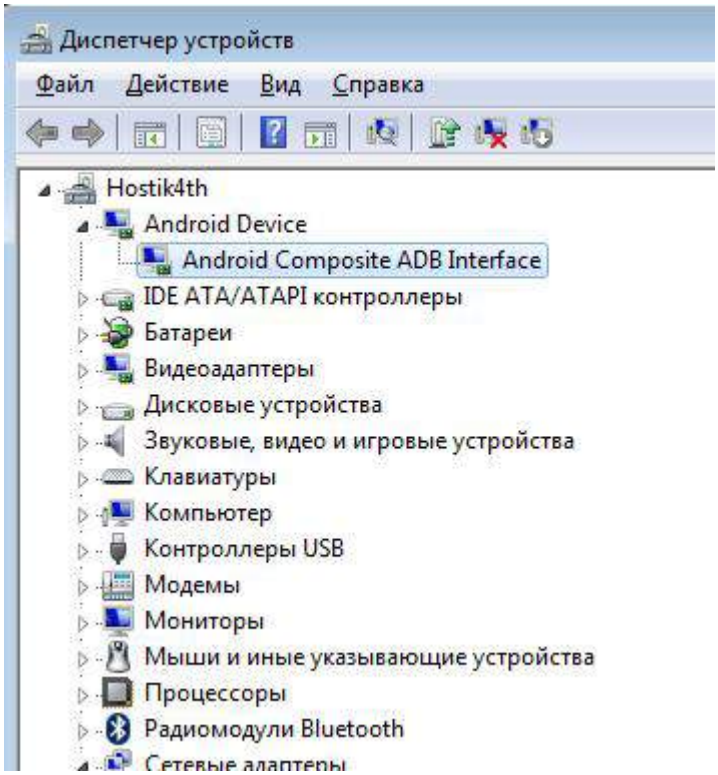


Рис. 2.27. Диспетчер устройств

2. Щелкнуть по нему правой кнопкой мыши и вызвать меню **Свойства**.

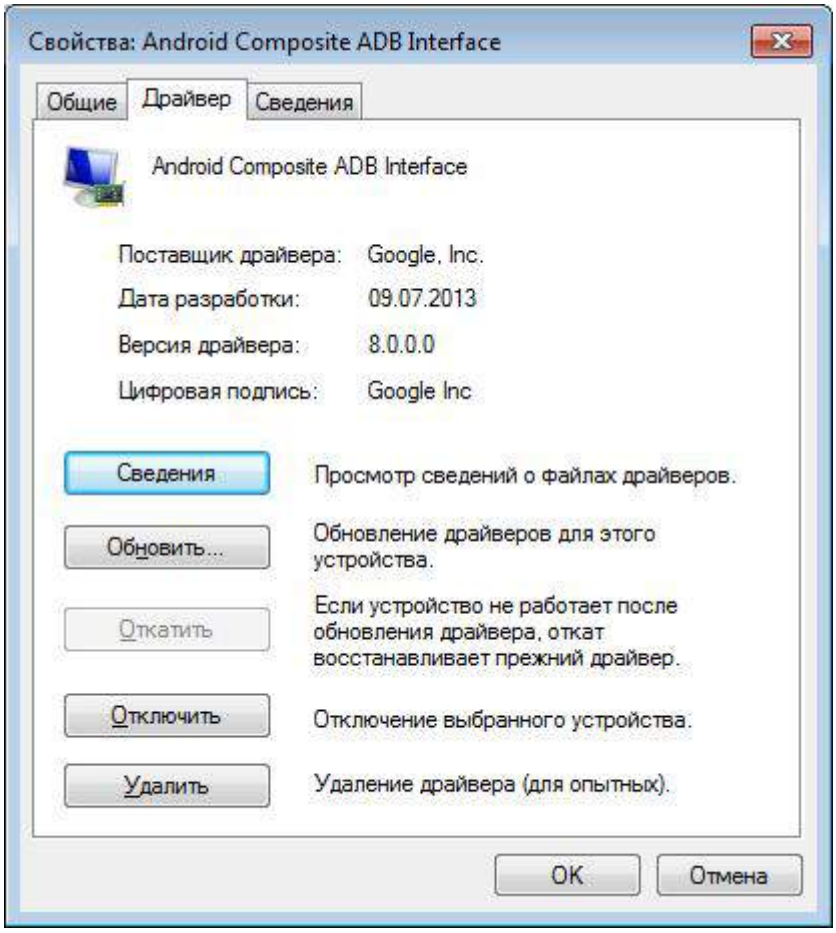
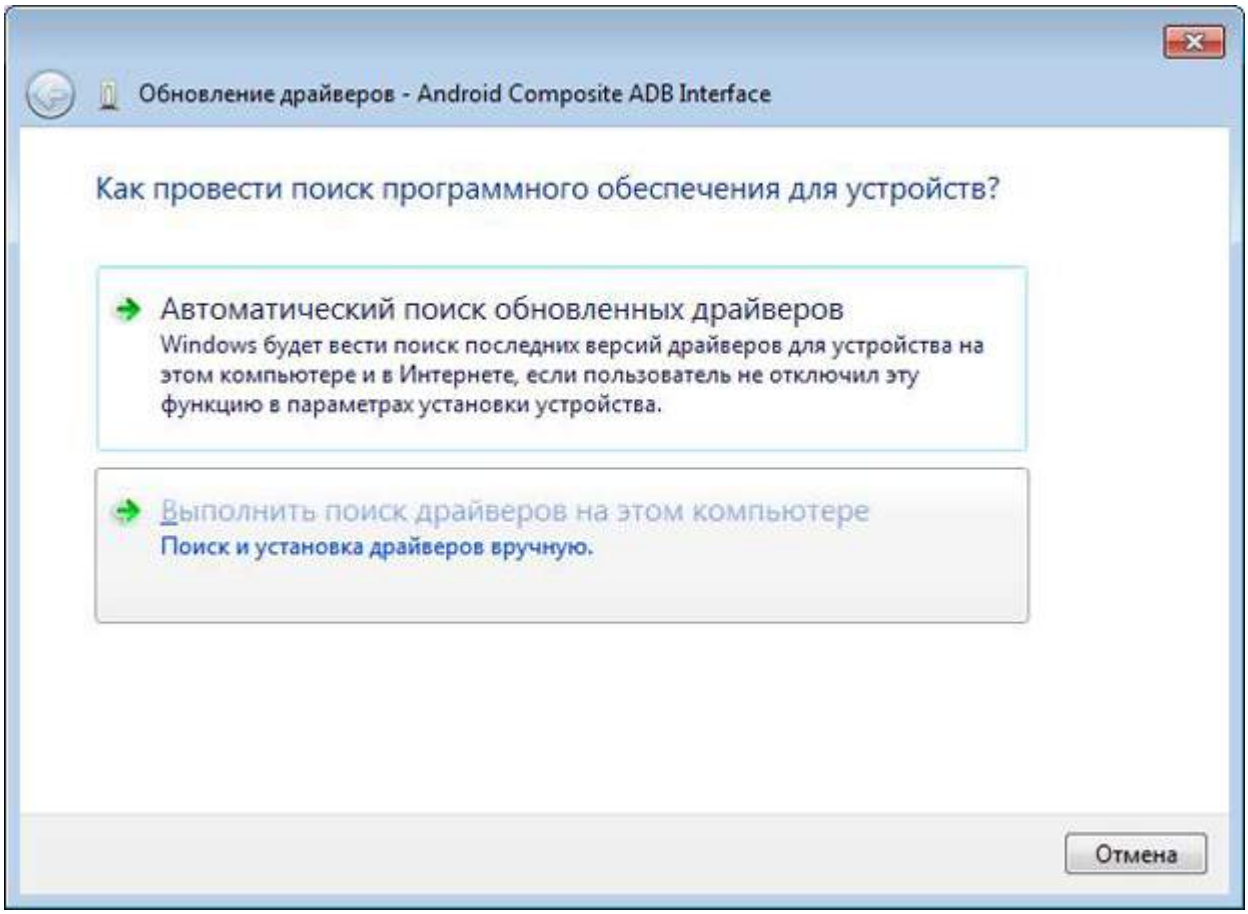


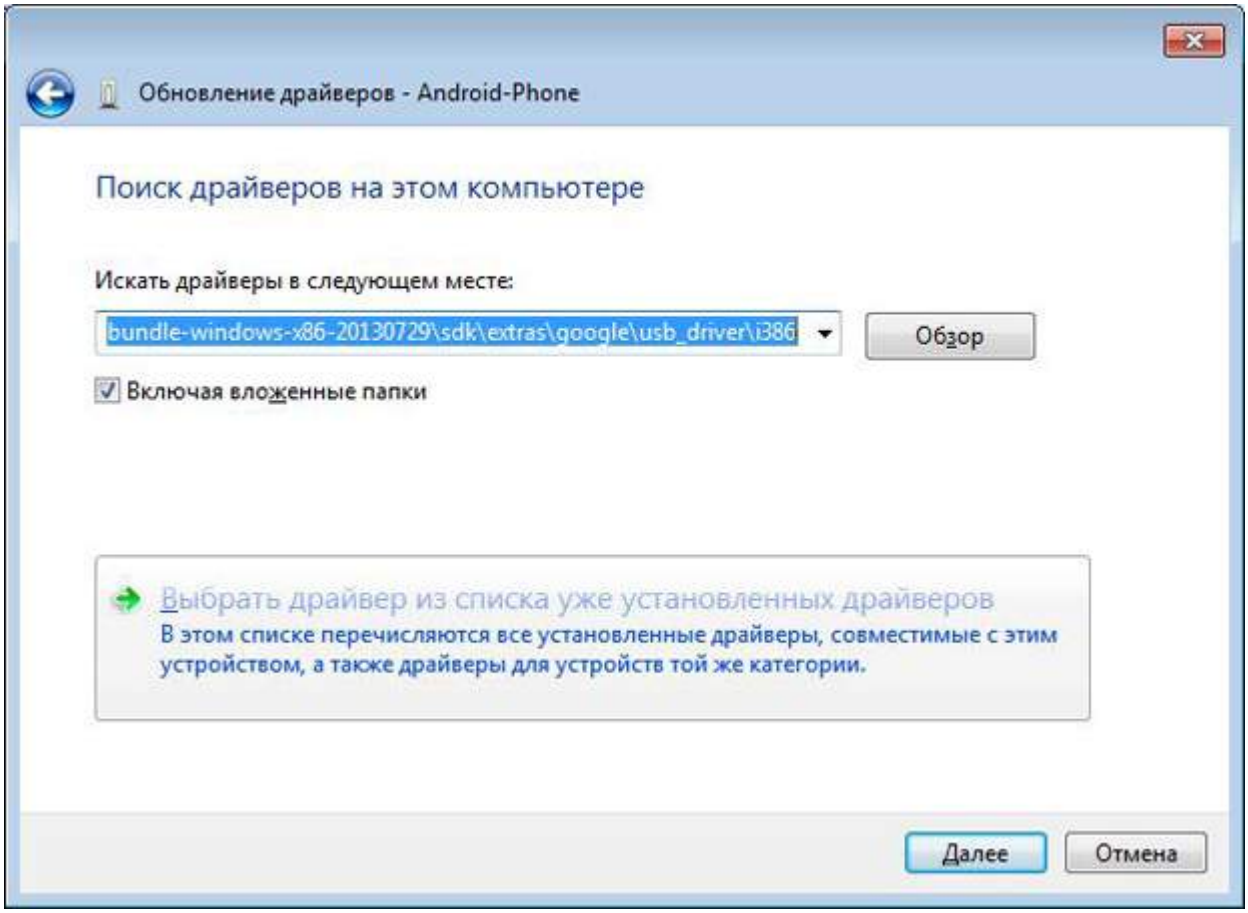
Рис. 2.28. Меню Свойства

3. Во вкладке **Драйвер** нажать на кнопку **Обновить** драйвер.



увеличить изображение
Рис. 2.29. Обновление драйвера

4. В появившемся окне выбрать **Выполнить поиск драйвера на этом компьютере**.



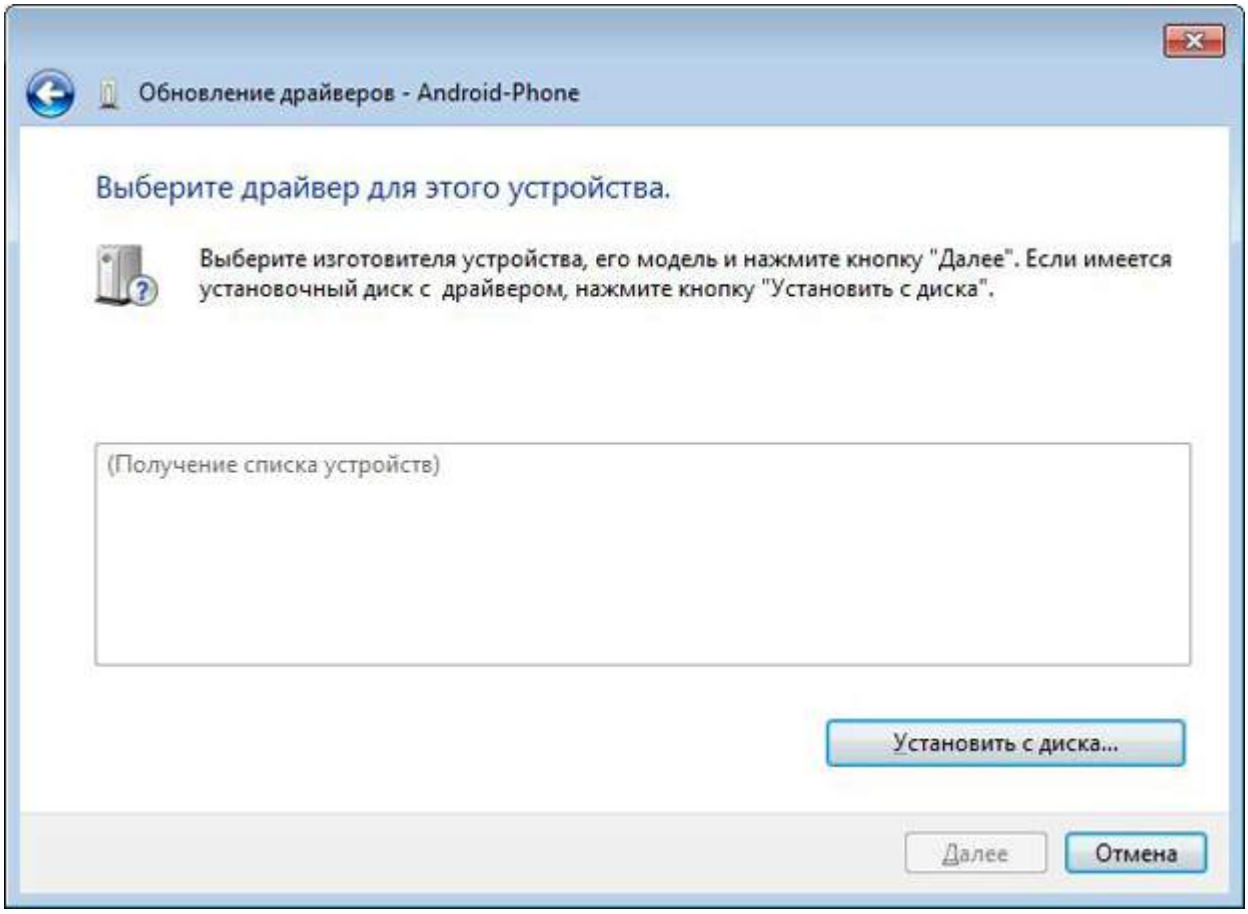
увеличить изображение
Рис. 2.30. Поиск драйверов на компьютере

5. В следующем окне нужно прописать путь, откуда будет установлен драйвер.

   Внашемслучаепутьследующий: **adt-bundle-windows-x86-20130729\sdk\extras\google\usb_driver\i386**.

   Или можно отыскать драйвер следующим образом:

   - В следующем окне нажмите **Установить с диска**



увеличить изображение

Рис. 2.31. Выбор драйвера устройства

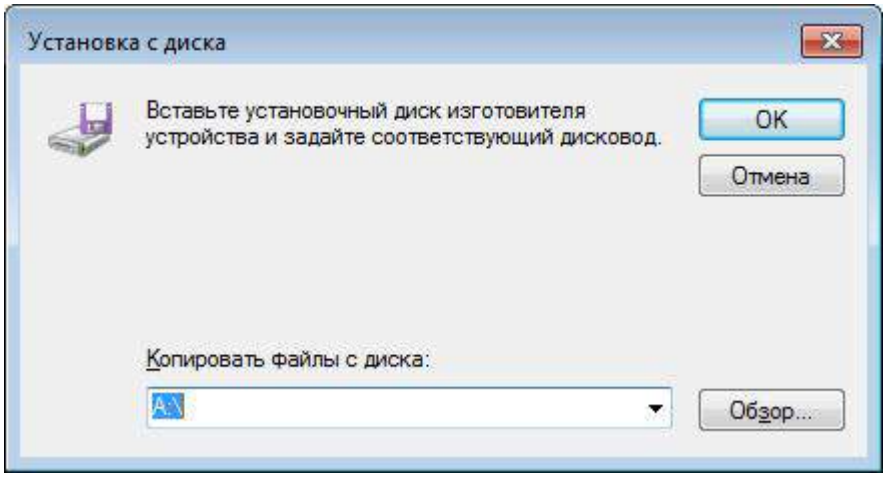


Рис. 2.32. Поиск драйвера

- Далее выберите файл android_winusb.inf

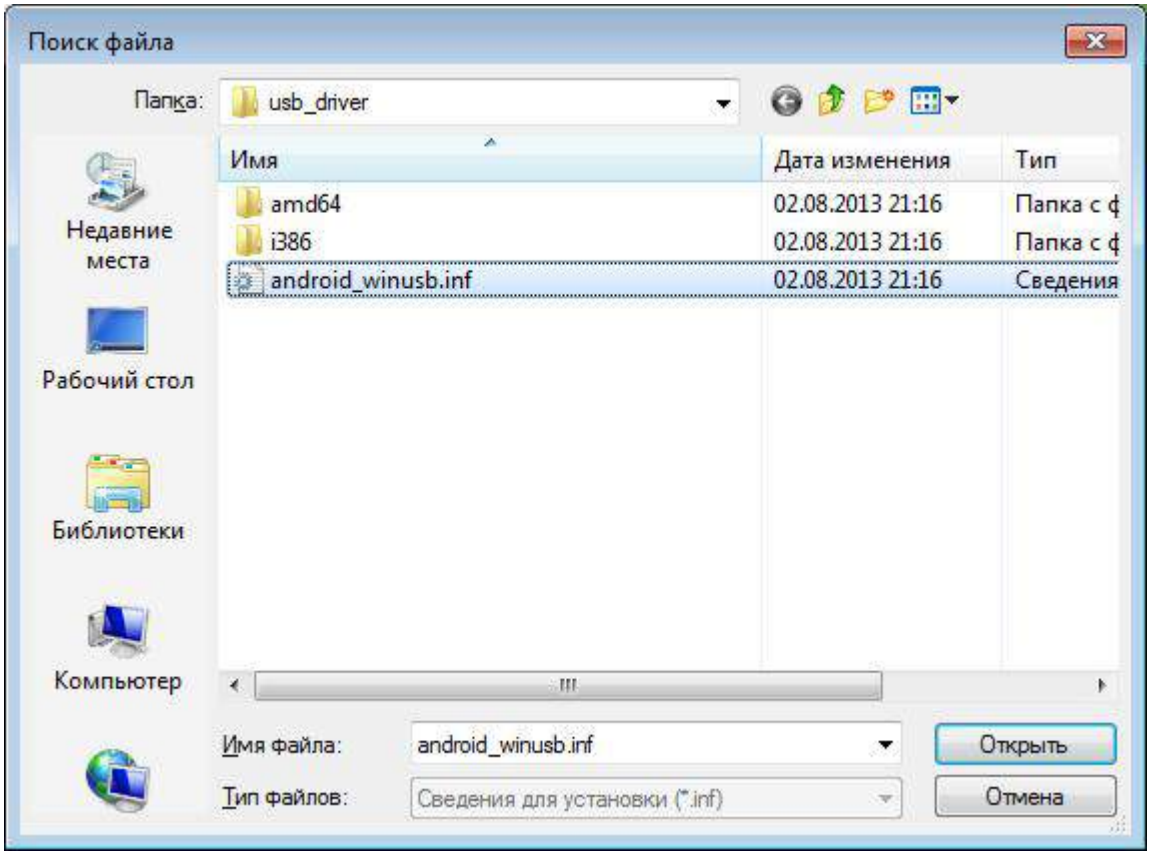
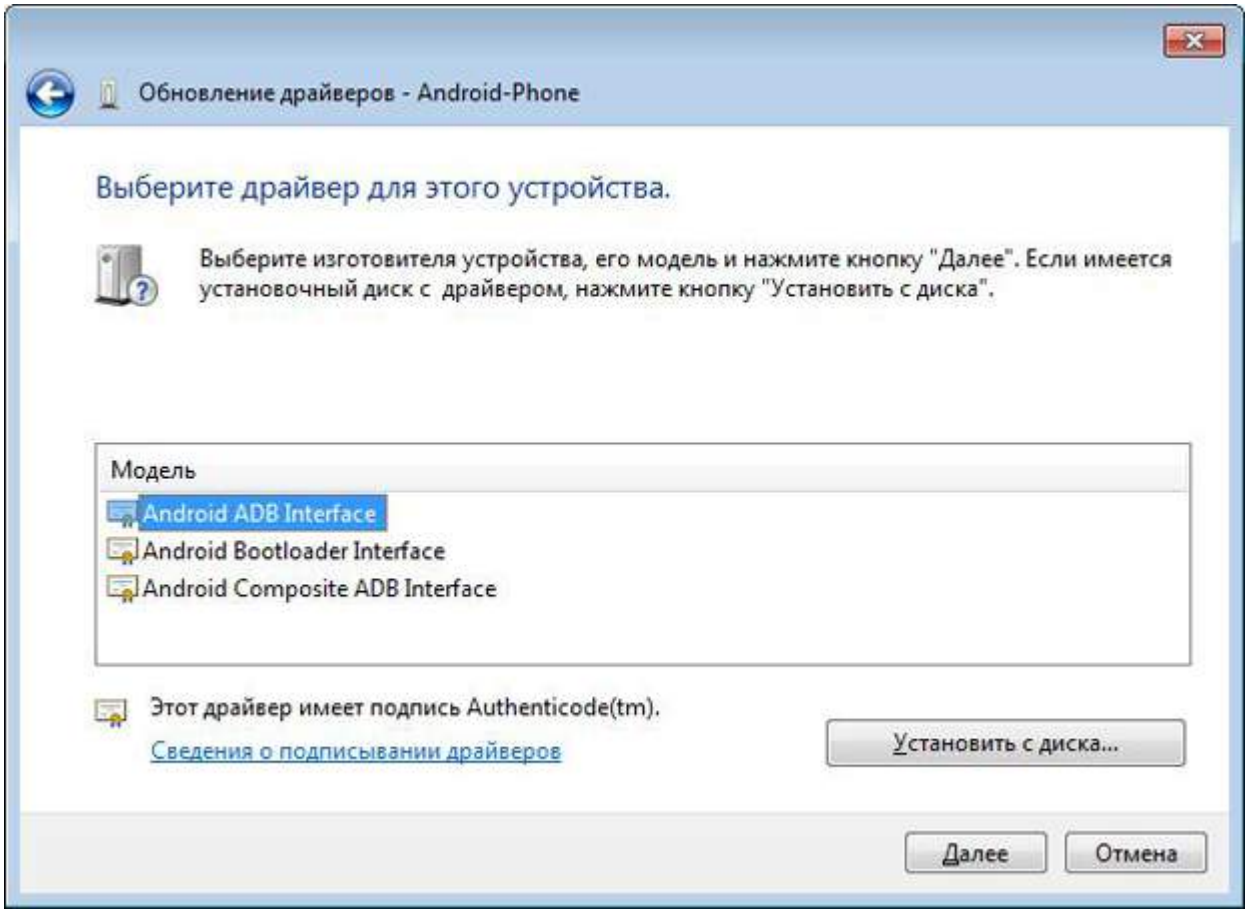


Рис. 2.33. Сведения для установки

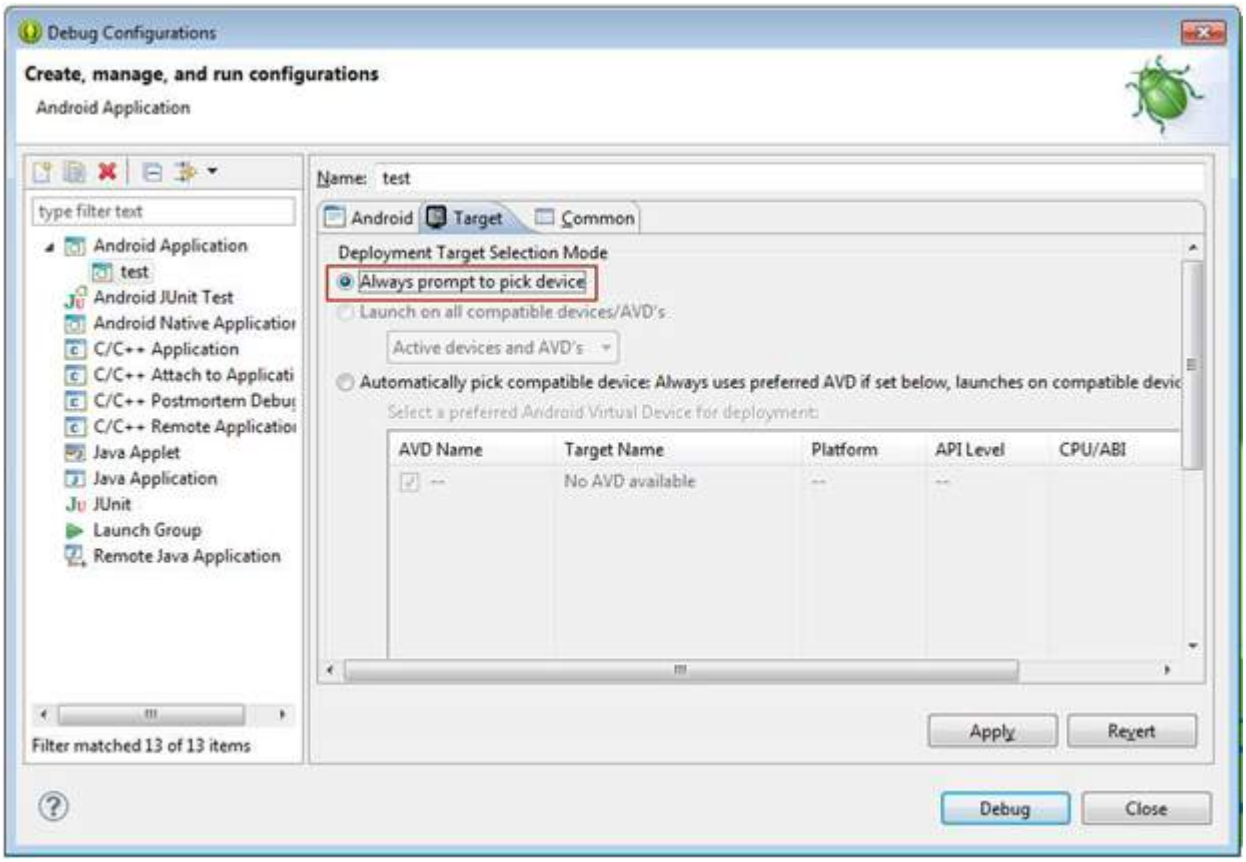
6. Нажмите **Далее** в этом и следующем окнах.
7. Из предложенного списка выберите **Android ADB Interface** и нажмите **Далее** и **Да**.



увеличить изображение
Рис. 2.34. Завершение установки драйвера

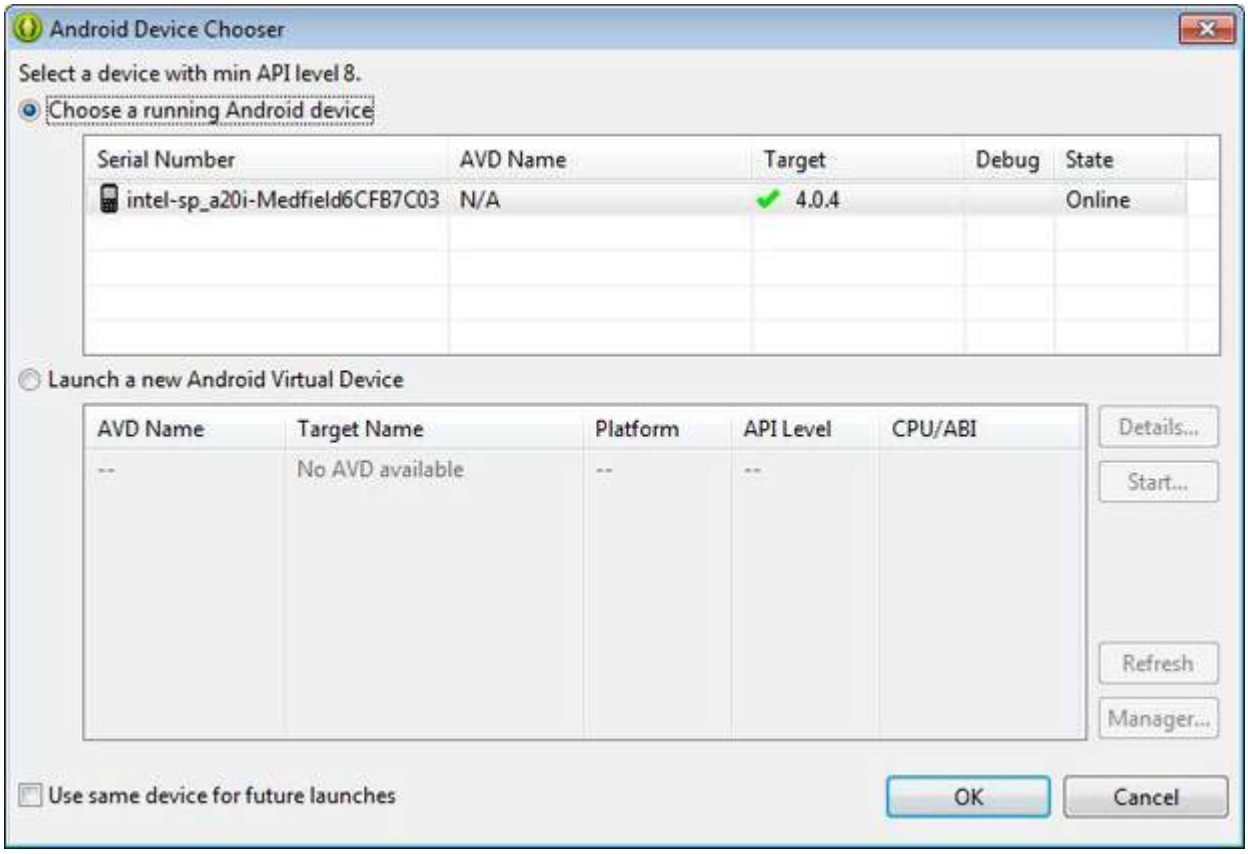
2.5.3 Настройка среды

1. В ADT зайдите в меню **Run\DebugConfigurations** и перейдите на вкладку **Target**. Поставьте флажок напротив **Alwayspromptpickdevice**.



увеличить изображение
Рис. 2.35. Настройка среды

2. В открывшемся окне выберите подключенное устройство, поставив флажок напротив **Choose a runningAndroidDevice**



увеличить изображение
Рис. 2.36. Выбор устройства

3. Теперь можно запустить приложение.

Задание

Создайте и запустите на вашем устройстве аналогичный проект, поменяв размер, цвет и рисунок иконки и изменив текст приветствия с *Hello, world!* на *Hello, Android!*

Внимание! Если Вы увидите ошибку на нашем сайте, выделите её и нажмите Ctrl+Enter.
© Национальный Открытый Университет "ИНТУИТ", 2016 | www.intuit.ru

Самостоятельная работа 2. Основные этапы разработки приложения с использованием Android IDE

Цель лабораторной работы:

Разработка простого приложения, помогающего понять структуру приложения, освоить основные операторы, привыкнуть к среде разработки.

Задачи лабораторной работы:

- создать новое приложение и изучить его структуру;
- настроить интерфейс приложения;
- реализовать логику приложения.

4.1 Введение

Для достижения поставленной цели в лабораторной работе создадим приложение в среде разработки Android IDE (Eclipse и ADT), подробно рассмотрим структуру полученного проекта и разберем назначение основных его элементов.

Чтобы дальнейшие действия приобрели некоторый смысл, сформулируем задачу, которую будет решать наше приложение, назовем его "Угадай число". Суть приложения в том, что программа случайным образом "загадывает" число от 0 до 100, а пользователь должен угадать это число. При каждом вводе числа, программа сообщает пользователю результат: введенное число больше загаданного, меньше или же "Ура, победа!" число угадано.

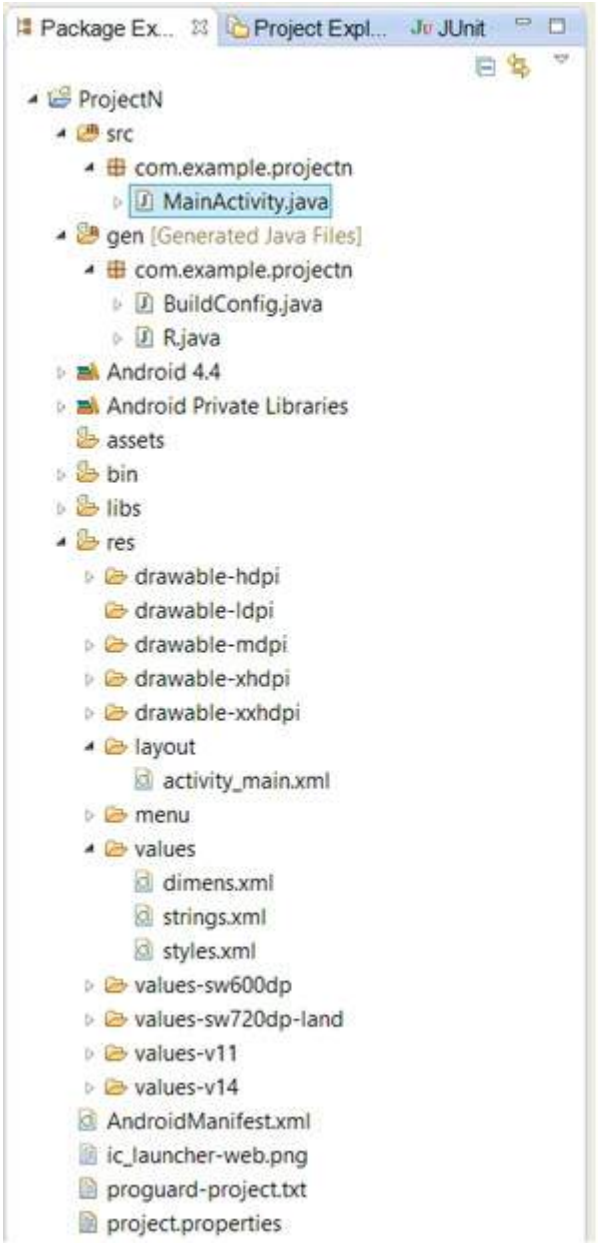
Разрабатываемое приложение выполняет свои функции только когда видимо на экране, когда оно не видимо его работа приостанавливается, т. е. имеем дело с приложением переднего плана. Для выполнения всей работы достаточно определить одну активность в приложении, фоновые процессы не предусмотрены.

Далее в работе рассмотрим простейшие элементы интерфейса пользователя и добавим их в приложение, а также рассмотрим вопросы, связанные непосредственно с программированием: научимся обрабатывать события, возникающие при взаимодействии приложения с пользователем; реализуем логику проверки числа на совпадение с загаданным.

4.2 Создание приложения и изучение его структуры

Создайте новый проект в среде Android IDE (Eclipse с ADT). Процесс создания нового проекта и описание основных настроек подробно рассмотрен в лабораторной работе к первой лекции.

В процессе создания проекта, мы назвали его ProjectN, среда разработки подготавливает необходимые папки и файлы. Полный иерархический список обязательных элементов проекта можно увидеть на вкладке Package Explorer (аналогичную информацию предоставляет вкладка Project Explorer), иерархия полученных папок и файлов для нашего проекта изображена на рис. 4.1.



увеличить изображение
Рис. 4.1. Структура проекта ProjectN

В настоящее время нас будет интересовать назначение нескольких файлов и папок.

Рассмотрим папки:

- папка **src** - содержит файлы с исходным кодом на языке Java. Именно в этой папке размещаются все классы, создаваемые в процессе разработки приложения. Сейчас в этой папке в пакете com.example.projectn размещается единственный класс MainActivity.java. Этот класс определяет главную и единственную активность в этом приложении.

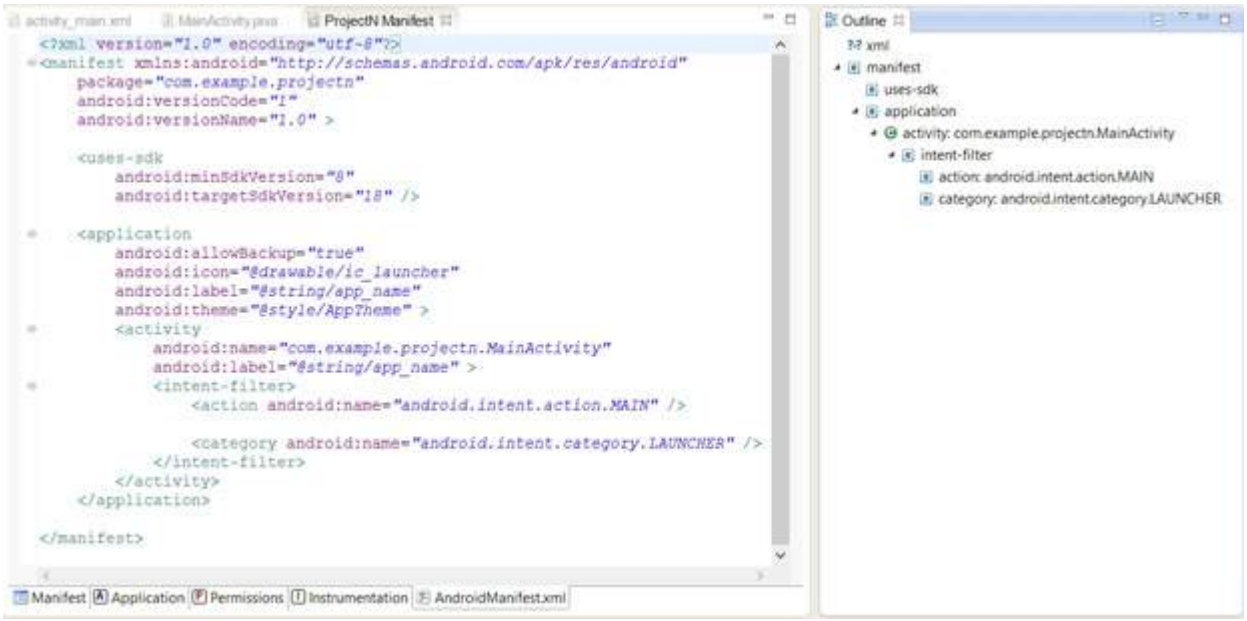
Комментарий 1: Имя пакету присваивается в процессе создания приложения в поле **Package Name**, использовать com.example не рекомендуется, т. к. пакет с таким именем нельзя загрузить в Google Play. Часто рекомендуют использовать в качестве имени пакета название сайта программиста, записанное в обратном порядке, можно просто использовать свои имя и фамилию. Последнее слово в имени пакета формируется автоматически и совпадает с именем проекта.

Комментарий 2: Имя файлу присваивается в процессе создания приложения на этапе настройки активности. Имя определяется в поле **Activity Name.**

Комментарий 3: Package Explorer отображает структуру папок, которая создается в каталоге, выбранном в качестве рабочего (Workspace) при запуске Eclipse. Например, рабочий каталог называется workspaceADT, в нем для нашего проекта появилась папка с именем ProjectN, в ней есть папка src, в ней com, в ней example, в ней projectn (заметьте, что название пакета распалось на три папки, каждое слово, отделенное точкой определило новую папку). И уже в папке projectn находится файл MainActivity.java и будут размещаться другие java-файлы проекта.

- папка **gen** - содержит java-файлы, которые не требуется изменять и лучше вообще не трогать. Эти файлы генерируются автоматически. Нас может заинтересовать файл R.java он содержит идентификаторы (ID) для всех ресурсов приложения.
- папка **res** - содержит структуру папок ресурсов приложения, рассмотрим некоторые из них:
    - **layout** - в данной папке содержатся xml-файлы, которые описывают внешний вид форм и их элементов, пока там находится только activity_main.xml;
    - **values** - содержит XML файлы, которые определяют простые значения, таких ресурсов как, строки, числа, цвета, темы, стили, которые можно использовать в данном проекте;
    - **menu** - содержит XML файлы, которые определяют все меню приложения.

Рассмотрим файл **AndroidManifest.xml** - файл в формате xml, который описывает основные свойства проекта, разрешение на использование ресурсов устройства и др. Сразу после создания приложения файл AndroidManifest.xml выглядит так, как показано на рис. 4.2.



увеличить изображение
Рис. 4.2. Файл AndroidManifest.xml только созданного проекта

Рассмотрим подробно файл манифеста.

Первый обязательный элемент <manifest> является корневым элементом файла, должен содержать обязательный элемент <application> и все остальные элементы по необходимости. Рассмотрим основные атрибуты этого элемента:

xmlns:android	- определяет пространство имен Android, всегда должен иметь значение: "http://schemas.android.com/apk/res/android". Обязательный атрибут.
package	- полное имя пакета, в котором располагается приложение. Обязательный атрибут. Имя должно быть уникальным, может содержать заглавные и строчные латинские буквы, числа и символ подчеркивания. Однако начинаться должно только с буквы. Для избежания конфликтов с другими разработчиками рекомендуется использовать имя вашего сайта

	(если он есть) записанное в обратном порядке. В нашем случае пакет имеет имя "com.example.projectn" и наше приложение не удастся разместить в Google Play (но мы на это и не претендуем). **Внимание:** если Вы опубликовали свое приложение, Вы **не можете менять имя пакета**, т.к. имя пакета служит уникальным идентификатором для приложения и в случае его смены приложение будет рассматриваться, как совсем другое и пользователи предыдущей версии не смогут его обновить.
android:versionCode	- внутренний номер версии приложения не виден пользователю. Этот номер используется только для определения является ли одна версия более современной по сравнению с другой, больший номер показывает более позднюю версию.
android:versionNumber	- номер версии, является строкой и используется только для того, чтобы показать пользователю номер версии приложения.
android:shareUserID, android:sharedUserLabel, android:installLocation	- эти атрибуты в нашем файле манифеста не представлены, про их назначение можно почитать по ссылке: http://developer.android.com/guide/topics/manifest/manifest-element.html.

Рассмотрим элемент <uses-sdk>, который показывает совместимость приложения с версиями Android. Основные атрибуты:

android:minSdkVersion	- указывает значение минимального уровня API, необходимого для работы приложения. Система Android не позволит установить приложение, если уровень API ниже, чем уровень, указанный в этом атрибуте. **Внимание:** если этот атрибут не указан, система установит значение по умолчанию равным "1", которое означает, что приложение совместимо со всеми версиями Android. И в случае, если приложение не совместимо со всеми версиями, установка пройдет на любую версию Android, а во время работы приложение сломается, когда попытается обратиться к недоступным элементам API. Поэтому необходимо всегда указывать значение этого атрибута.
android:targetSdkVersion	- указывает уровень API целевой платформы Android приложения, если этот атрибут пропущен, по умолчанию принимается значение android:minSdkVersion.
android:maxSdkVersion	- указывает максимальное значение уровня API, под который разрабатывалось приложение. Если значение этого атрибута ниже, чем уровень API соответствующий версии Android, на которую устанавливается приложение, то система не позволит установить такое приложение.

Если внимательно посмотреть на файл манифеста рис. 4.2, можно заметить, что данный атрибут в нем отсутствует. На самом деле разработчикам не рекомендуются задавать значение этого атрибута. Во-первых, это значение будет препятствовать использованию приложений на новых версиях Android при их появлении, несмотря на то, что все новые версии полностью обратно-совместимы. Во-вторых, стоит иметь ввиду, что задание этого атрибута может привести к тому, что приложение будет удалено с устройства после обновления Android до более высокого уровня API.

Подробнее с элементом <uses-sdk> и его атрибутами можно ознакомиться по ссылке: http://developer.android.com/guide/topics/manifest/uses-sdk-element.html.

Рассмотрим элемент <application>, который является обязательным элементом манифеста, полностью определяет состав приложения. Представляет собой контейнер для элементов <activity>, <service>, <receiver>, <provider> (и не только), каждый из которых определяет соответствующий компонент приложения. Содержит набор атрибутов, действие которых распространяется на все

компоненты приложения. Рассмотрим атрибуты элемента <application>, представленные в манифесте на рис. 4.2:

android:allowBackup	- определяет разрешение для приложения участвовать в резервном копировании и восстановлении. Если значение этого атрибута false, то для приложения никогда не может быть создана резервная копия, даже если проводится резервное копирование всей системы целиком. По умолчанию значение этого атрибута равно true.
android:icon	- определяет иконку для приложения целиком, а также иконку по умолчанию для компонентов приложения, которая может быть переопределена атрибутом android:icon каждого компонента. Задается как ссылка на графический ресурс, содержащий изображение, в нашем случае значение этого атрибута равно "@drawable/ic_launcher".
android:label	- определяет видимый для пользователя заголовок приложения целиком, а также заголовок по умолчанию для компонентов приложения, который может быть переопределен атрибутом android:label каждого компонента. Задается как ссылка на строковый ресурс, в нашем случае значение атрибута равно "@string/app_name".
android:theme	- определяет тему по умолчанию для всех активностей приложения, может быть переопределен атрибутом android:theme каждой активности. Задается как ссылка на стилевой ресурс, в нашем случае значение атрибута равно "@style/AppTheme".

На самом деле у элемента <application> гораздо больше атрибутов, чем нам удалось рассмотреть, найти полный список атрибутов с описаниями можно по ссылке: http://developer.android.com/guide/topics/manifest/application-element.html.

В приложении всего одна активность, других компонентов нет, в связи с этим элемент **<application>** в манифесте содержит ровно один элемент <activity> и больше никаких других элементов не содержит. Рассмотрим элемент <activity>, который определяет активность. Для каждой активности обязательно необходим свой элемент <activity> в манифесте. Рассмотрим атрибуты элемента <activity>, представленные в манифесте на рис. 4.3:

android:name	- определяет имя класса, который задает активность. Значение атрибута должно полностью определять имя класса с указанием пакета, в котором располагается класс. В нашем случае атрибут имеет значение: "com.example.projectn.MainActivity". Можно использовать сокращенную запись ".MainActivity", в этом случае добавляется имя пакета, определенное соответствующим атрибутом элемента <manifest>.
android:configChanges	- перечисляет изменения конфигурации, которыми может управлять активность. Если конфигурация меняется во время работы, то по умолчанию активность останавливается и перезапускается. Если же изменение конфигурации указано в этом атрибуте, то при появлении этого изменения активность не перезапускается, вместо этого она продолжает работать и вызывает метод onConfigurationChanged(). В нашем случае атрибут имеет значение "orientation|keyboardHidden|screenSize", т. е. при смене ориентации экрана, смене размера экрана и изменении доступности клавиатуры не произойдет перезапуск активности.
android:label	- определяет видимый пользователю заголовок активности, если он отличается от общего заголовка приложения. Задается как ссылка на строковый ресурс, в нашем случае значение атрибута равно "@string/app_name"(т.е. можно было и не задавать).
android:theme	- определяет тему активности, если она отличается от общей темы

	приложения, заданной соответствующим атрибутом элемента <application>. Задается как ссылка на стилевой ресурс, в нашем случае значение атрибута равно "@style/FullscreenTheme".

На самом деле у элемента <activity> гораздо больше атрибутов, чем нам удалось рассмотреть, найти полный список атрибутов с описаниями можно по ссылке: http://developer.android.com/guide/topics/manifest/application-element.html.

В манифесте для нашего приложения элемент <activity> содержит ровно один элемент: <intent-filter>, определяющий типы намерений, которые может принимать активность. Этот элемент содержит два элемента: <action> и <category>.

Первый элемент определяет действия, которые проходят в фильтр намерений, при этом <intent-filter> должен содержать хотя бы один элемент <action>, в противном случае ни один объект-намерение не сможет пройти через фильтр и активность не возможно будет запустить. Элемент <action> имеет единственный атрибут android:name="android.intent.action.MAIN".

Второй элемент определяет имя категории в фильтре намерений. Имеет единственный атрибут android:name="android.intent.category.LAUNCHER".

На этом разбор манифеста приложения закончим, подробно с описанием всех элементов этого файла можно познакомиться по ссылке: http://developer.android.com/guide/topics/manifest/manifest-intro.html.

Чаще всего, при создании приложения приходится иметь дело с папками **src**, **res/layout** и **res/values**, т.к. там находятся основные файлы проекта.

4.3 Настройка интерфейса приложения

До того, как начнем формировать интерфейс, имеет смысл подготовить возможность проверки разрабатываемого приложения на ошибки. Чтобы не загружать каждый раз приложение на реальное устройство, в Android SDK предусмотрена возможность использования виртуального устройства (AVD или Android virtual device), эмулирующего работу реального смартфона. Процесс создания виртуального устройства или эмулятора подробно расписан в "Установка и настройка среды программирования ADT Bundle" .

Пришло время задуматься о внешнем виде приложения. Для начала необходимо определить какие элементы графического интерфейса нам нужны, как эти элементы будут располагаться на форме и каким образом будет реализовано взаимодействие с пользователем.

Так как приложение очень простое, то и интерфейс особой сложностью отличаться не будет. Нам потребуется поле для ввода чисел (**TextEdit**), текстовая метка для вывода информации (**TextView**) и кнопка для подтверждения введенного числа (**Button**). Располагать элементы интерфейса будем друг под другом, сверху информационная часть, ниже поле ввода, кнопку разместим в самом низу приложения. Взаимодействие приложения с пользователем организуется очень просто: пользователь вводит число в поле для ввода и нажимает кнопку, читает результат в информационном поле и, либо радуется победе, либо вводит новое число.

Если пользователь вводит правильное число, то приложение предлагает ему сыграть снова при этом кнопка будет играть роль подтверждения, а в информационное поле будет выведено приглашение к повторной игре.

Схематично интерфейс приложения изображен на рис. 4.3.

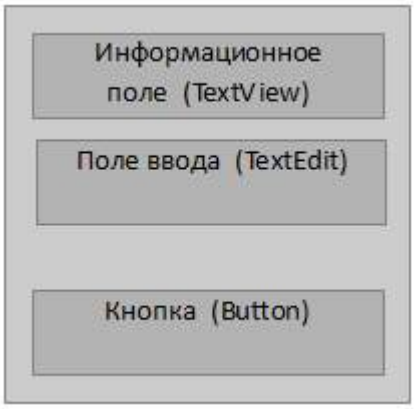
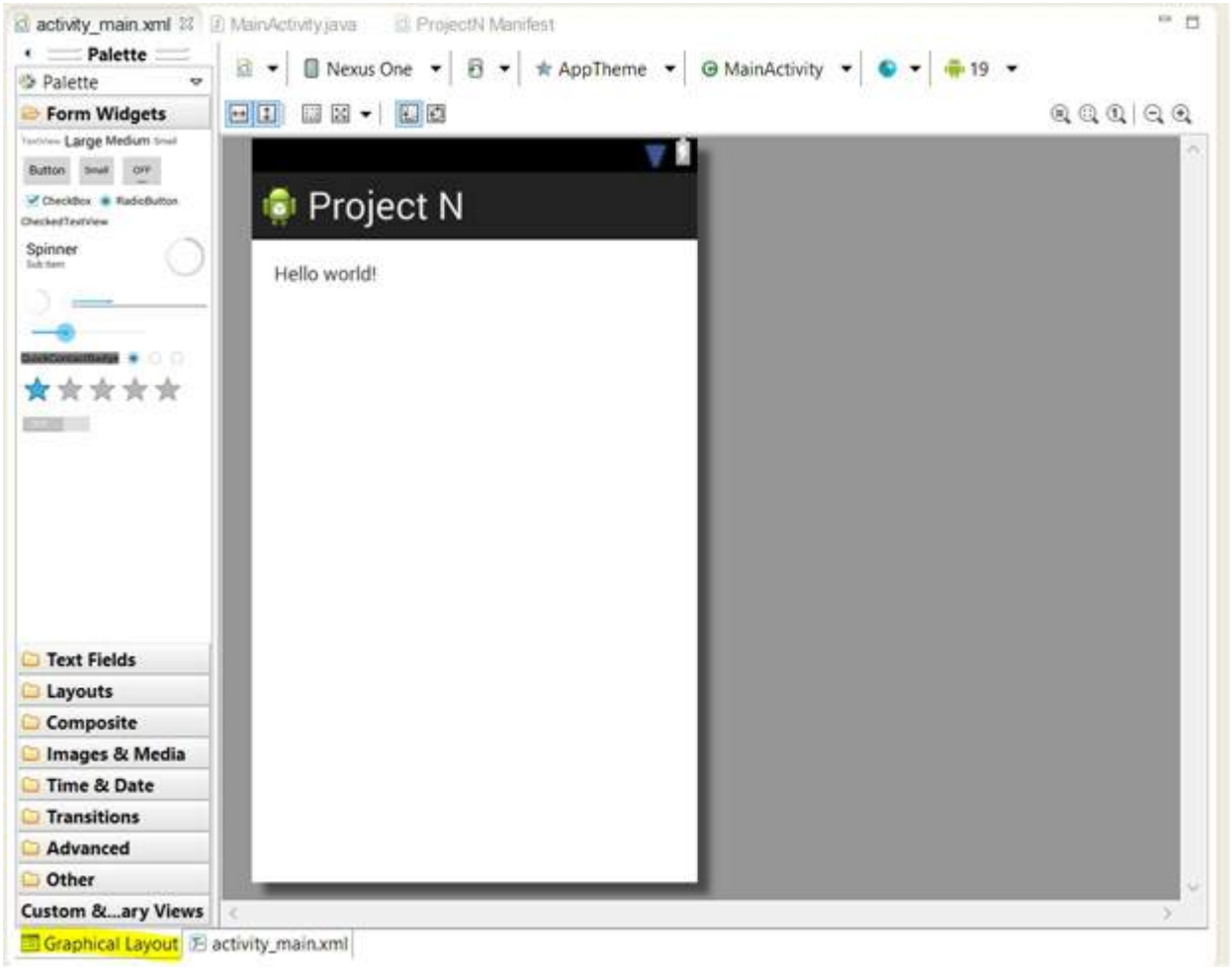


Рис. 4.3. Схема интерфейса приложения "Угадай число"

Нам необходимо добавить на форму три элемента: информационное поле (**TextView**), поле ввода (**TextEdit**) и кнопку (**Button**).

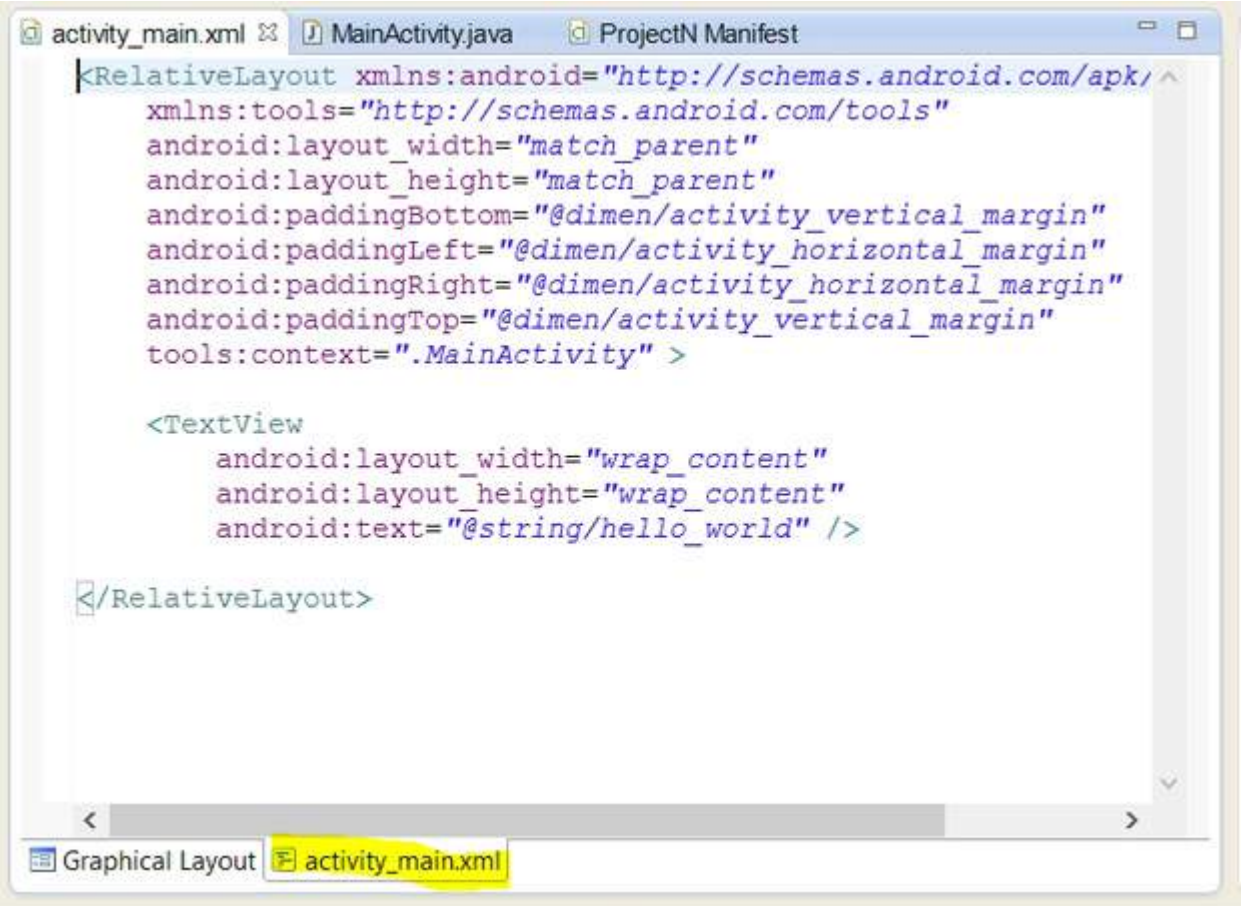
Android IDE поддерживает два способа для выполнения действий по формированию интерфейса приложения: первый основан на XML-разметке, второй относится к визуальному программированию и позволяет перетаскивать объекты интерфейса и размещать их на форме с помощью мыши. Считается, что визуальный способ подходит для новичков, а более продвинутые разработчики могут писать код вручную, однако чаще всего используется комбинированный подход.

Для формирования интерфейса будем работать с файлом **res/layout/activity_main.xml**. На рис. 4.4 можно увидеть редактор, соответствующий визуальному способу формирования интерфейса, этому режиму соответствует вкладка **Graphical Layout**.



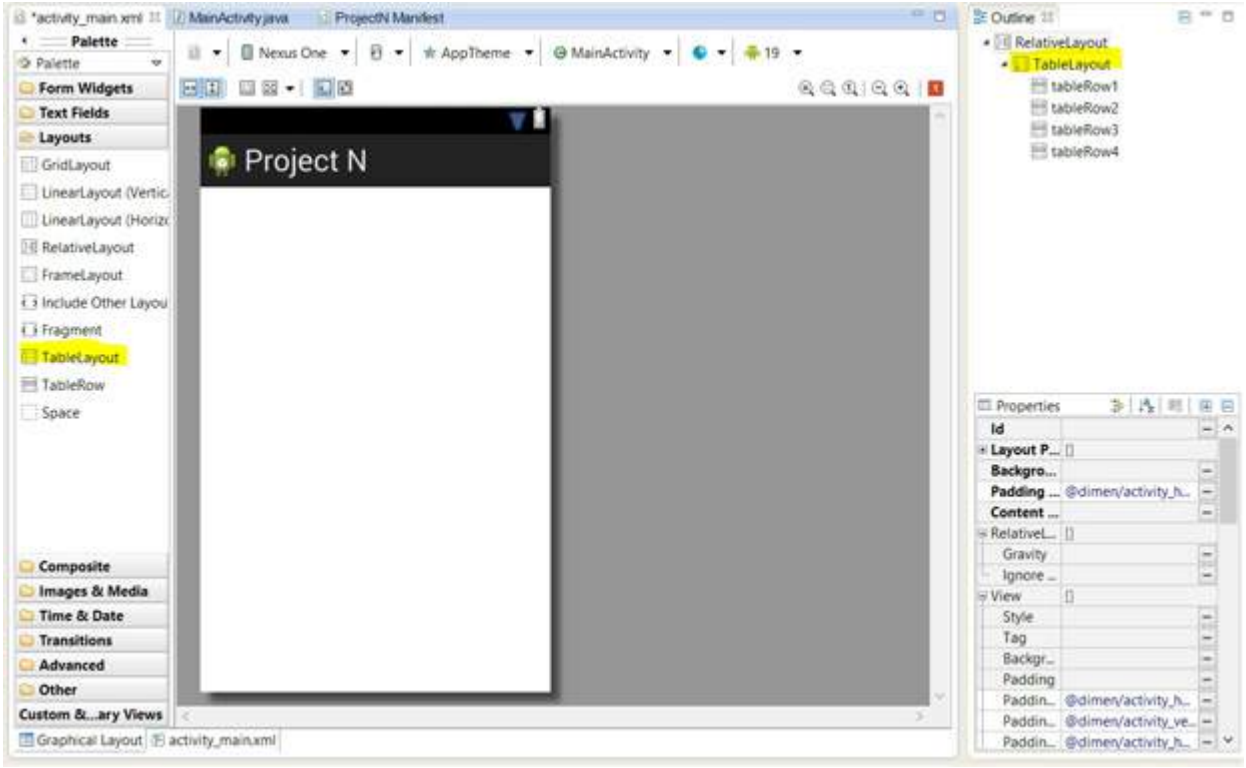
увеличить изображение
Рис. 4.4. Графическое изображение активности приложения

На рис. 4.4 рядом с вкладкой **Graphical Layout** расположена вкладка **activity_fullscreen.xml**. Она соответствует режиму редактирования интерфейса путем формирования XML файла. На рис. 4.5 можно увидеть редактор XML файла.



увеличить изображение
Рис. 4.5. Описание активности в XML формате

Зададим табличное расположение компонентов на форме, для этого выберем вкладку **Layouts,** найдем там **TableLayout** и добавим его на форму. На рис. 4.6 можно увидеть результат этих действий.



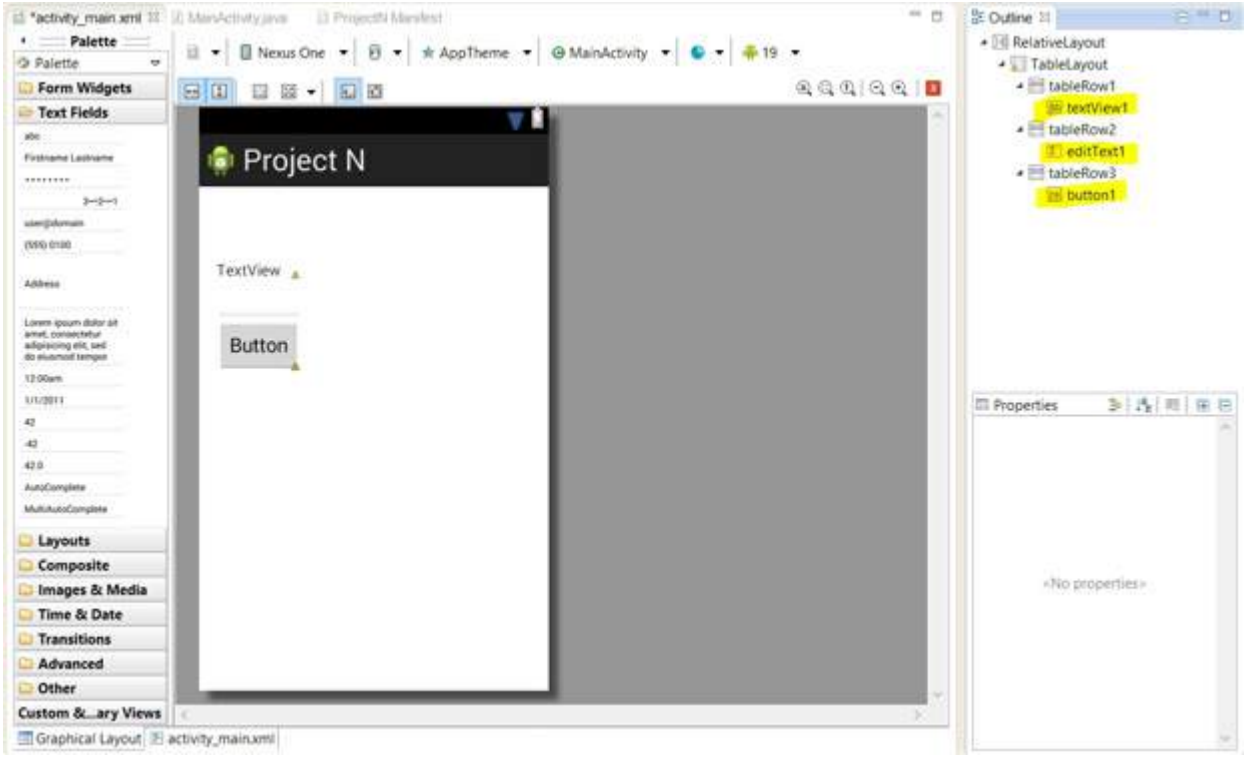
увеличить изображение
Рис. 4.6. Настройка интерфейса, добавление TableLayout

Теперь начнем добавлять элементы интерфейса, будем использовать графический режим правки.

Во-первых, нам необходимо добавить информационное поле. Для этого на панели **=Palette=** выбираем вкладку **Form Widgets**, на этой вкладке найдем поле **TextView**, перенесем в окно приложения, разместим в первой строке таблицы (**TableLayout**).

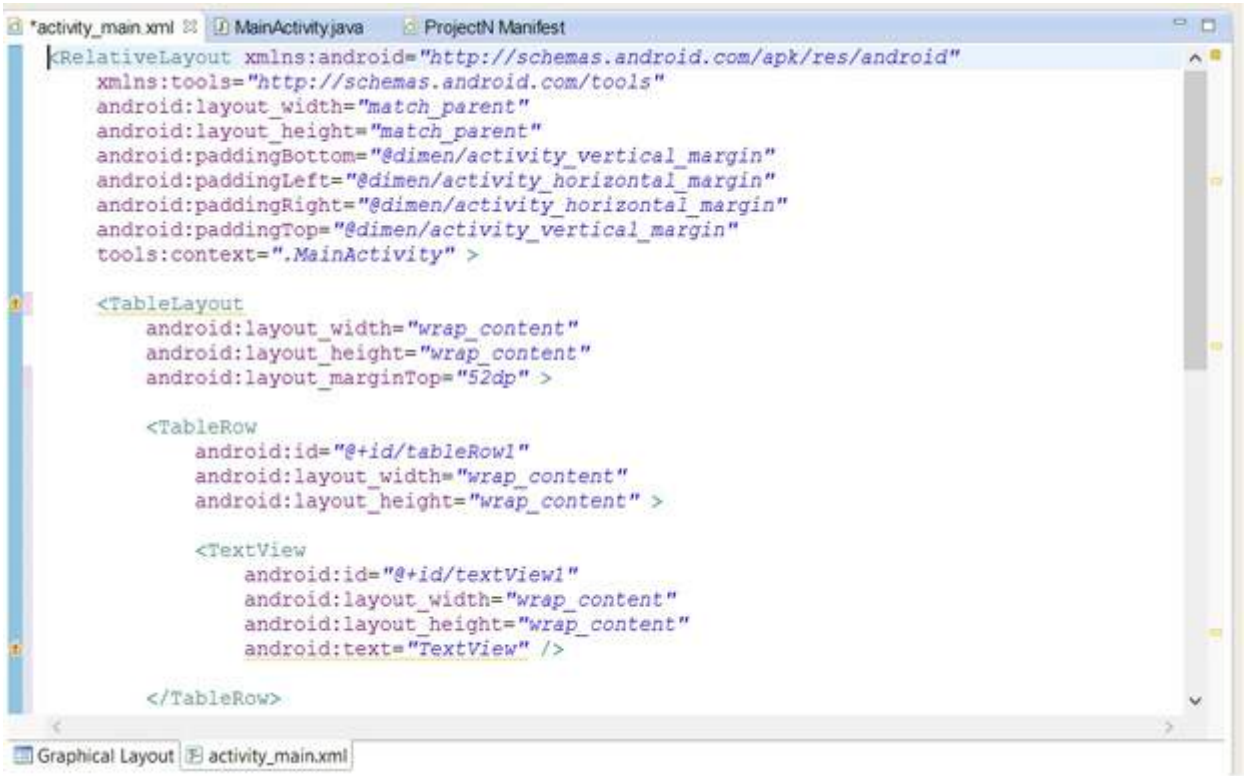
Во-вторых, нам потребуется поле ввода информации, на вкладке **Text Fields** найдем текстовое поле **Number** и разместим во второй строке таблицы.

В-третьих, вернемся на вкладку **Form Widgets**, выберем там элемент **Button** и добавим в третью строку таблицы. Не нужную четвертую строку таблицы удалим, получим следующий вид приложения, см. рис. 4.7.



[увеличить изображение]
Рис. 4.7. Интерфейс приложения

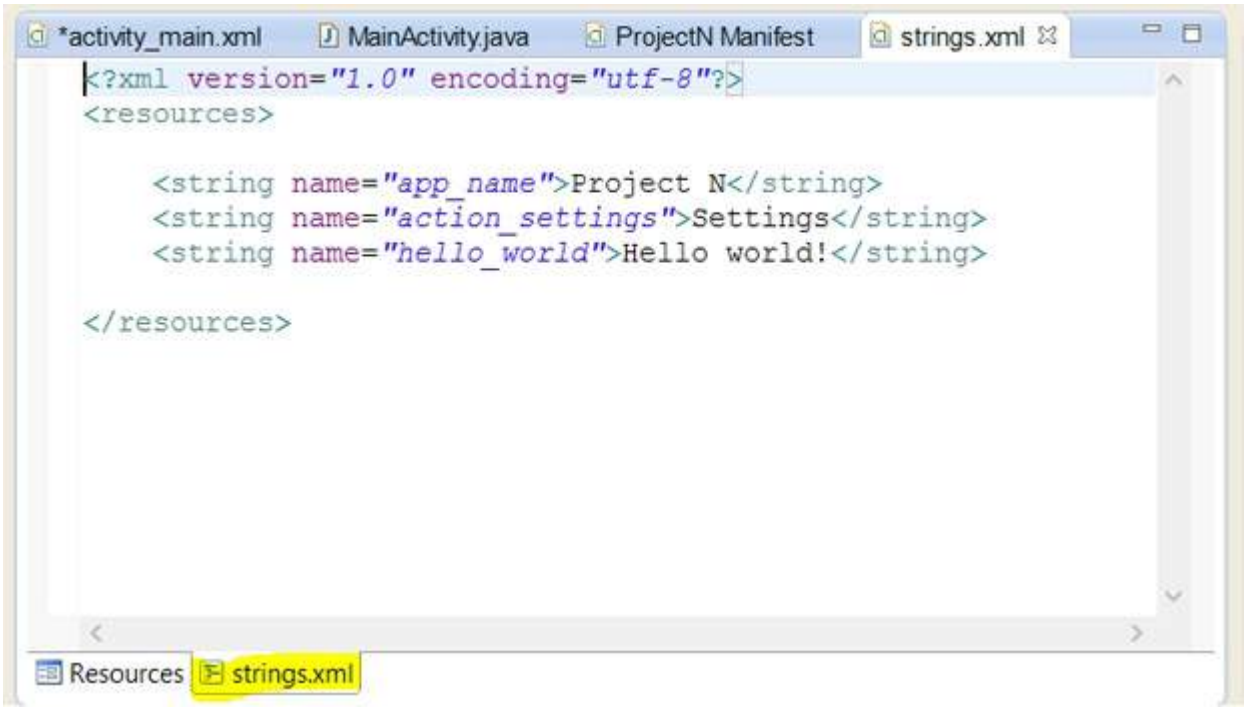
После настройки интерфейса можно заглянуть в файл **activity_main.xml**, в этом файле прописано, что используется **TableLayout** и дано описание каждой из трех строк. На [рис. 4.8] можно увидеть, как выглядит такое описание на примере первой строки таблицы.



[увеличить изображение]
Рис. 4.8. Фрагмент файла activity_main.xml, описание строки в TableLayout

Теперь необходимо наполнить наши элементы интерфейса смыслом, нам понадобится текст для общения с пользователем, при программировании под Android существует практика разделять

ресурсы и код приложения. Для хранения любых строк, которые могут понадобиться приложению, используется файл **strings.xml**. Хранение всех строковых ресурсов в этом файле серьезно облегчает локализацию приложения на другие языки. Этот файл можно найти в Package Explorer в папке **res/values**. Откроем его и посмотрим, что там есть, см. рис. 4.9.



увеличить изображение
Рис. 4.9. Файл string.xml

Уберем лишние строки и добавим новые, результат можно посмотреть на рис. 4.10.

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

    <string name="app_name">Угадай число!</string>
    <string name="behind">Недолёт!</string>
    <string name="ahead">Перелёт!</string>
    <string name="hit">В точку!</string>
    <string name="input_value">Ввести значение</string>
    <string name="play_more">Сыграть ещё</string>
    <string name="try_to_guess">Попробуйте угадать число (1 &#8211; 100) </string>
    <string name="error">Неверный ввод!</string>

</resources>
```

Рис. 4.10. Отредактированный файл string.xml

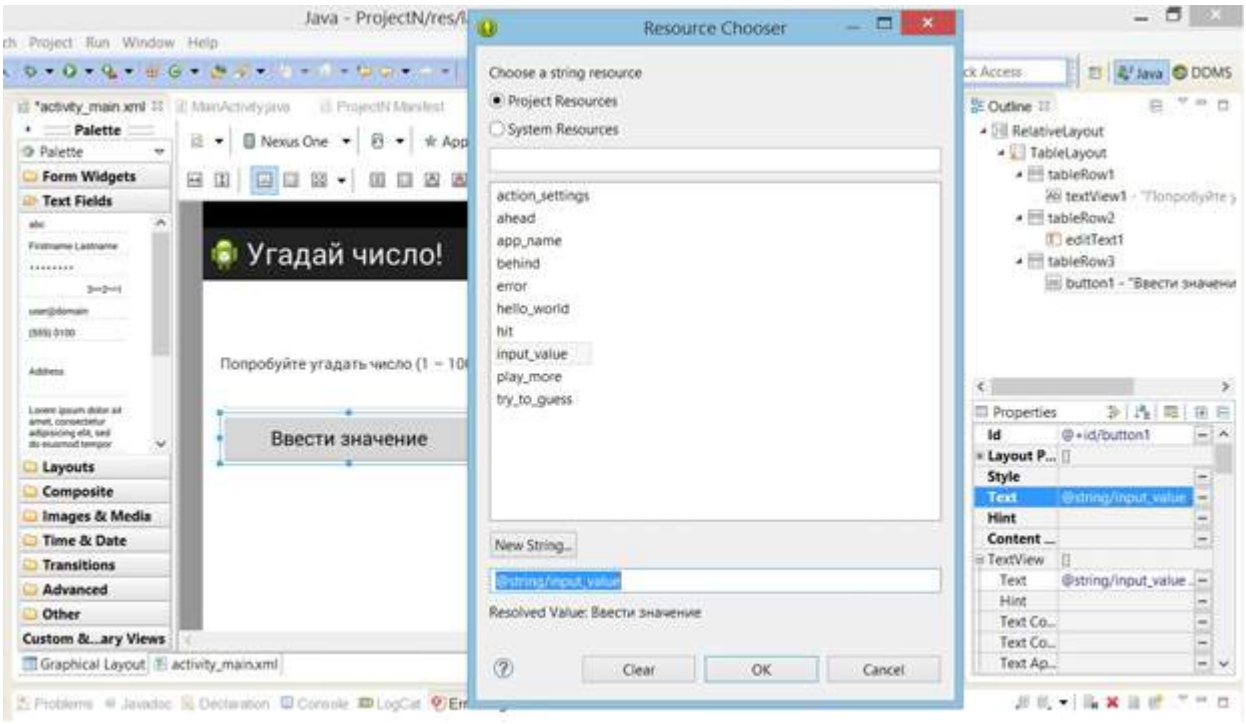
Данные переменные будут выполнять следующие задачи:

- app_name установит "видимое" название приложения;
- behind, ahead, hit оповестят пользователя об его успехах в игре;
- play_more и try_to_guess установит название кнопки, которое объяснит её функции;
- input_value пригласит пользователя к вводу числа;
- error сообщит о неверном вводе.

После изменения **strings.xml**, при переходе на другую вкладку, не забудьте сохранить изменения (самый быстрый способ - нажать **Ctrl+S**).

Настроим текст в информационном поле. Для этого на вкладке **Properties** в правой части окна выберем элемент **textView1** (это и есть наше информационное поле, имеет смысл придумать ему более осмысленное имя). Найдем свойство **Text**, подставим в него значение строки с именем try_to_guess, см. рис. 4.11.

Аналогично можно настроить текст, которым нас будет приветствовать кнопка, только в этом случае надо работать с элементом **button1**.



увеличить изображение
Рис. 4.11. Настройка текста для кнопки button1

Пришло время вспомнить о виртуальном устройстве, если оно работает, уже можно запустить проект и посмотреть, как приложение будет выглядеть на экране устройства, а выглядеть оно может как показано на рис. 4.12.

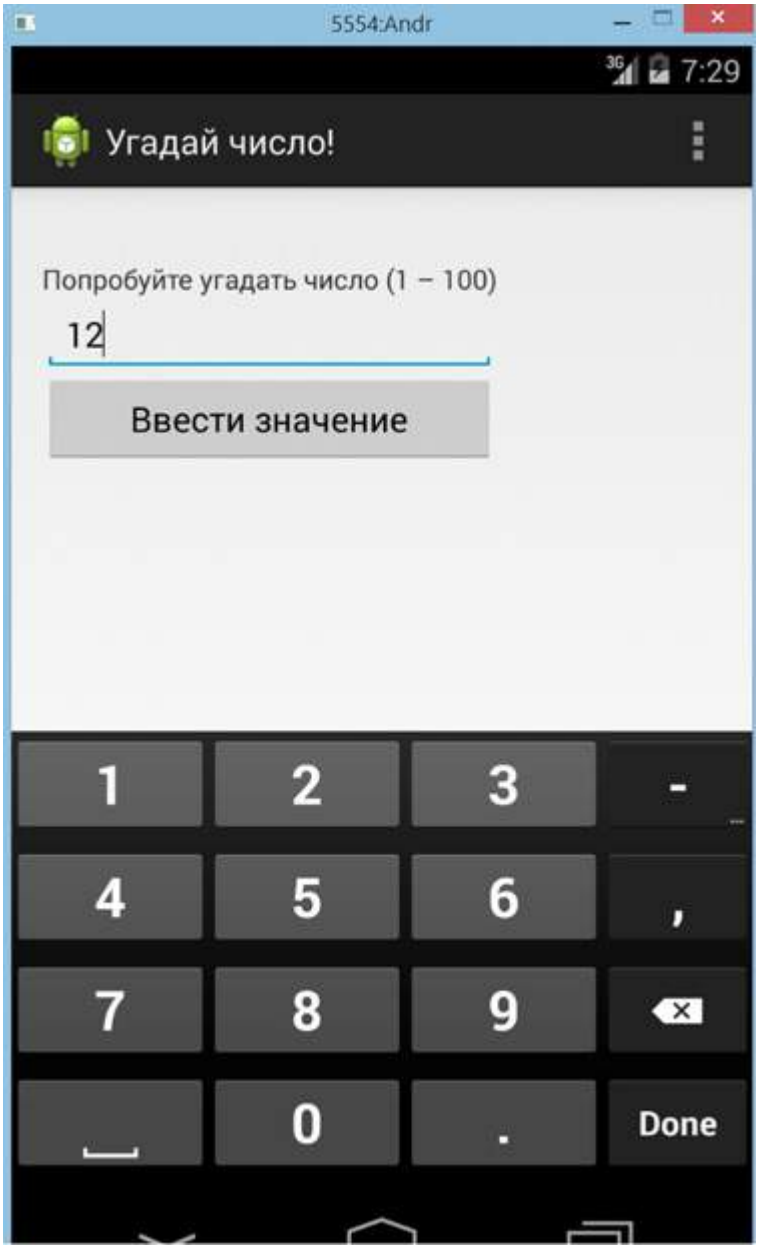


Рис. 4.12. Запуск приложения на виртуальном устройстве

Приложение выглядит довольно просто, но мы на многое и не рассчитывали. Главное, что нас интересует, это наличие всех элементов на экране, верный текст в каждом элементе, где он предусмотрен и возможность вводить числа в поле ввода. На рис. 4.12 видно, что все требования выполнены. Приложение есть, его можно запустить на виртуальном или реальном устройстве, но оно ничего не делает. Следующим шагом будет реализация логики приложения, т. е. обработка события нажатия на кнопку, как было прописано в задании.

4.4 Реализация логики приложения

Приступим непосредственно к программированию, работать будем с файлом **src/com.example.projectn/MainActivity.java**. Найдем этот файл в Package Explorer см. рис. 4.13, откроем и начнем редактировать.

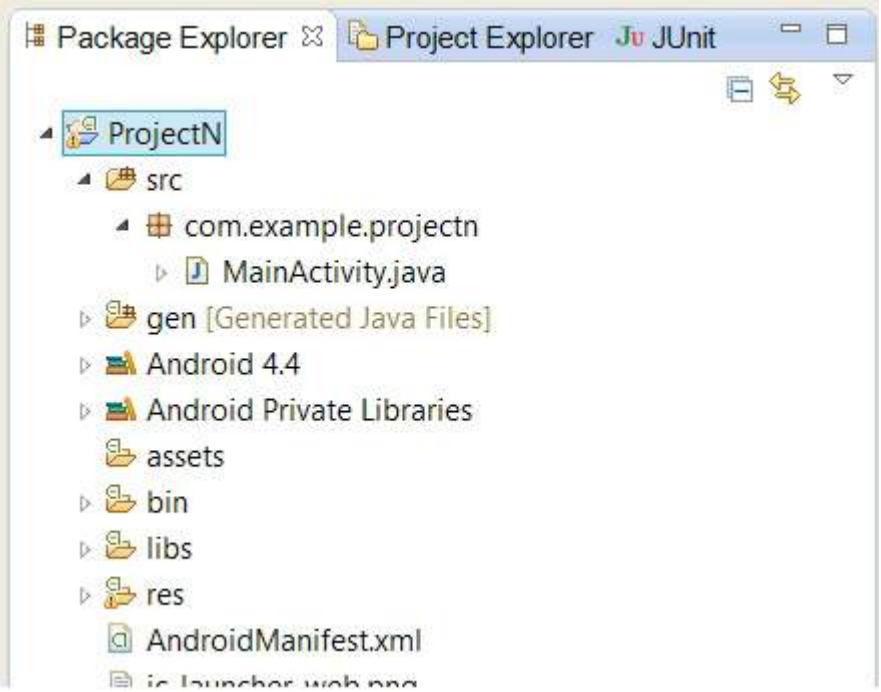
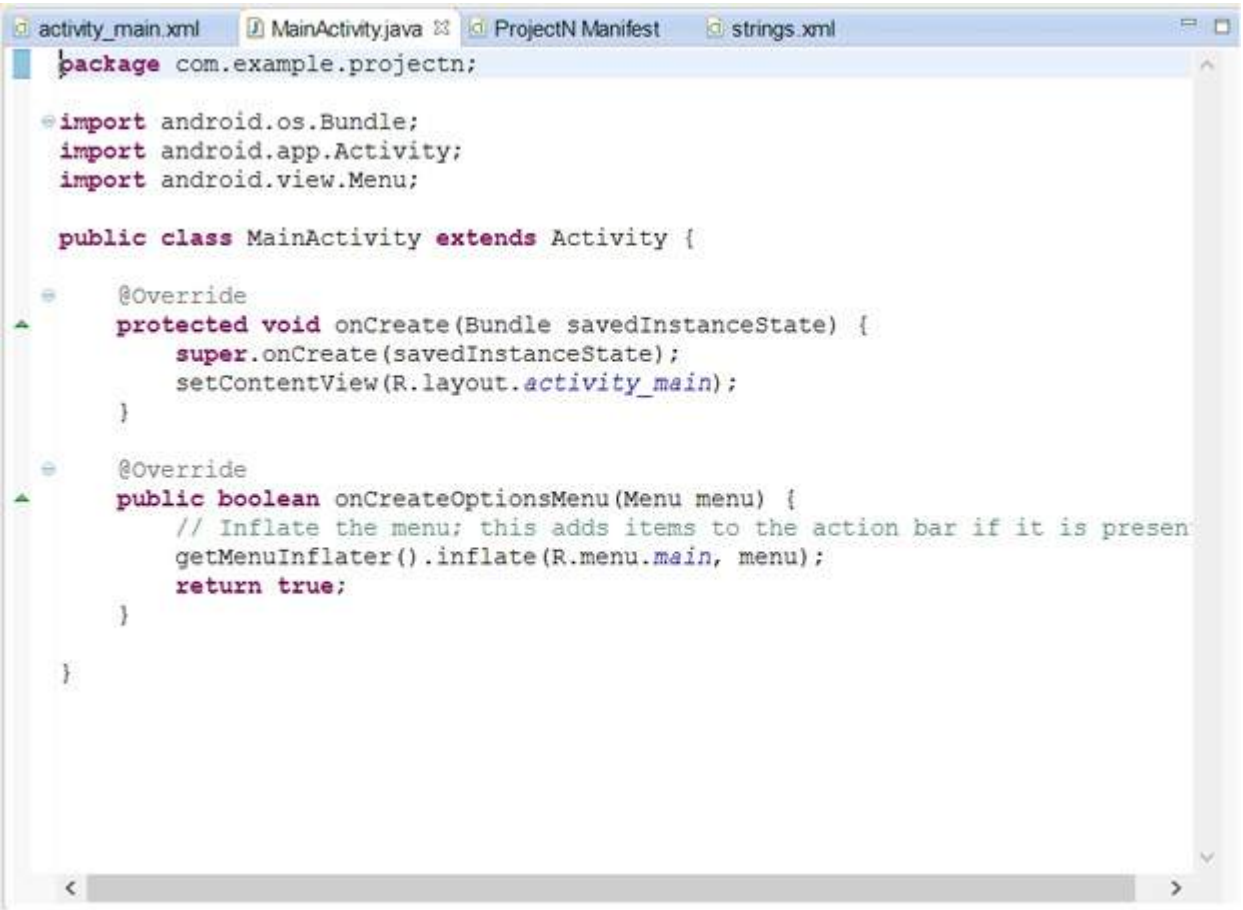


Рис. 4.13. Файл MainActivity.java в Package Explorer

Пока файл выглядит следующим образом, см. рис. 4.14.



увеличить изображение
Рис. 4.14. Файл MainActivity.java после создания приложения

Можно заметить, что класс MainActivity является наследником класса Activity и в нем уже реализован метод onCreate(), который запускается при первоначальном создании активности, нам потребуется его дополнить, но об этом чуть позже.

Мы предполагаем программно менять информацию в поле **TextView**, получать значение из поля **EditText** и обрабатывать события нажатия на кнопку **Button**, поэтому необходимо объявить соответствующие переменные, как поля класса MainActivity:

```
TextView tvInfo;
EditText etInput;
Button bControl;
```

Чтобы не было ошибок, необходимо импортировать пакет android.widget, который содержит все элементы графического интерфейса:

```
import android.widget.*;
```

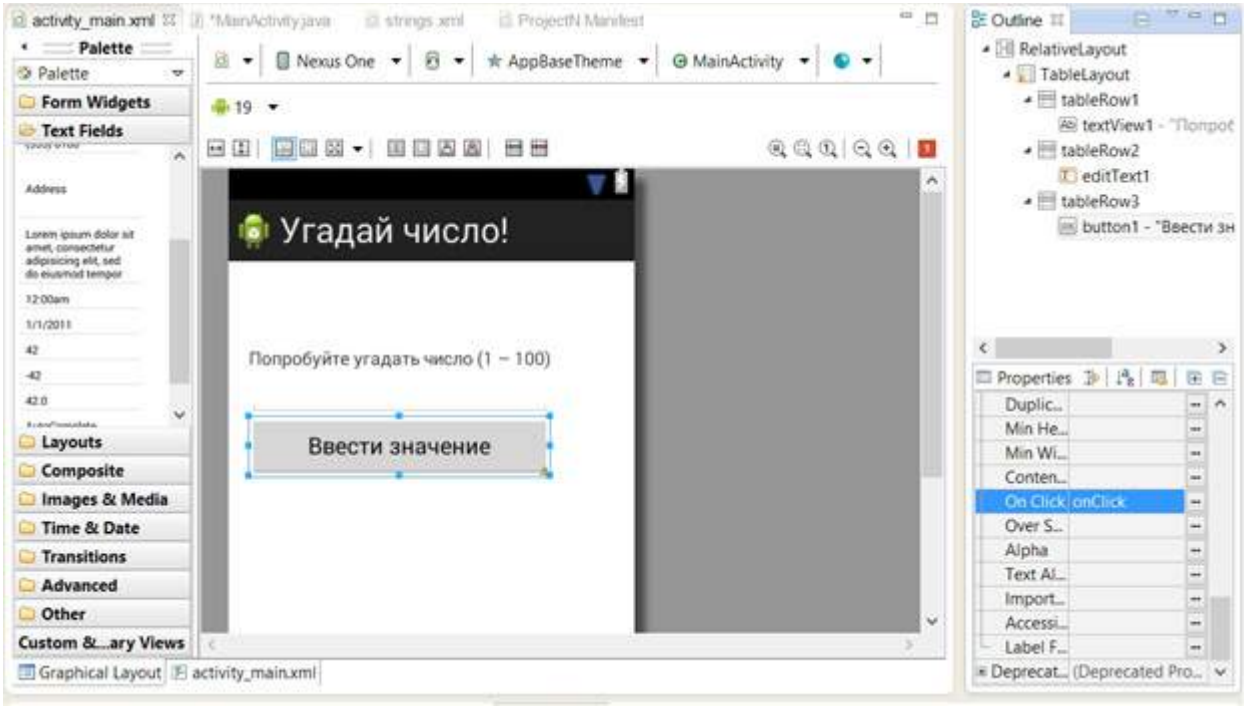
На самом деле среда разработки подскажет, что делать.

Теперь необходимо связать эти переменные с элементами интерфейса, уже добавленными нами в **activity_main.xml**, сделать это необходимо в методе onCreate(), а для получения уже созданного элемента интерфейса воспользуемся методом findViewById(). Итак в метод onCreate() добавим следующие строки:

```
tvInfo = (TextView)findViewById(R.id.textView1);
etInput = (EditText)findViewById(R.id.editText1);
bControl = (Button)findViewById(R.id.button1);
```

Метод findViewById() возвращает объект класса View, который является общим предком для всех компонентов пользовательского интерфейса, для того чтобы избежать возможных ошибок в скобках перед вызовом метода указываем до какого конкретно компонента необходимо сузить возможности объекта View.

Пришло время выполнить обработку нажатия на кнопку. Вернемся к файлу **activity_main.xml** в графический режим редактирования, выберем элемент **Button** и на вкладке со свойствами элемента найдем свойство **On Click** и запишем в него **onClick** - имя метода, который будет обрабатывать нажатие на кнопку. Как это выглядит показывает рис. 4.15.



увеличить изображение
Рис. 4.15. Настройка свойства On Click для кнопки

Эти же действия можно выполнить в фале **activity_main.xml**, достаточно дописать выделенную на рис. 4.16 строку:

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

Рис. 4.16. Настройка свойствами On Click для кнопки в файле XML

Для настройки свойств элементов интерфейса достаточно использовать любой способ: графический или редактирование XML файла.

Вернемся в файл **MainActivity.java**, в класс активности необходимо добавить метод:

```
public void onClick(View v){...}
```

Имя метода не обязательно должно быть onClick(), главное, чтобы оно совпадало с именем, указанным в свойстве **On Click**. В этом методе и будет происходить все наше программирование в этой лабораторной работе.

Нам потребуются две переменные:

- целочисленная для хранения загаданного числа (случайное число от 1 до 100);
- логическая для хранения состояния закончена игра или нет.

Обе эти переменные имеет смысл объявить как поля класса активности, первоначальные значения присвоить в методе onCreate.

Получить целочисленное значение из поля ввода, можно с помощью следующей конструкции:

```
Integer.parseInt(etInput.getText().toString())
```

изменить значение текста в информационном поле можно с помощью следующей конструкции:

```
tvInfo.setText(getResources().getString(R.string.ahead));
```

в данном случае в информационном поле появится значение строкового ресурса с именем ahead.

Осталось реализовать логику приложения в методе onClick(). Предлагаем написать код этого метода самостоятельно, для контроля в приложении предложен листинг, который содержит один из вариантов кода описанного приложения.

Задания для самостоятельной работы:

Разумеется предложенная в приложении реализация не идеальна, требует доработок:

- Что произойдет, если кнопка будет нажата до ввода какого-либо числа? Скорей всего приложение будет остановлено, необходимо как-то отслеживать этот вариант развития событий и адекватно реагировать. Предлагаем поработать самостоятельно над этой проблемой.
- Что произойдет, если пользователь введет число меньшее нуля или большее 100? Скорей всего приложение обработает этот ввод, но было бы лучше, если бы появилось сообщение о том, что введенное число не удовлетворяет условиям задачи.
- Как завершить приложение? И надо ли это делать?
- Предлагаем подумать, как еще можно улучшить приложение и реализовать эти улучшения.

4.5 Немного о работе с эмулятором

При выполнении данной работы мы использовали эмулятор для проверки работоспособности приложения. Рассмотрим некоторые возможности, облегчающие и ускоряющие взаимодействие с виртуальным устройством.

Во-первых, существует набор полезных комбинаций клавиш для управления виртуальным устройством:

- **Alt+Enter** - разворачивает эмулятор до размеров экрана;
- **Ctrl+F11** - меняет ориентацию эмулятора с портретной на альбомную и обратно;
- **F8** - включает/выключает сеть.

Полный список комбинаций клавиш для работы с эмулятором можно найти по ссылке: http://developer.android.com/tools/help/emulator.html.

Во-вторых, кто бы ни работал с эмулятором, тот на себе прочувствовал насколько терпеливым надо быть, чтобы взаимодействовать с ним, так медленно он работает. Существует решение для ускорения работы эмулятора Android и этим решением является Intel Hardware Accelerated Execution Manager (Intel® HAXM).

Intel Hardware Accelerated Execution Manager (Intel® HAXM) - это приложение с поддержкой аппаратной виртуализации (гипервизор), которое использует технологию виртуализации Intel для ускорения эмуляции приложений Android на компьютере для разработки. (http://software.intel.com/ru-ru/android/articles/intel-hardware-accelerated-execution-manager)

Intel HAXM способен ускорить работу эмулятора для x86 устройств. При этом эмулятор будет работать со скоростью, приближенной к скорости работы реального устройства, что поможет сократить время на запуск и отладку приложения. Подробно познакомиться с установкой Intel HAXM и настройкой эмулятора на работу с ускорителем можно в статье по ссылке: http://habrahabr.ru/company/intel/blog/146114/.

В-третьих, не стоит прерывать процесс запуска виртуального устройства, наберитесь терпения, на старых компьютерах первый запуск AVD может занять до 10 минут, на современных - от одной до трех минут. После того, как создали и запустили виртуальное устройство имеет смысл оставить его открытым, при всех последующих запусках приложения будет использоваться уже открытое виртуальное устройство, что позволит сэкономить время.

4.6 Заключение

В лабораторной работе рассмотрен процесс разработки простого приложения переднего плана. Описано создание активности, настройка интерфейса и реализация логики приложения. Других компонентов в приложении не предусмотрено. В последующих работах будут рассматриваться

приложения, содержащие несколько активностей. Смешанные приложения, работающие на переднем плане и при этом поддерживающие сервисы, работающие в фоновом режиме.

Приложение 1.

```
package com.example.projectn;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.widget.*;

public class MainActivity extends Activity {
  TextView tvInfo;
  EditText etInput;
  Button bControl;

  int guess;
  boolean gameFinished;

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

    tvInfo = (TextView)findViewById(R.id.textView1);
    etInput = (EditText)findViewById(R.id.editText1);
    bControl = (Button)findViewById(R.id.button1);

    guess = (int)(Math.random()*100);
    gameFinished = false;
  }

  @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;
  }
  public void onClick(View v){
    if (!gameFinished){
      int inp=Integer.parseInt(etInput.getText().toString());
      if (inp > guess)
        tvInfo.setText(getResources().getString(R.string.ahead));
      if (inp < guess)
        tvInfo.setText(getResources().getString(R.string.behind));
      if (inp == guess)
      {
        tvInfo.setText(getResources().getString(R.string.hit));
        bControl.setText(getResources().getString(R.string.play_more));
        gameFinished = true;
      }
    }
    else
    {
      guess = (int)(Math.random()*100);
      bControl.setText(getResources().getString(R.string.input_value));
      tvInfo.setText(getResources().getString(R.string.try_to_guess));
      gameFinished = false;
    }
    etInput.setText("");
  }
}
```

Внимание! Если Вы увидите ошибку на нашем сайте, выделите её и нажмите Ctrl+Enter.

© Национальный Открытый Университет "ИНТУИТ", 2016 | www.intuit.ru

Самостоятельная работа 3. Основы разработки интерфейсов мобильных приложений

Цель лабораторной работы:

Изучение основ разработки интерфейсов мобильных приложений

Задачи лабораторной работы:

- Изучить элементы интерфейса
- Практическим путём научиться размещать элементы и менять их свойства
- Разработать прототип интерфейса собственного приложения

6.1 Введение

Лабораторная работа посвящена разработке интерфейсов мобильных приложений. Работа содержит подробное описание построения гармоничного понятного пользовательского интерфейса для главной активности приложения и описание основных элементов интерфейса. Лабораторная работа поможет выбрать концепцию своего приложения и начать разработку его интерфейса.

Более подробную информацию о разработке интерфейсов мобильных приложений под Android можно узнать на сайте Design [Android Developers].

6.2 Создание прототипа интерфейса

Рассмотрим пример разработки интерфейса приложения, которое ищет в сети Интернет изображения по запросу пользователя, позволяет оценивать их, скачивать, и посещать интернет-страницы сайтов, на которых было найдено изображение.

6.2.1 Создание заготовки для приложения

Выглядеть главное окно будет примерно так:

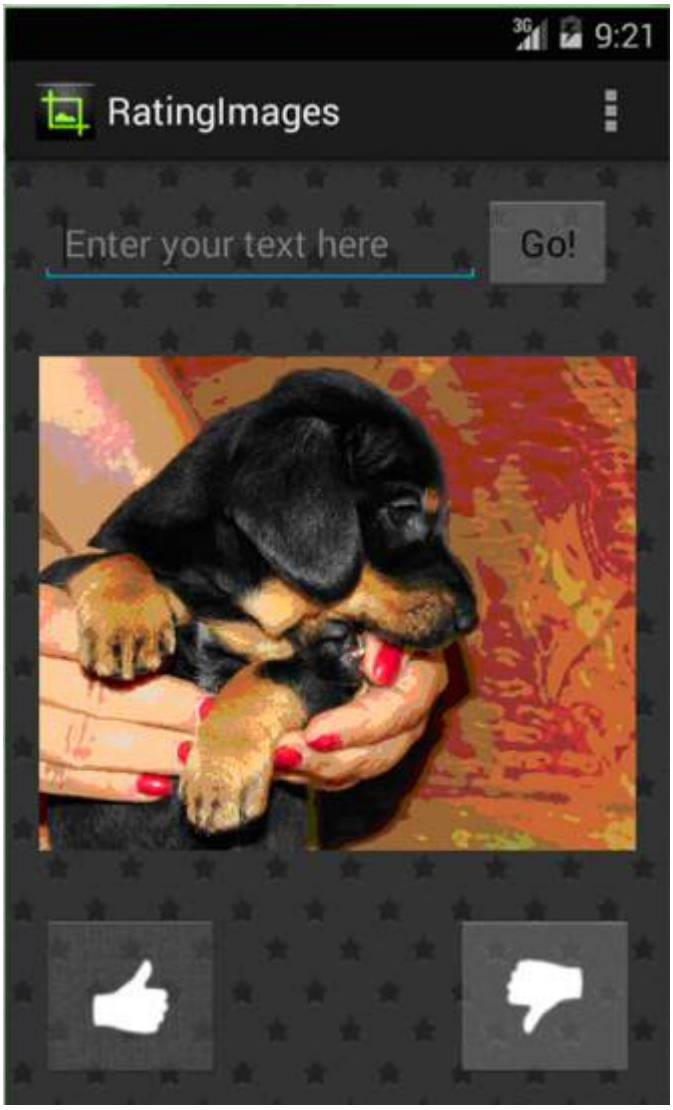


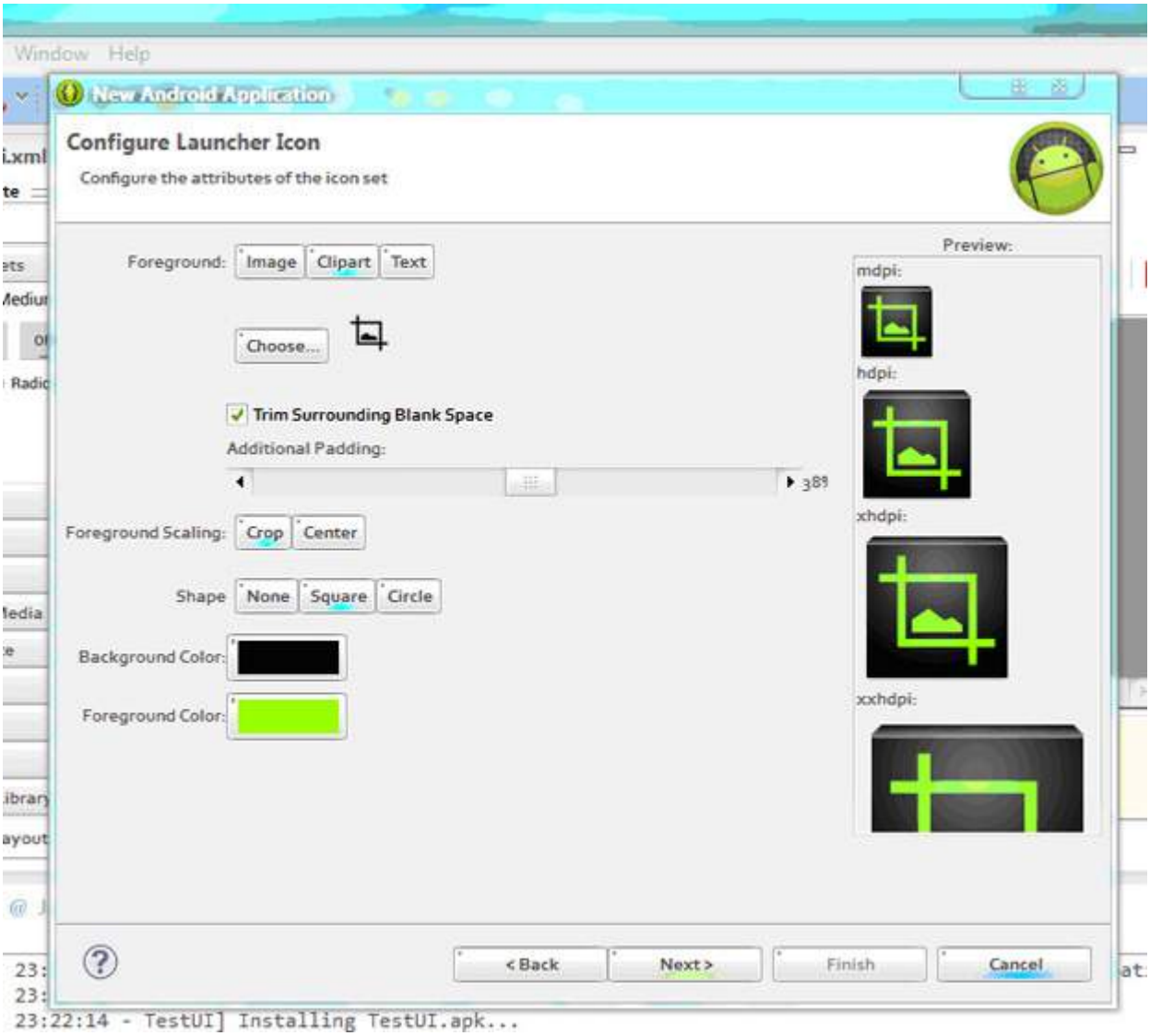
Рис. 6.1. Интерфейс главной активности

На нём присутствуют поле ввода текста для запроса пользователя и кнопка, начинающая поиск изображений. Внизу экрана две кнопки: "**like**" и "**dislike**", с их помощью пользователь сможет оценить изображение. После того, как пользователь сделает оценку изображения, текущее изображение закрывается и загружается следующее.

Итак, начнём с создания нового проекта. Назовём его "**RatingImages**" ("Рейтинг изображений").

На данном этапе изучения программирования под Android не обязательно менять иконку запуска, но эта лабораторная работа направлена на разработку дизайна интерфейса, и потому, приступим.

Создайте иконку на свой вкус.



увеличить изображение
Рис. 6.2. Создание иконки

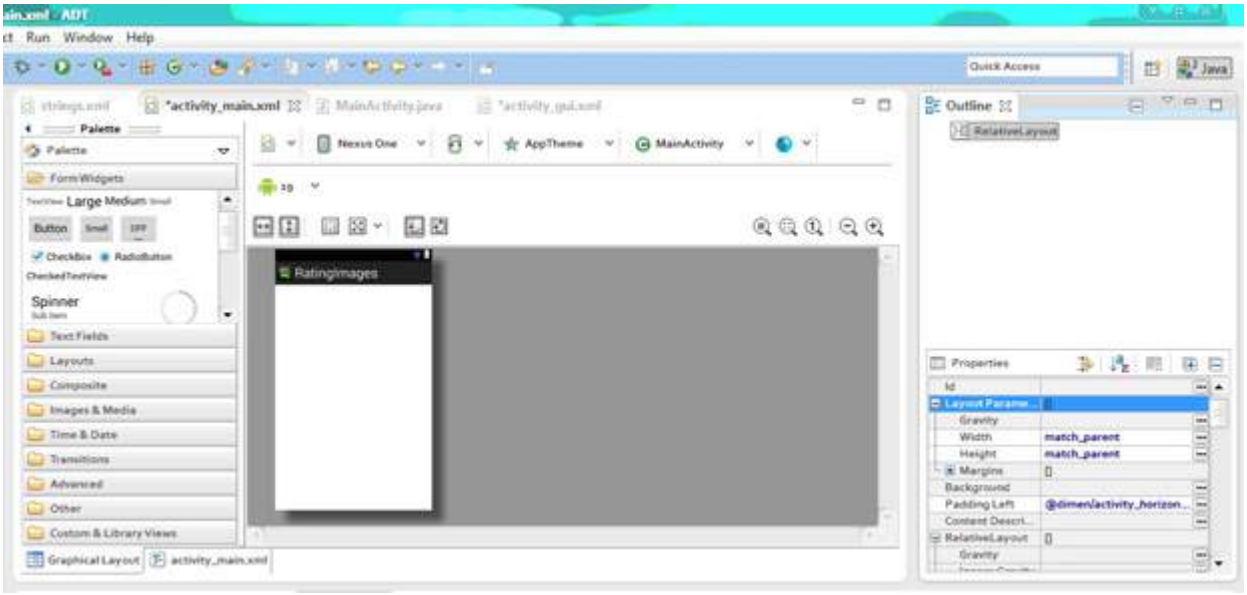
После создания проекта откройте **activity_main.xml** из каталога **res/layout/**.

Когда вы откроете файл **activity_main.xml**, вы увидите графический редактор макета. Благодаря этому редактору создание интерфейсов стало ещё интереснее, поскольку добавить элемент на форму можно при помощи перетаскивания мышью, к тому же, благодаря графическому редактору, не обязательно запускать эмулятор, чтобы увидеть результат своих трудов.

Теперь щелкните по вкладке **activity_main.xml** в нижней части экрана. Открылся XML-редактор кода. Этот способ редактирования стандартный, но все изменения, вносимые в этот документ, можно так же ощутить визуально, перейдя на графический редактор.

Вернёмся на вкладку с графическим редактором. Во-первых, подготовим документ к началу работы, для этого удалите <TextView>.

Результат выглядит так:



[увеличить изображение]
Рис. 6.3. Проект, готовый к началу разработки

На рабочей области экрана остался один элемент. Это макет <RelativeLayout>. В нём позиция дочерних элементов может быть описана по отношению друг к другу или к родителю. Подробнее о макетах можно узнать здесь.

Два атрибута, ширина и высота (android:layout_width и android:layout_height), требуются для всех элементов для того, чтобы указать их размер.

Так как <RelativeLayout> - это корень в макете, то нужно, чтобы он заполнял всю область экрана. Это достигается при помощи установки параметра "match_parent" для ширины и высоты. Это значение указывает, что ширина и высота элемента будет равна ширине и высоте родителя.

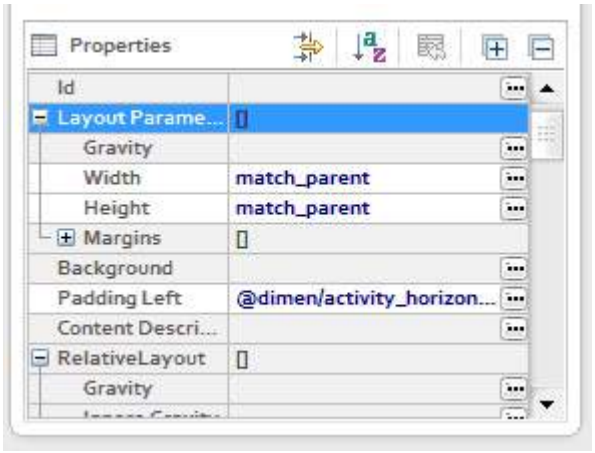


Рис. 6.4. Свойства <RelativeLayout> элемента

Как установить это значение? Будем разбираться.

Однократным щелчком левой кнопкой мыши по надписи "**Width**" активируйте строку с параметрами ширины:

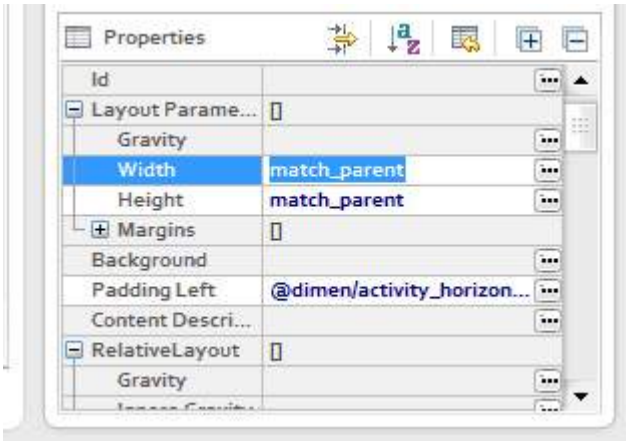


Рис. 6.5. Свойства <RelativeLayout> элемента, выбран атрибут android:layout_width

Щелчком левой кнопки мыши по области ввода вызовите диалоговое окно, и двойным щелчком сделайте выбор параметра:

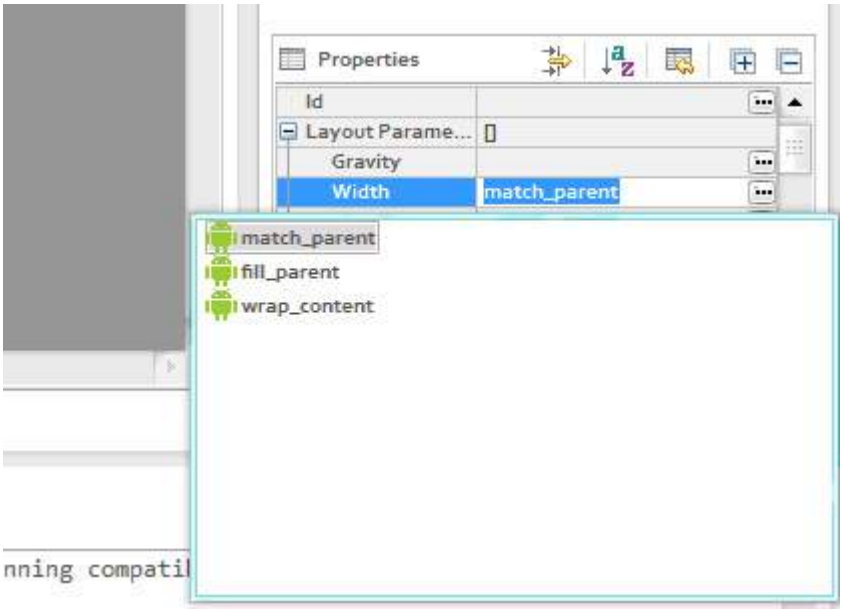


Рис. 6.6. Свойства <RelativeLayout> элемента, выбран параметр "match_parent"

Или щелчком правой кнопки мыши по <RelativeLayout> в Outline:

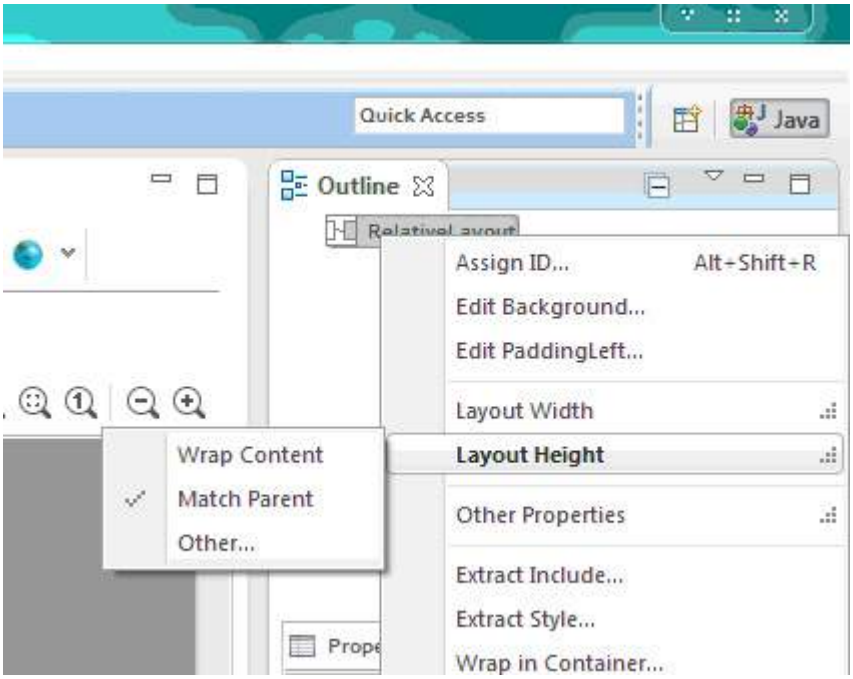


Рис. 6.7. Контекстное меню <RelativeLayout> элемента

При выполнении первого способа вы увидели еще два возможных параметра: "fill_parent" и "wrap_content".

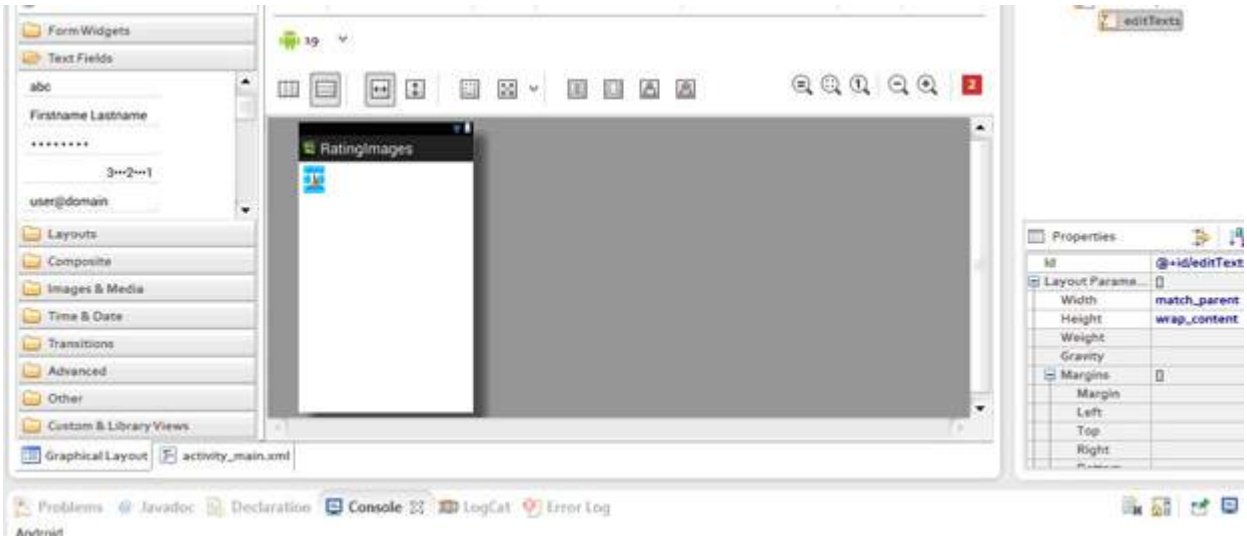
На самом деле, match_parent = fill_parent, но "fill_parent" считается устаревшим, и к использованию в новых проектах предлагается "match_parent".

Параметр "wrap_content" указывает, что представление будет увеличиваться при необходимости, чтобы поддерживать соответствие содержанию экрана.

6.2.2 Добавление текстового поля

Для начала добавьте элемент <LinearLayout> с горизонтальной ориентацией в <RelativeLayout>, и укажите для ширины и высоты параметр "wrap_content". Теперь, для создания пользовательского редактируемого текстового поля, добавьте элемент <EditText> с параметром "wrap_content" для ширины и высоты в <LinearLayout>.

Сейчас должно получиться примерно следующее:



увеличить изображение
Рис. 6.8. Добавление текстового поля

Возможно, появился желтый предупреждающий знак, но сейчас это не важно, со временем он исчезнет. Наличие таких предупреждений никак не влияет на компилируемость проекта.

Теперь переходим к настройке добавленных нами элементов.

Для многих элементов нужно назначать id, он обеспечивает уникальный идентификатор, который можно использовать как ссылку на объект из кода вашего приложения для управления им.

Откроем редактор кода XML-файла и обратим внимание на элемент

```
<EditText>:
        <EditText
                android:id="@+id/editText1"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content">
                <requestFocus/>
        </EditText>
```

Строка <requestFocus/> появляется добавлением элемента requestFocus (папка Advanced), и позволяет установить фокус на нужном компоненте. Важно использовать этот элемент, когда у вас имеется, к примеру, три текстовых поля, и нужно, чтобы фокус был на втором из них.

При указании id, знак (@) требуется в том случае, если вы имеете в виду любой ресурс объекта из XML-файла. За ним следуют тип ресурса (в данном случае ID), косая черта (слеш) и имя ресурса (editText1).

Знак плюс (+) перед типом ресурсов необходим только тогда, когда вы впервые определяете идентификатор ресурса.

По сути, id, который создается автоматически, уже уникален, но грамотнее переименовывать id в соответствии со назначением элемента.

Зададим id для текстового поля. Для этого прямо в коде строку android:id="@+id/editText1" заменяем на android:id="@+id/edit_message", жмём **CTRL+S** и открываем графический редактор. Если всё хорошо, то в свойствах текстового поля в графе **id** будет следующее:

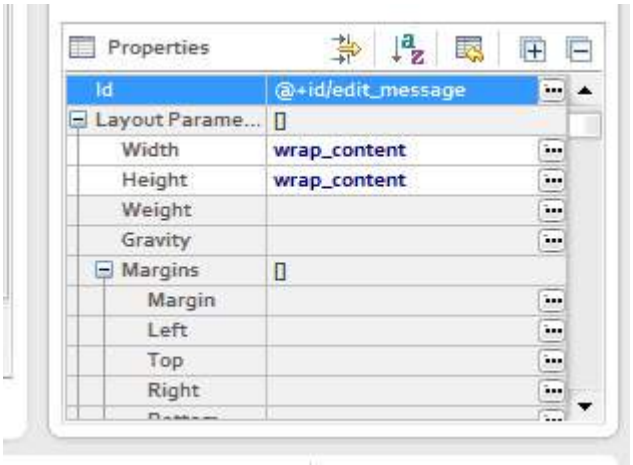


Рис. 6.9. Идентификатор текстового поля

Добавим в код ещё две строки:

android:ems="10" - задает соответствия для симметричного отображения шрифтов,

android:hint="@string/edit_message" - содержание тестового поля "по умолчанию", т.е. пока пользователь не начал вводить в поле текст. Вместо того, чтобы использовать просто слово (например android:hint="message"), что крайне не удобно при изменении основного языка приложения, используется ссылка на значение, хранящееся в файле **strings.xml**. Поскольку это относится к конкретному ресурсу (а не только к **id**), знак плюс не нужен.

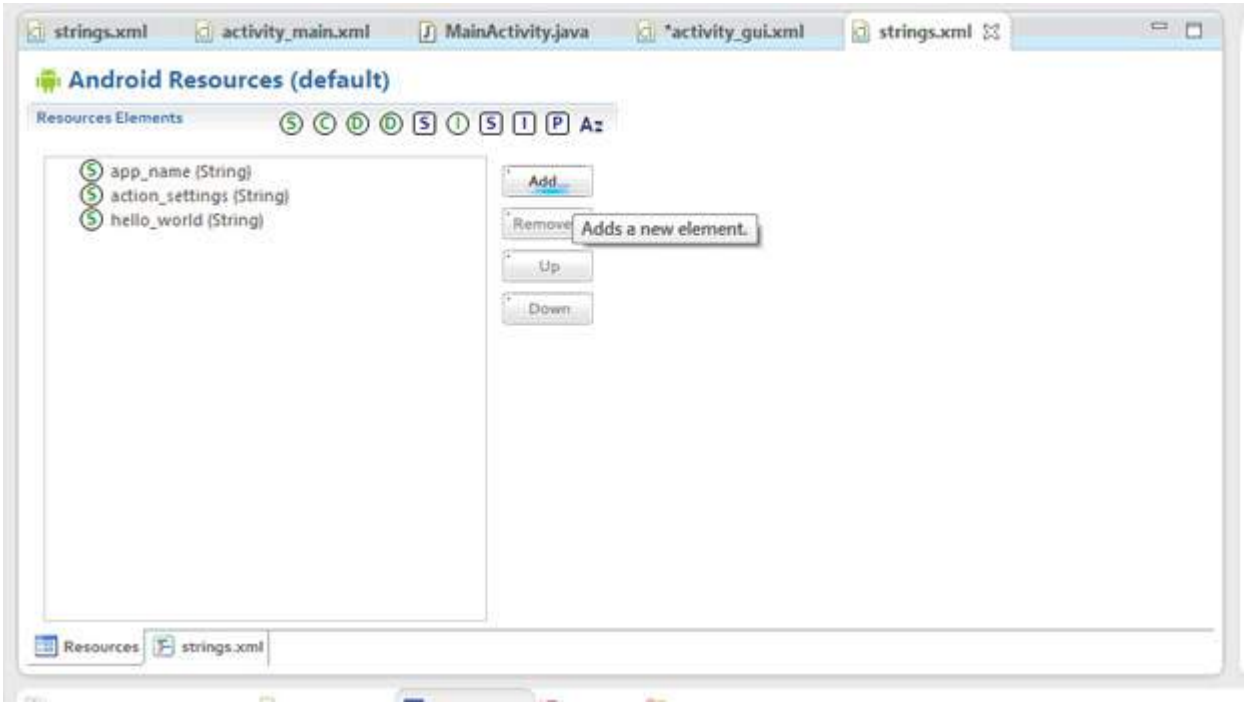
Однако, мы ещё не определили строку ресурсов файле **strings.xml**, и потому вы получите следующее:

Рис. 6.10. Предупреждение: не найден ресурс, на который прописана ссылка

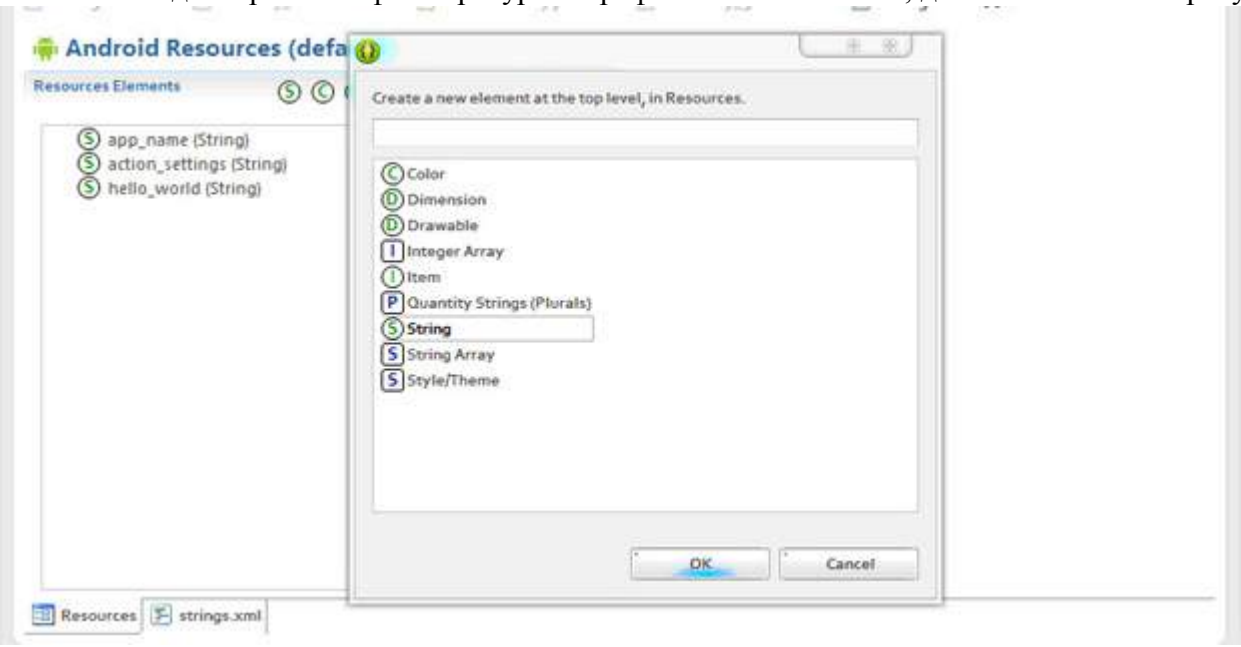
Для того, чтобы ссылка на ресурс начала работать, нужно этот ресурс создать.

Откройте файл **res/values/strings.xml**. Очевидно, что его тоже можно редактировать двумя способами: графически и вручную.

Выбирайте тот способ, который больше по душе.



увеличить изображение
Рис. 6.11. Редактирование файла ресурсов графическим способом; добавление нового ресурса



увеличить изображение
Рис. 6.12. Редактирование файла ресурсов графическим способом; выбор типа ресурса, выбран тип String

Заполняем поля "**Имя**" и "**Значение**":

увеличить изображение
Рис. 6.13. Редактирование файла ресурсов графическим способом

Сохраняем и любуемся результатом:

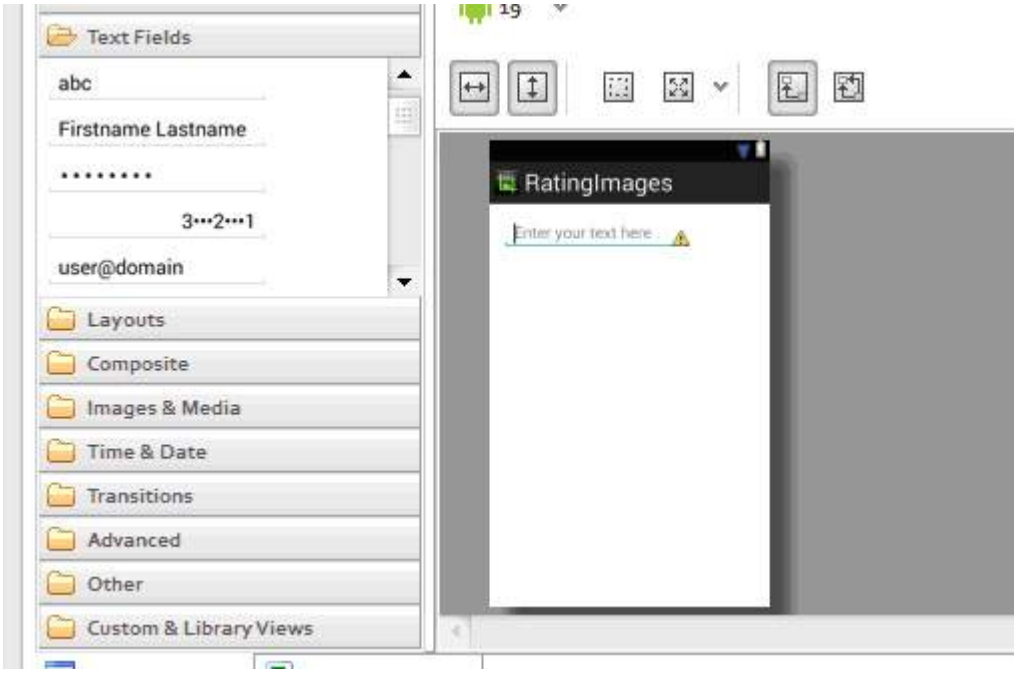


Рис. 6.14. Отображение текста "По умолчанию" в поле ввода

Подробнее о ресурсах - здесь.

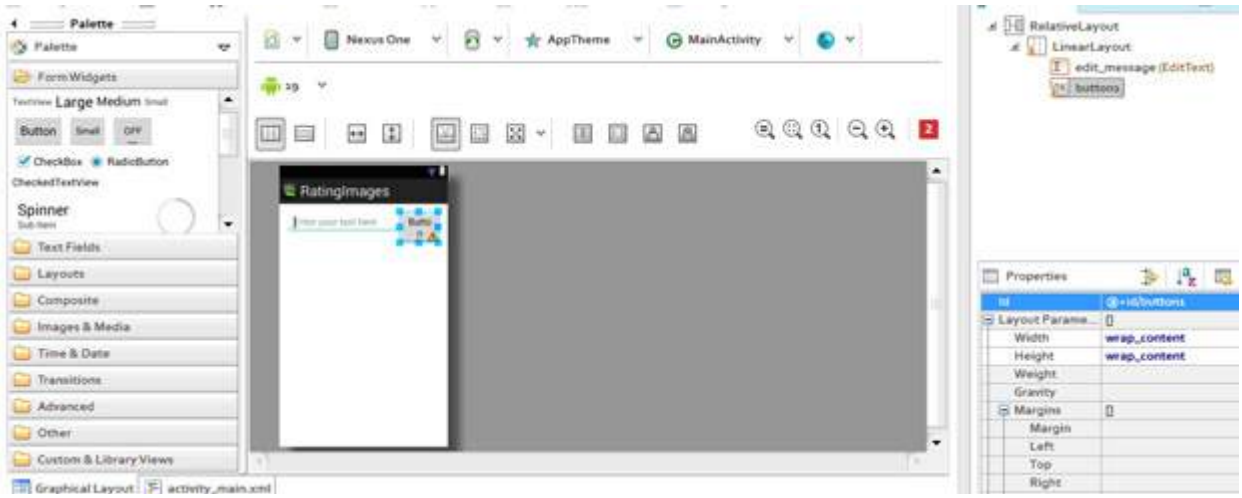
Аналогично создадим id для <LinearLayout>

```
android:id="@+id/linear1"
```

Сохраните изменения.

6.2.3 Добавление кнопки

Теперь добавьте <Button> в макет после элемента <EditText>:



увеличить изображение
Рис. 6.15. Новая кнопка

Чтобы кнопка трансформировалась в соответствии с текстом кнопки, ширина и высота должны быть установлены во "wrap_content".

Теперь поменяем надпись на кнопке на "**Go!**" с помощью ссылки на ресурс в XML-коде главной активности и добавления одного ресурса в файл **strings.xml**:

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

    <string name="app_name">RatingImages</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="edit_message">Enter your text here</string>
    <string name="button_send">Go!</string>

</resources>
```

Рис. 6.16. Редактор XML-кода файла strings.xml

Сохраните.

На графическом редакторе главной активности будут изменения:

Рис. 6.17. Кнопка приняла новую форму, в соответствии с надписью на ней

Теперь, когда мы поместили два главных представления на <LinearLayout> элемент, настало время добавить ещё два параметра для этого элемента.

Речь идет о "приращении" правого и левого краёв лейаута к правому и левому краям <RelativeLayout> элемента соответственно. А сделать это проще простого - просто потяните мышкой один край к другому!

Рис. 6.18. "Приращение" правого края

Зелёные стрелки по краям дают понять, к какому элементу удалось прицепить край:

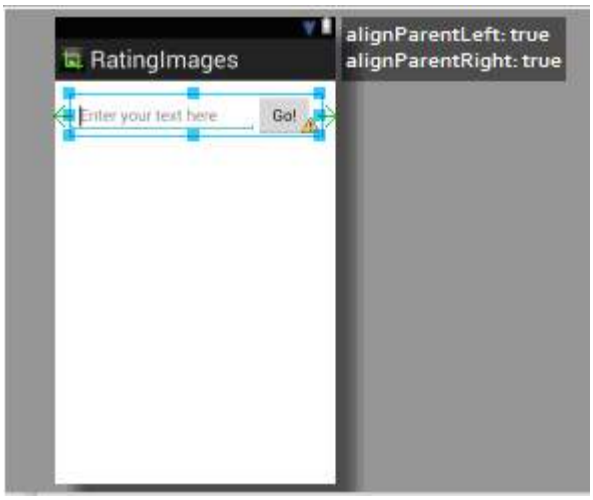


Рис. 6.19. <LinearLayout> после выравнивания

И, конечно, это отобразится в свойствах:

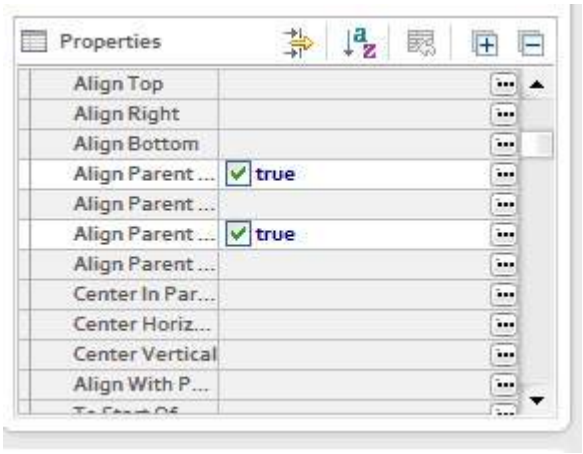


Рис. 6.20. Свойства <LinearLayout>

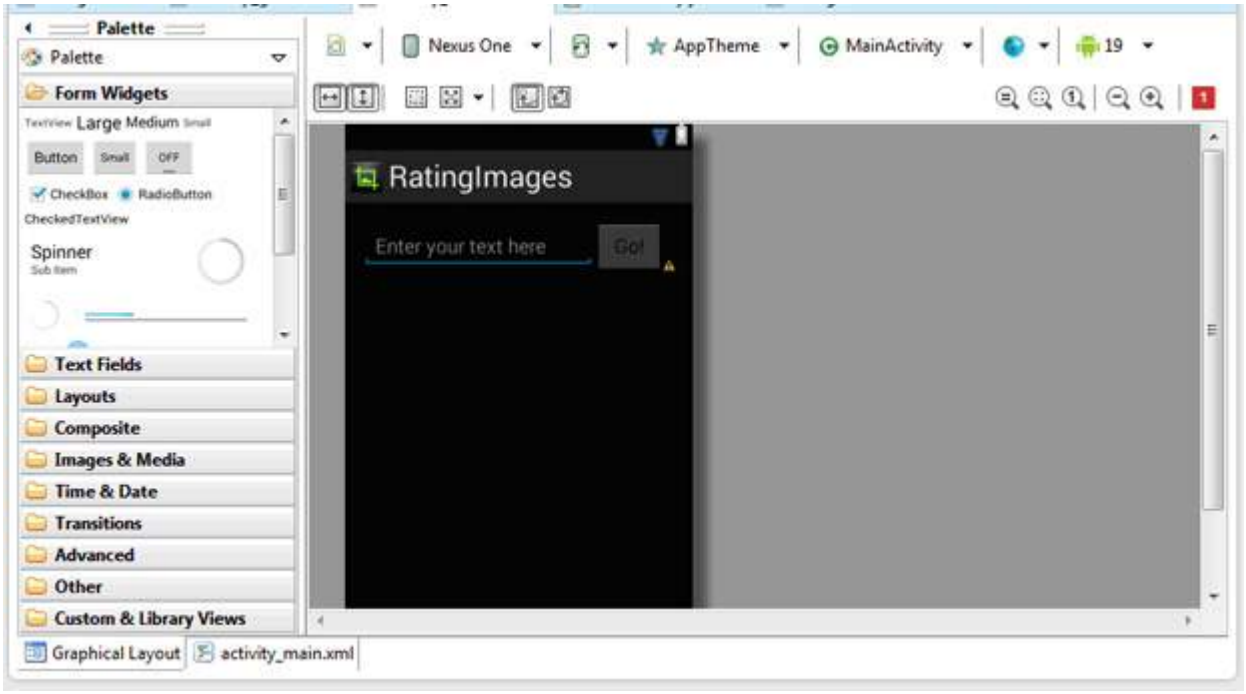
6.2.4 Смена фона

Идём дальше - попробуем поменять фон.

Чтобы изменить цвет фона на чёрный, нужно в XML-коде главной активности написать одну строку в блоке <RelativeLayout> элемента:

```
android:background="#000000".
```

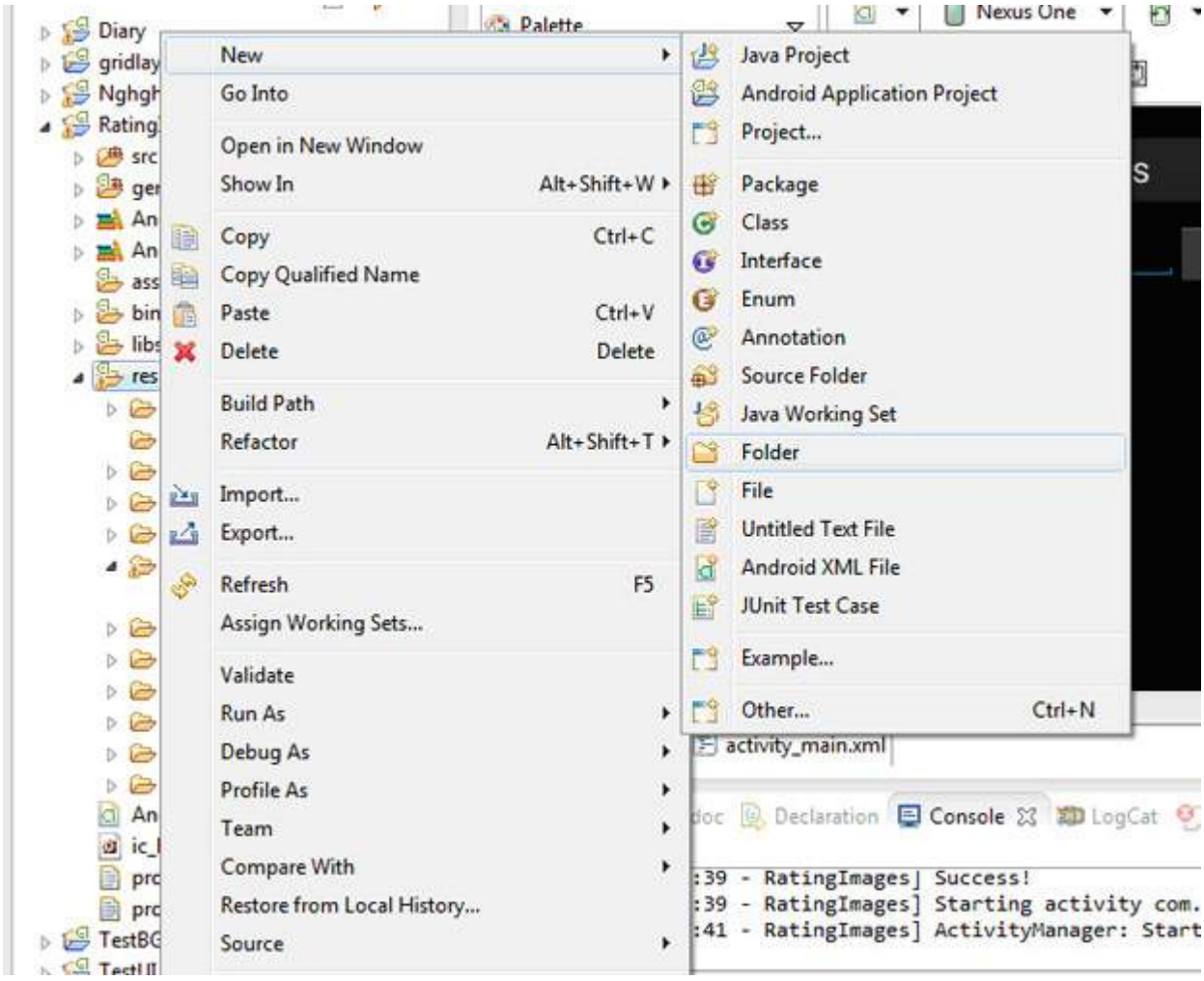
Сохраните и проверьте результат, открыв графический редактор.



[увеличить изображение]
Рис. 6.21. Фон стал чёрным

Красиво, но скучно. Как сделать фон ещё интереснее? Поместить на него рисунок!

Для этого сначала в папке **res/** создадим папку **drawable/**



увеличить изображение

Рис. 6.22. Создание папки

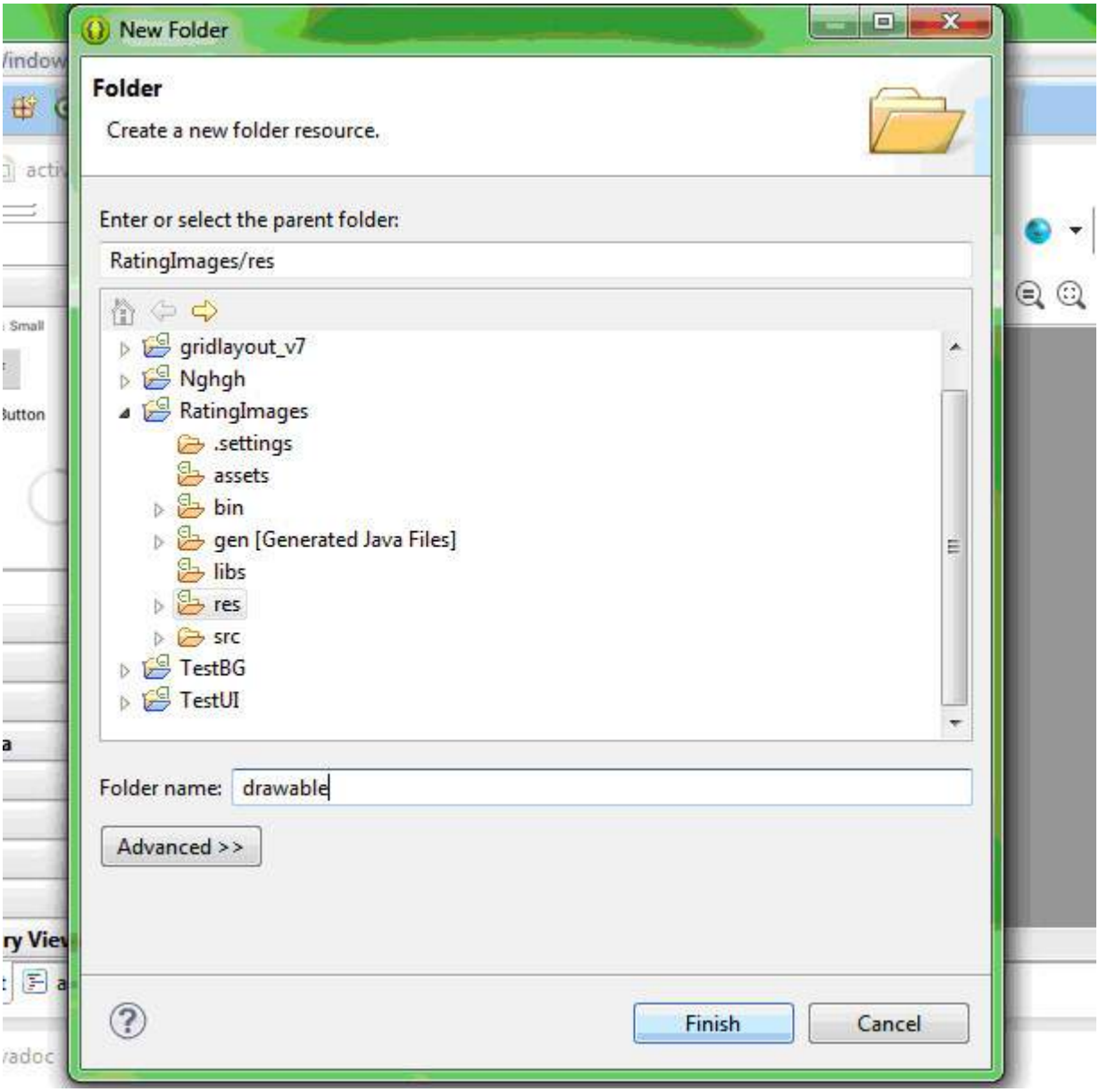


Рис. 6.23. Создание папки: имя папки

После того, как папка создана, нужно положить в эту папку изображение - картинка называется **got.png**:

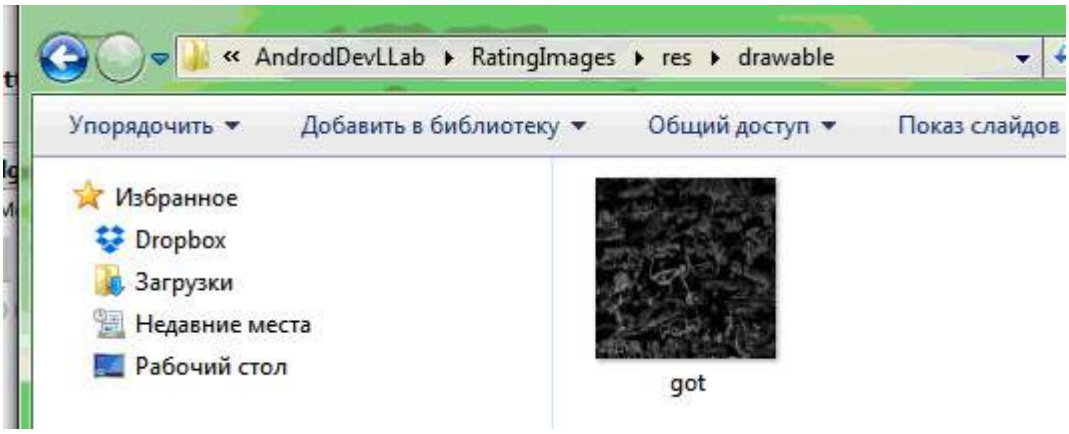
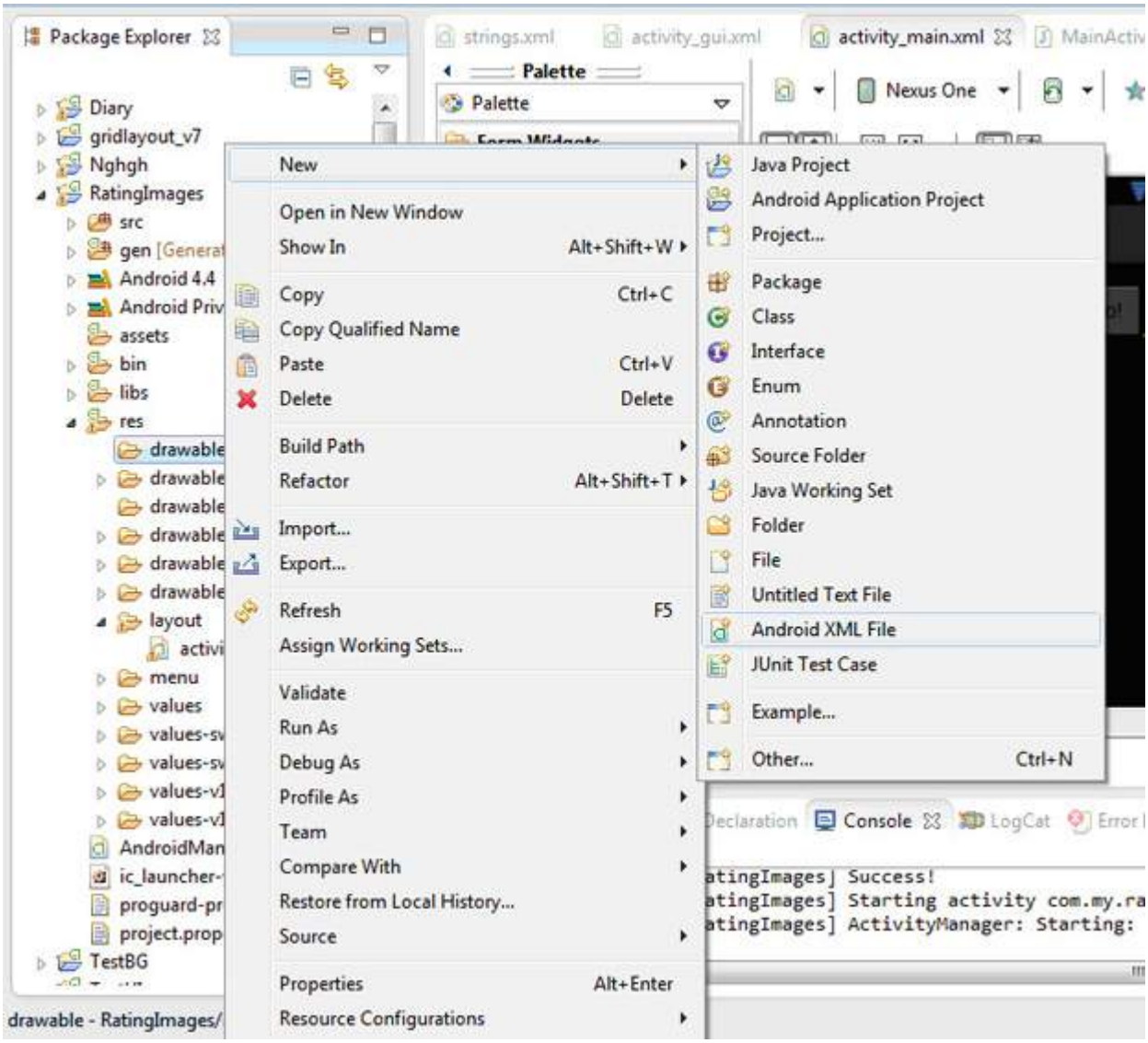


Рис. 6.24. Изображение в папке drawable/

После этого в папке **drawable/** нужно создать файл **background.xml**, важно при создании выбрать параметр bitmap.



[увеличить изображение]
Рис. 6.25. Создание нового XML-файла

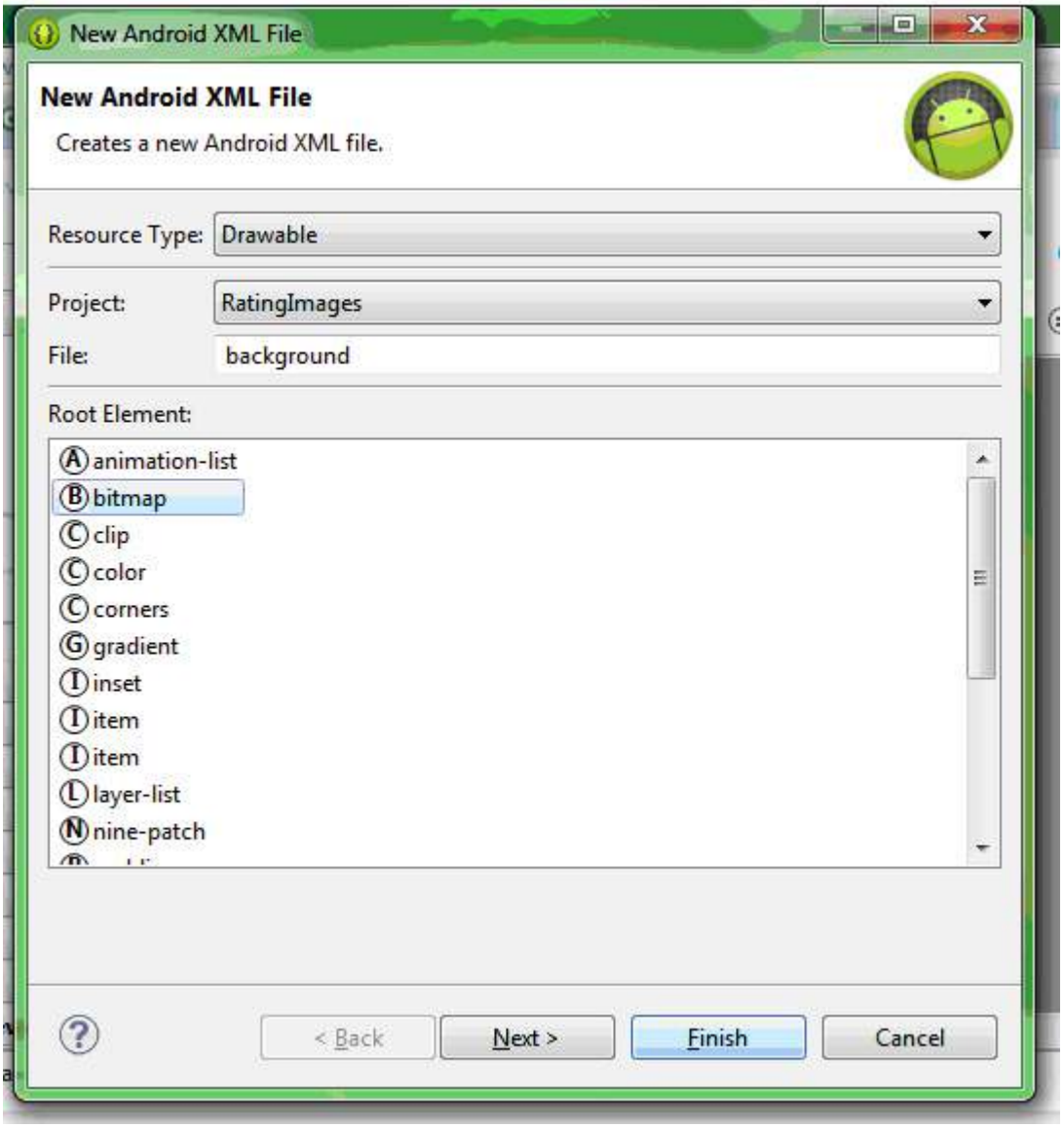


Рис. 6.26. Создание нового XML-файла

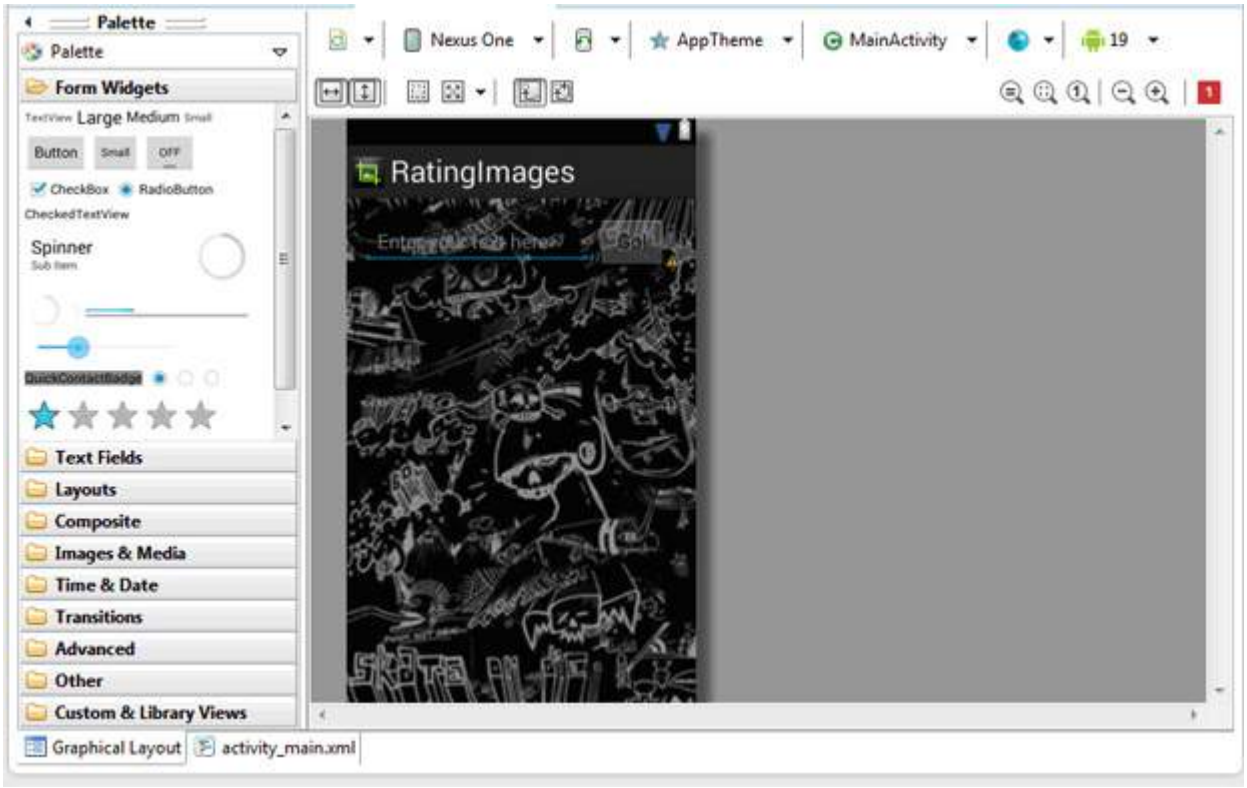
Как только новый файл открылся, пропишем в него одну строчку, с указанием на то, откуда и какой файл использовать:

```
<?xml version="1.0" encoding="utf-8"?>
<bitmap xmlns:android="http://schemas.android.com/apk/res/android"
        android:src="@drawable/got">
</bitmap>
```

Вернемся в редактор XML-кода, туда, где прописывали цвет фона.

Вместо строки android:background="#000000" напишем ссылку на XML-файл android:background="@drawable/background".

Сохраняем и видим результат:



[увеличить изображение]
Рис. 6.27. Новый фон

Несомненно, фон смотрится хорошо, но очевидно, что кнопка и поле ввода просто затерялись, а это значит, что для этого приложения такой фон не подходит. Можно продолжить подбирать изображения на фон, но лучше создать черепичную заливку небольшим изображением. На этом сайте можно найти узор на любой вкус!

Когда вы выбрали узор и скачали его, скопируйте изображение в папку **drawable/**.

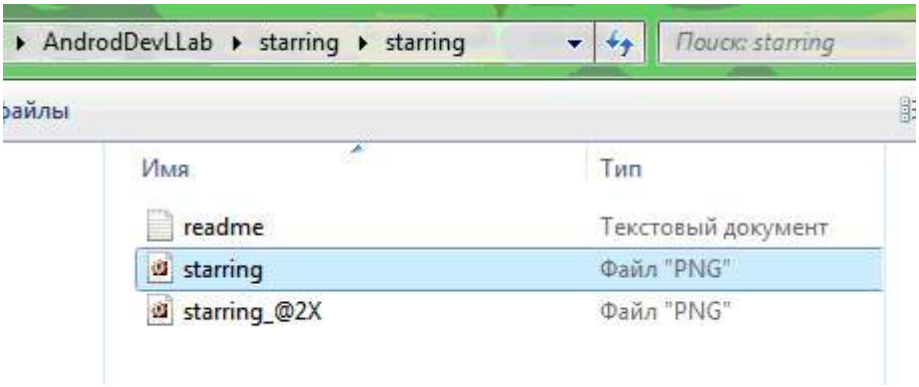


Рис. 6.28. Копирование изображения

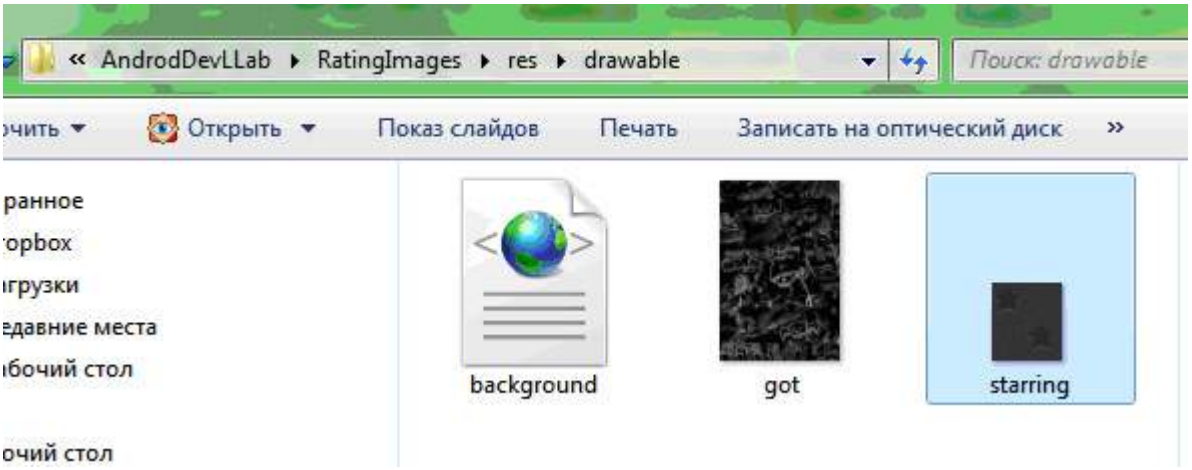


Рис. 6.29. Скопированное изображение

Теперь немного изменим файл **background.xml**.

Во-первых, нужно изменить имя изображение со старого на новое.

```
android:src="@drawable/starring"
```

Во-вторых, добавим такую строчку:

```
android:tileMode="repeat"
```

Сохраняем.

Атрибут android:tileMode задает тип заполнения, в данном случае простое повторение исходного изображения. Кроме repeat возможны варианты clamp и mirror. Помните, что данный приём применим только к bitmap, к фигурам, созданным при помощи XML, применить данную операцию нельзя.

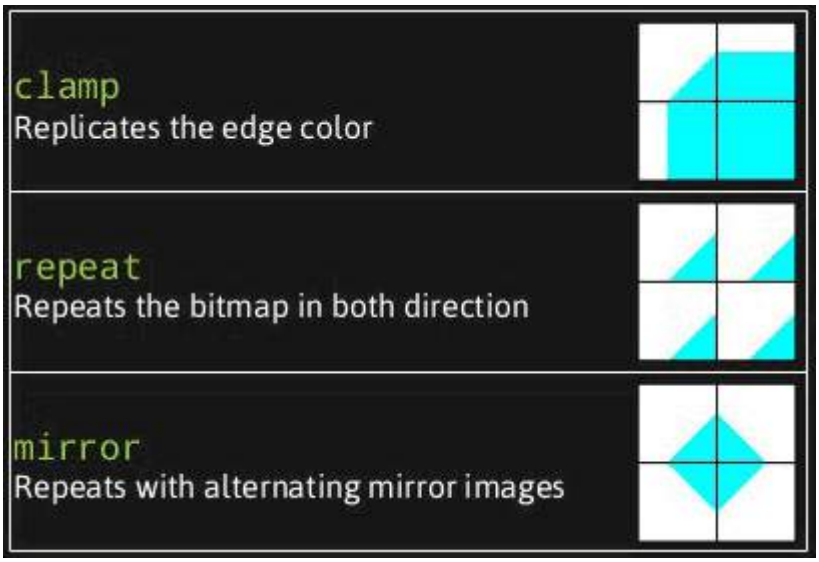
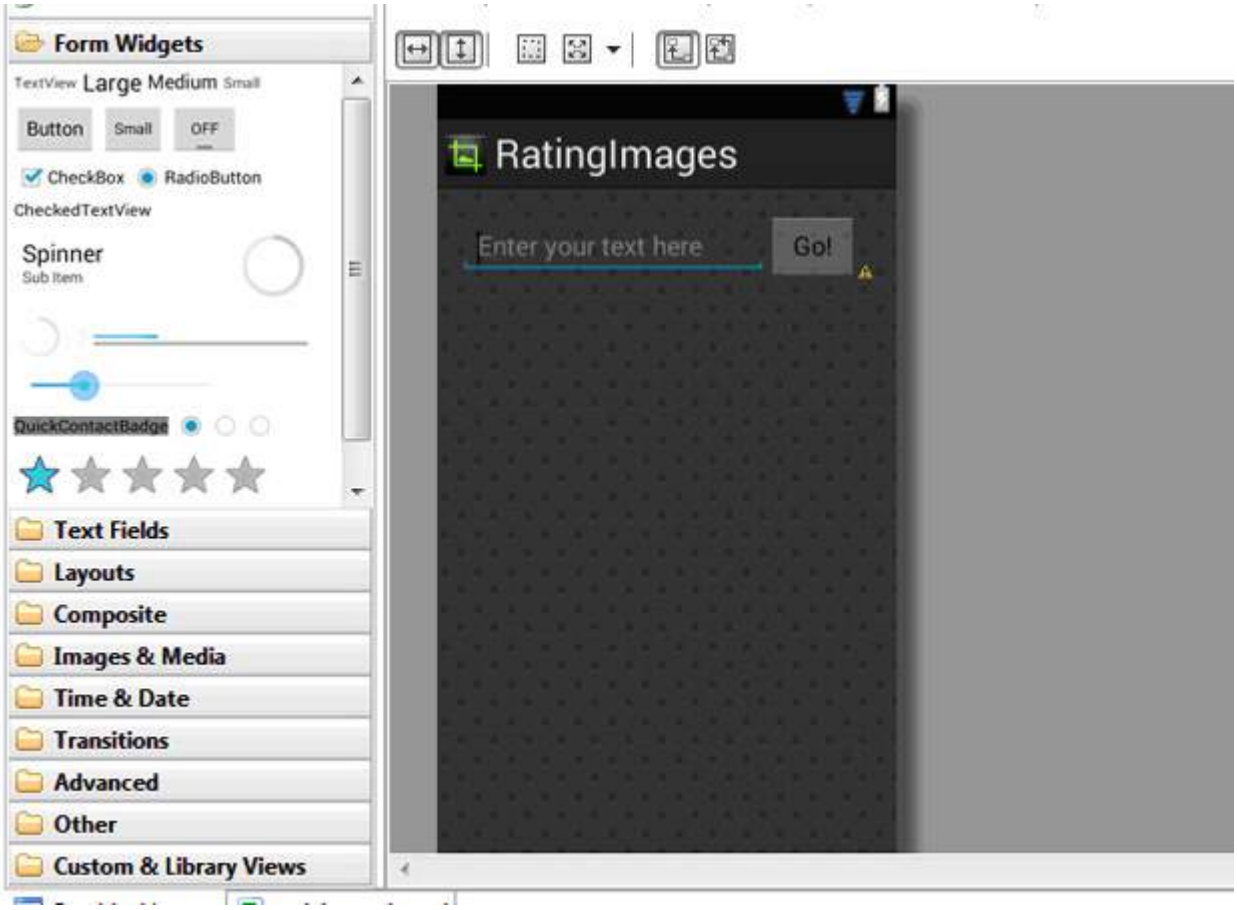


Рис. 6.30. Варианты заполнения

Настало время посмотреть, что из этого получилось:



увеличить изображение
Рис. 6.31. Фон из звездочек

В <RelativeLayout> стоит добавлять android:background только в том случае, если вы хотите неподвижный фон, а в <ScrollView> чтобы фон прокручивался вместе с контентом.

Если вы выбрали тёмный фон, то стоит поменять цвет текста, вводимого в поле ввода, например на белый.

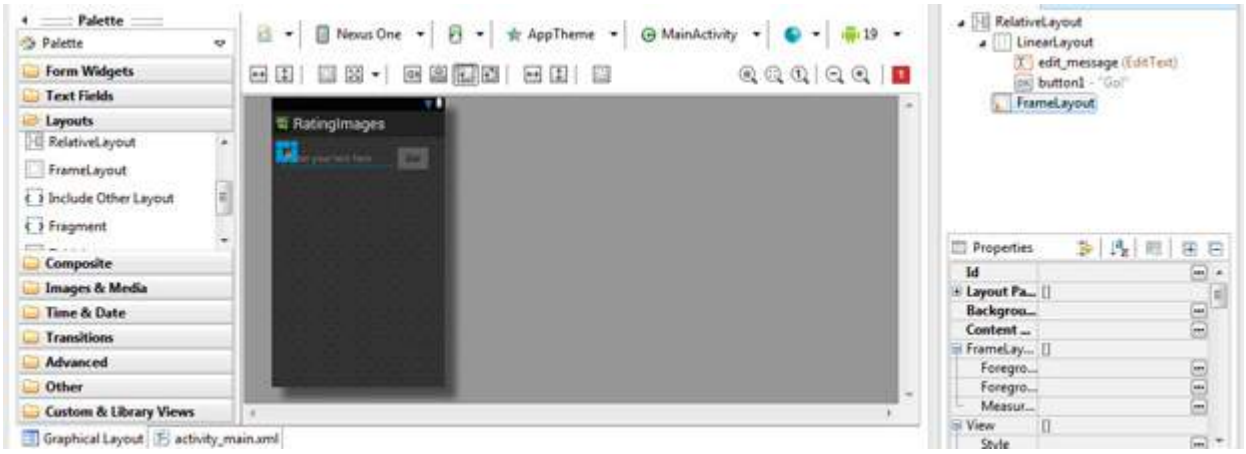
Для этого в блок <EditText> добавим строчку

```
android:textColor="#ffffff"
```

6.2.5 Область просмотра изображений

Теперь займемся созданием области отображения изображений, которые пользователь будет оценивать.

Добавьте "рамку" <FrameLayout> в <RelativeLayout>:



[увеличить изображение]
Рис. 6.32. Добавление "рамки"

Обратите внимание, что предупреждающий жёлтый треугольник исчез, но появился у нового элемента, и это значит, что всё идёт по плану.

Укажем этому элементу ширину, высоту и id. Предупреждение должно пропасть.

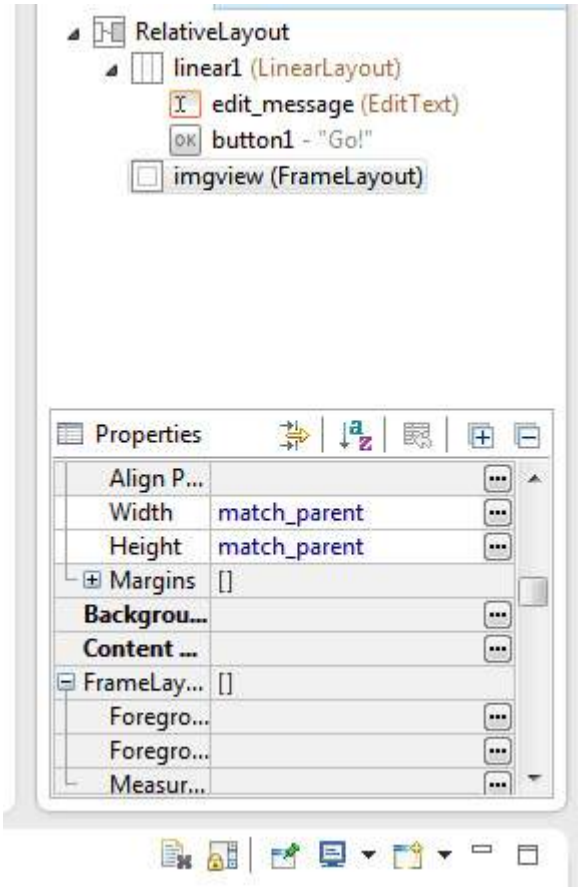
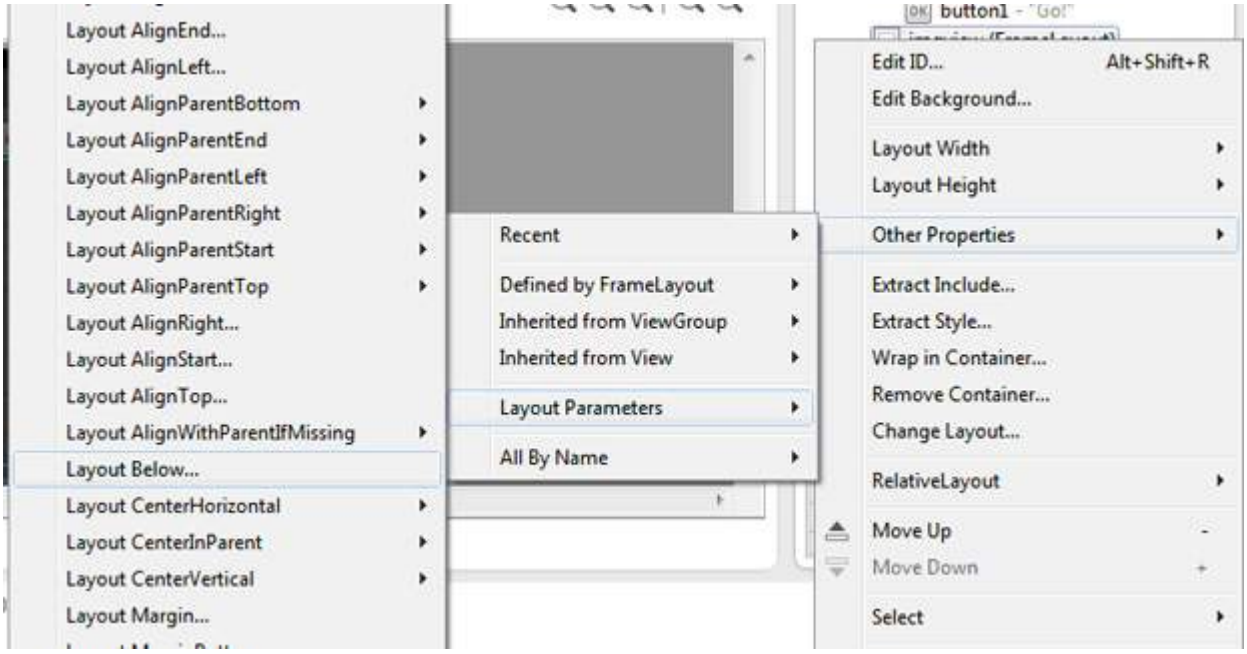


Рис. 6.33. Свойства "рамки"

Сейчас **FrameLayout** "заезжает" на поле ввода и кнопку. Чтобы исправить это, нужно задать для "рамки" параметр, который будет следить, чтобы "рамка" находилась ниже, чем поле ввода и кнопка.

Выберите из списка **Outline** "рамку" и кликом правой кнопки мыши вызовите контекстное меню: **Other Properties -> Layout Parametrs -> Layout Below…**



[увеличить изображение]
Рис. 6.34. Контекстное меню "рамки"

Появится окно. В нём выбираем из списка "**ID**" имя лейаута, на котором находятся поле ввода и кнопка:

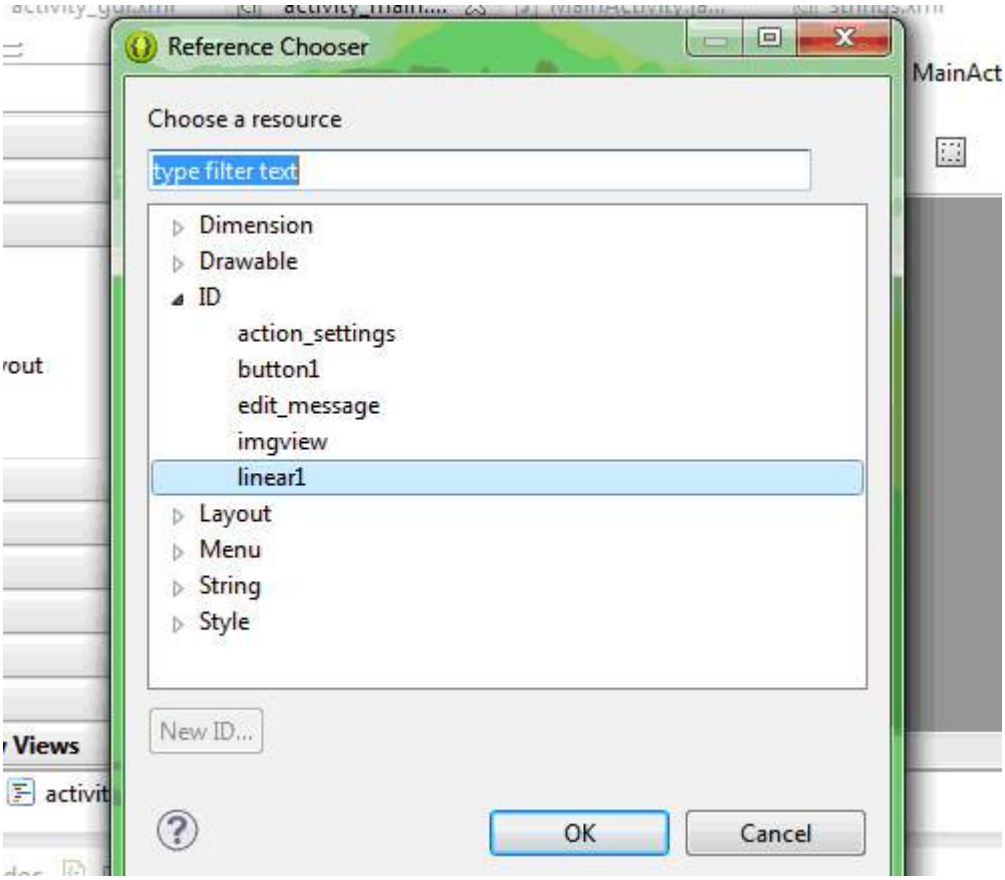
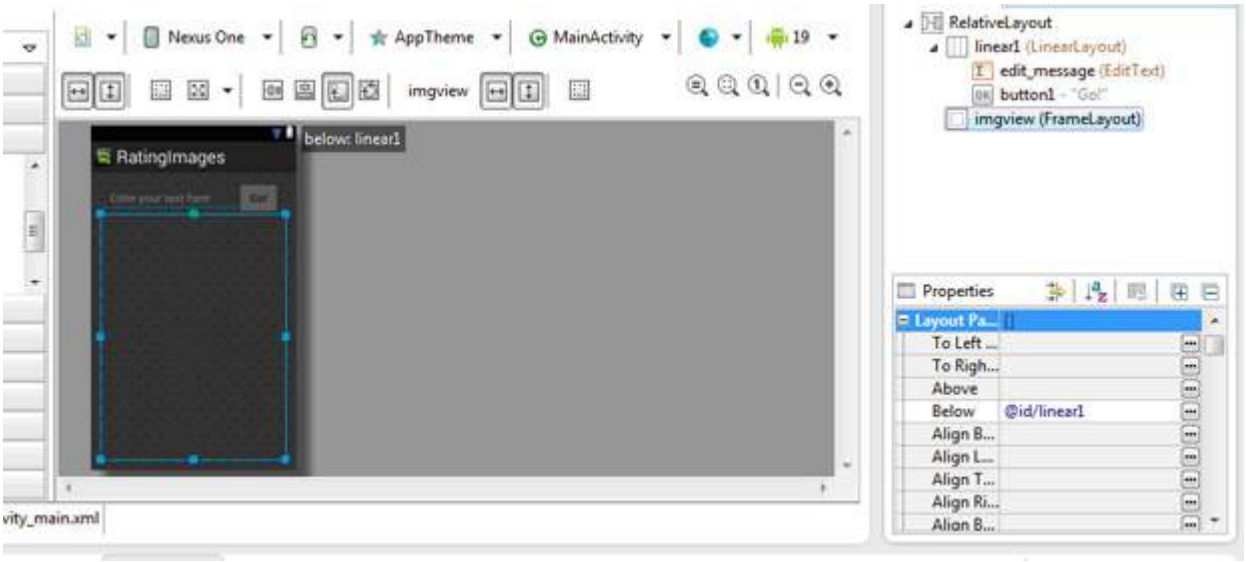


Рис. 6.35. Выбор элемента, ниже которого должна быть "рамка"

Жмём **ОК** и "рамка" примет вид:



[увеличить изображение]
Рис. 6.36. Новое расположение "рамки"

Теперь для наглядности поместим туда изображение.

Сначала поместим само изображение в папку **res/drawable/** и обновим её.

Это нужно для того, чтобы новые данные загрузились в проект.

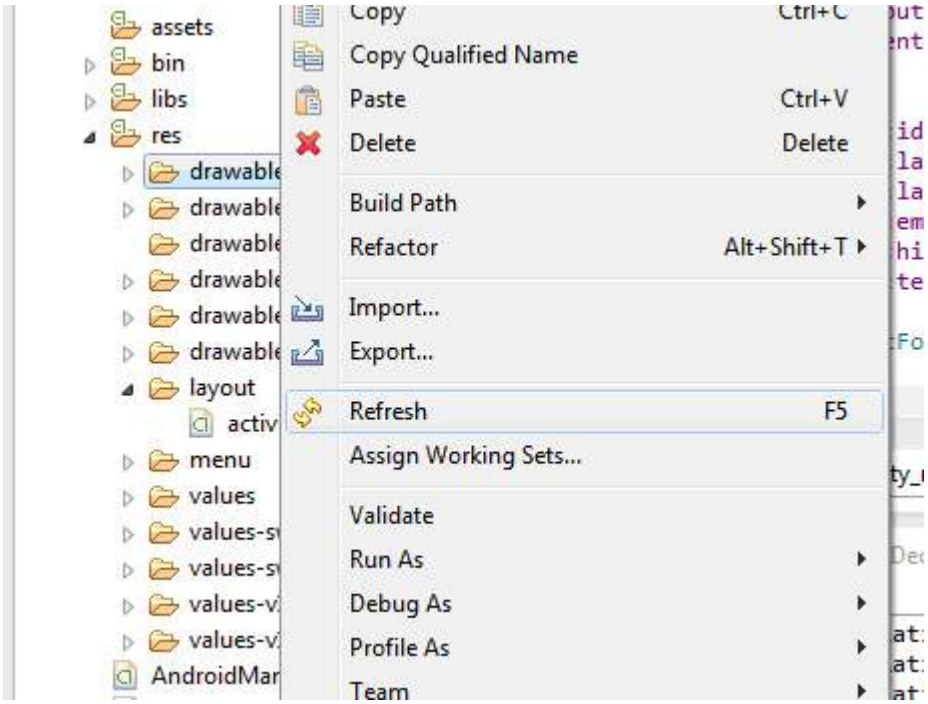


Рис. 6.37. Обновление папки

Теперь поместим внутрь "рамки" <ImageView>

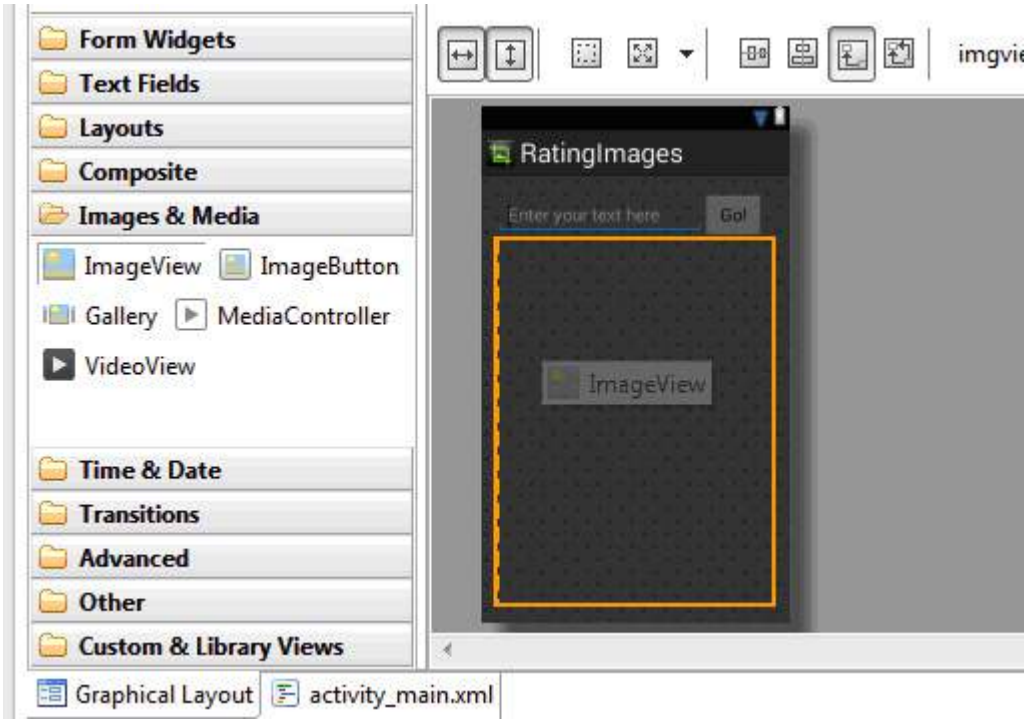


Рис. 6.38. Перемещение <ImageView>

В появившемся окне выбираем нужное нам изображение и жмём **OK**

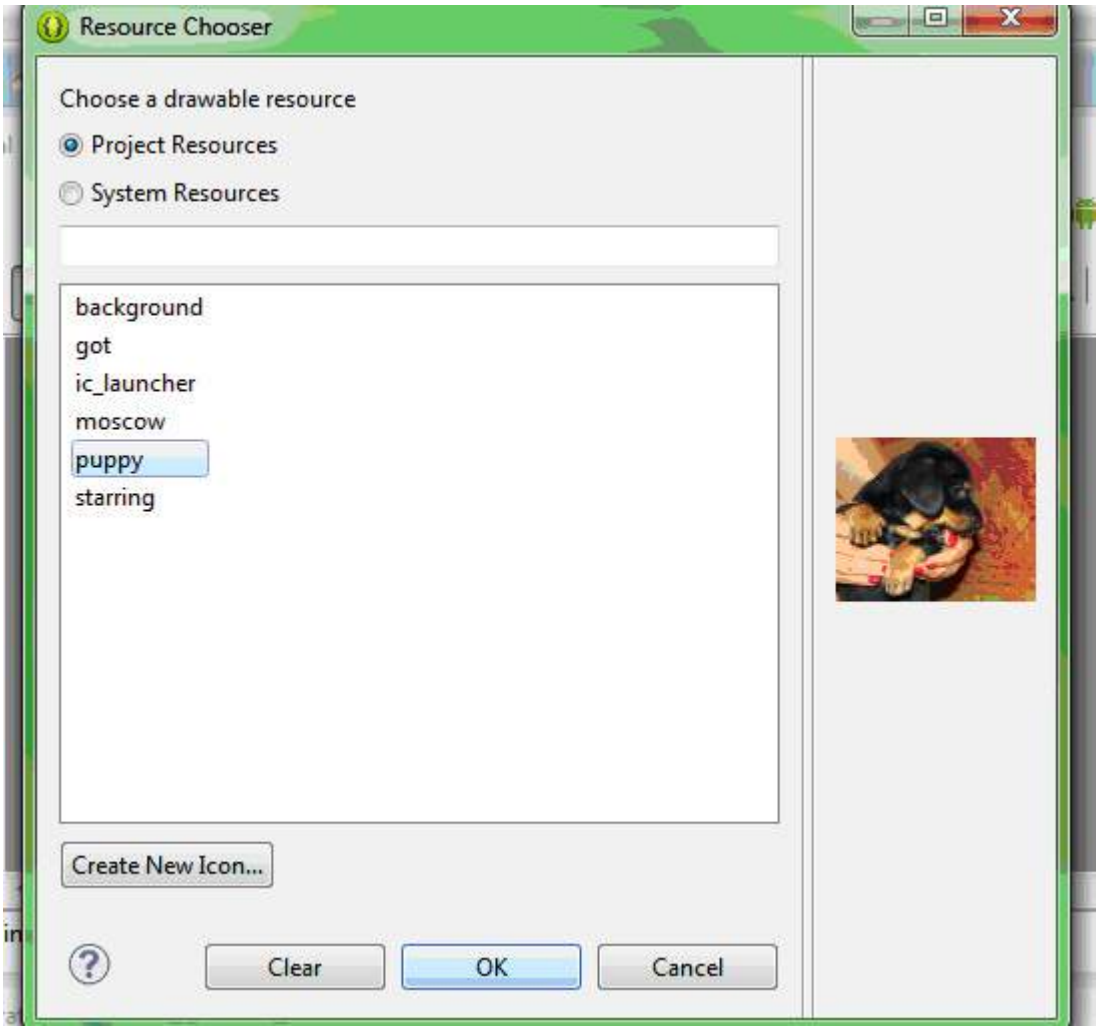


Рис. 6.39. Выбор файла

Изображение должно появится на активности:

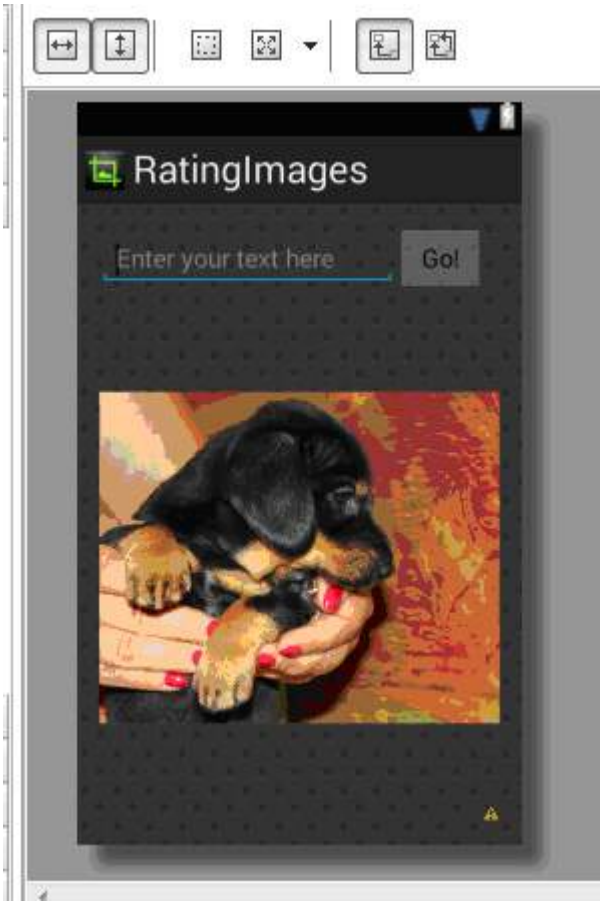


Рис. 6.40. Визуализация изображения

Вместе с изображением появился новый предупреждающий знак.

В описании предупреждения сказано: "**[Accessibility] Missing contentDescription attribute on image**". Давайте разберёмся, что это означает.

Атрибут android:contentDescription используется в программах, которые поддерживают специальные возможности для людей с нарушениями зрения и слуха. То есть, текст, прописанный в этом атрибуте, не будет показан на экране, но при включенной в "Специальных возможностях" услуге "звуковые подсказки" этот текст будет проговариваться, когда пользователь переходит к этому элементу управления.

Этот атрибут можно задать для ImageButton, ImageView и CheckBox, и задавать его или нет - дело каждого.

Свойства у изображения должны быть следующими:

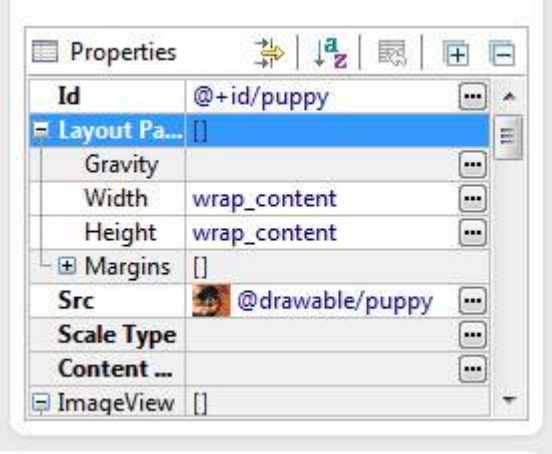


Рис. 6.41. Свойства изображения

6.2.6 Кнопки "like" и "dislike"

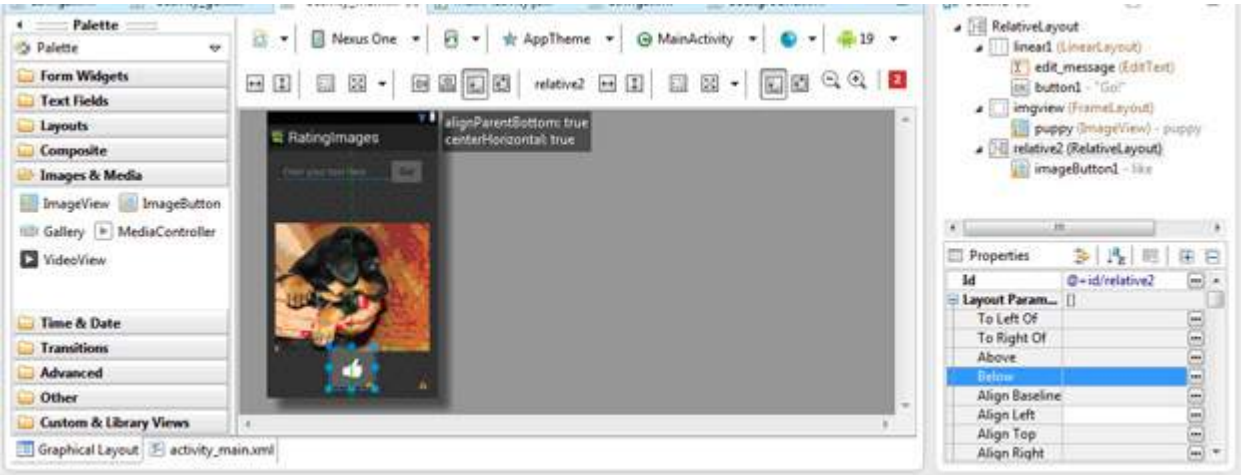
Пришло время создать кнопки оценивания.

Для этого добавьте на форму <RelativeLayout>, задайте для него ширину и высоту wrap_content, и укажите id.

Снова в папку **res/drawable/** нужно добавить файлы. Найдите изображения "Палец вверх" и "Палец вниз", и поместите их в эту папку, после чего обновите её.

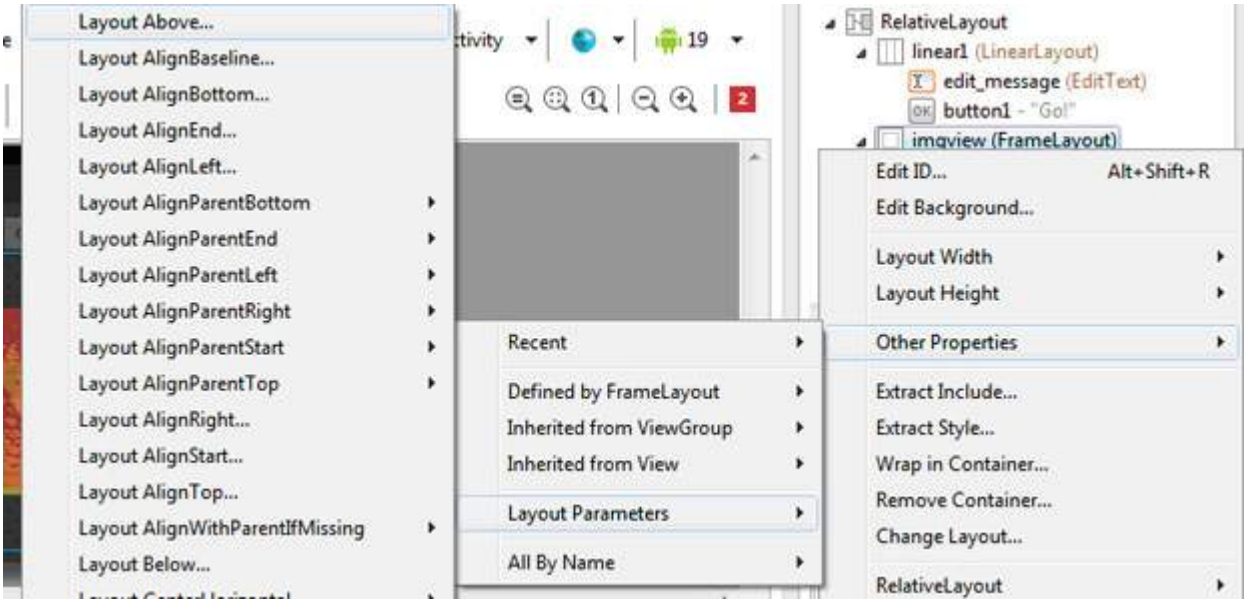
Изображения и другие полезные файлы можно скачать здесь.

Добавьте <ImageButton>, выберите изображение "Палец вверх", и переместите <RelativeLayout> в такое положение:

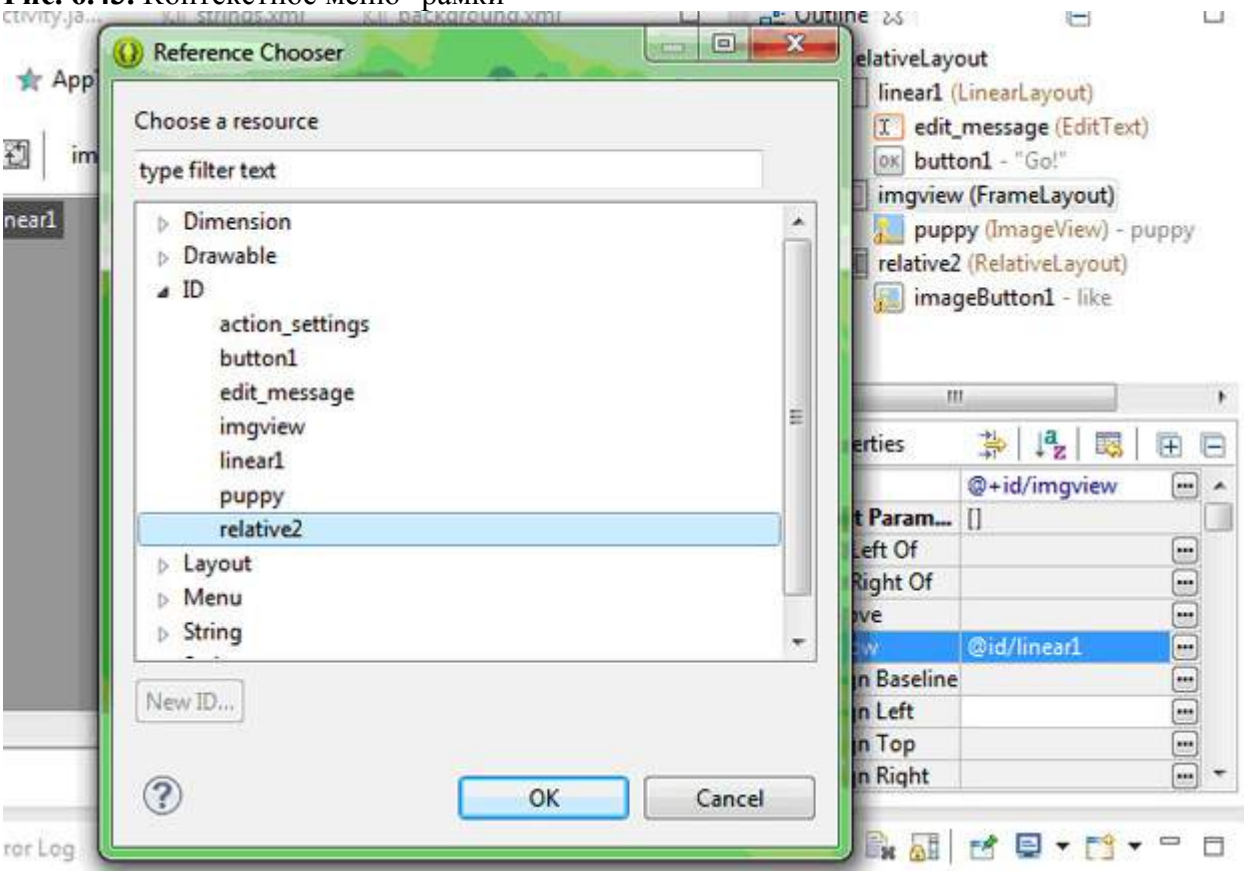


увеличить изображение
Рис. 6.42. Новая кнопка

Укажите "рамке" быть выше, чем поле с кнопкой оценки:

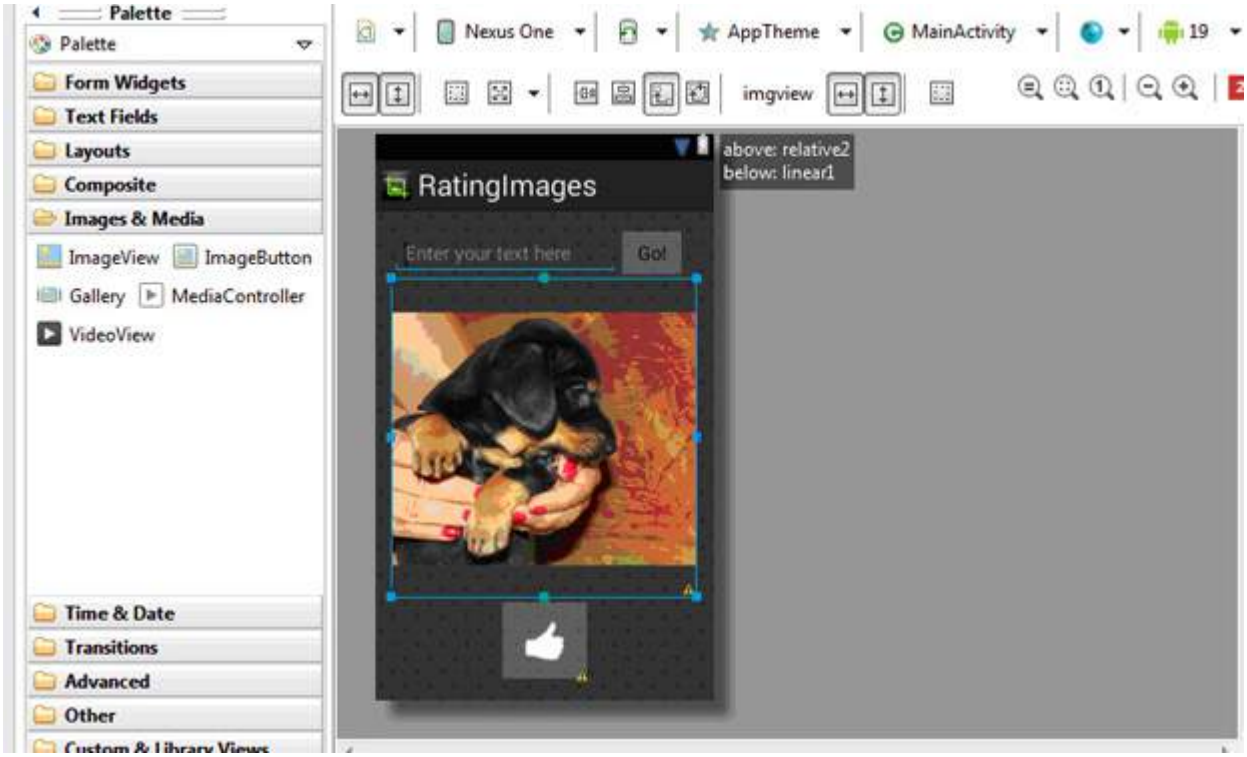


[увеличить изображение]
Рис. 6.43. Контекстное меню "рамки"



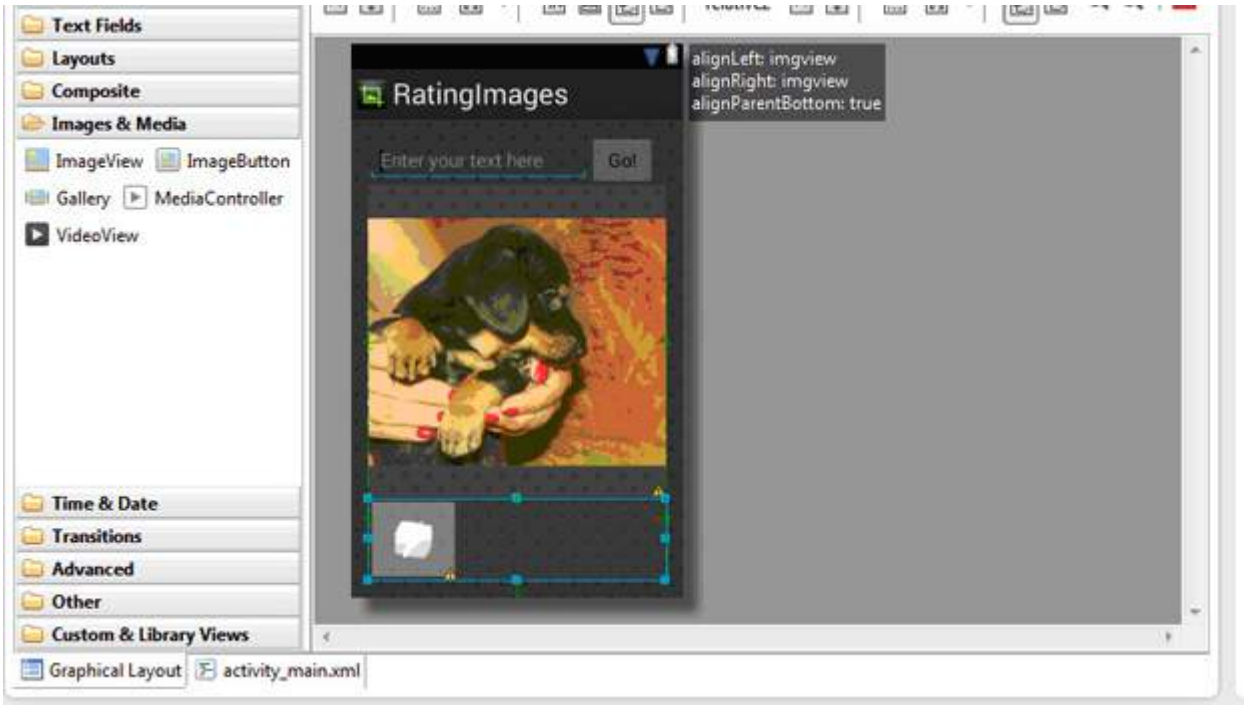
[увеличить изображение]
Рис. 6.44. Выбор элемента, выше которого должна быть "рамка"

В результате должно получиться следующее:



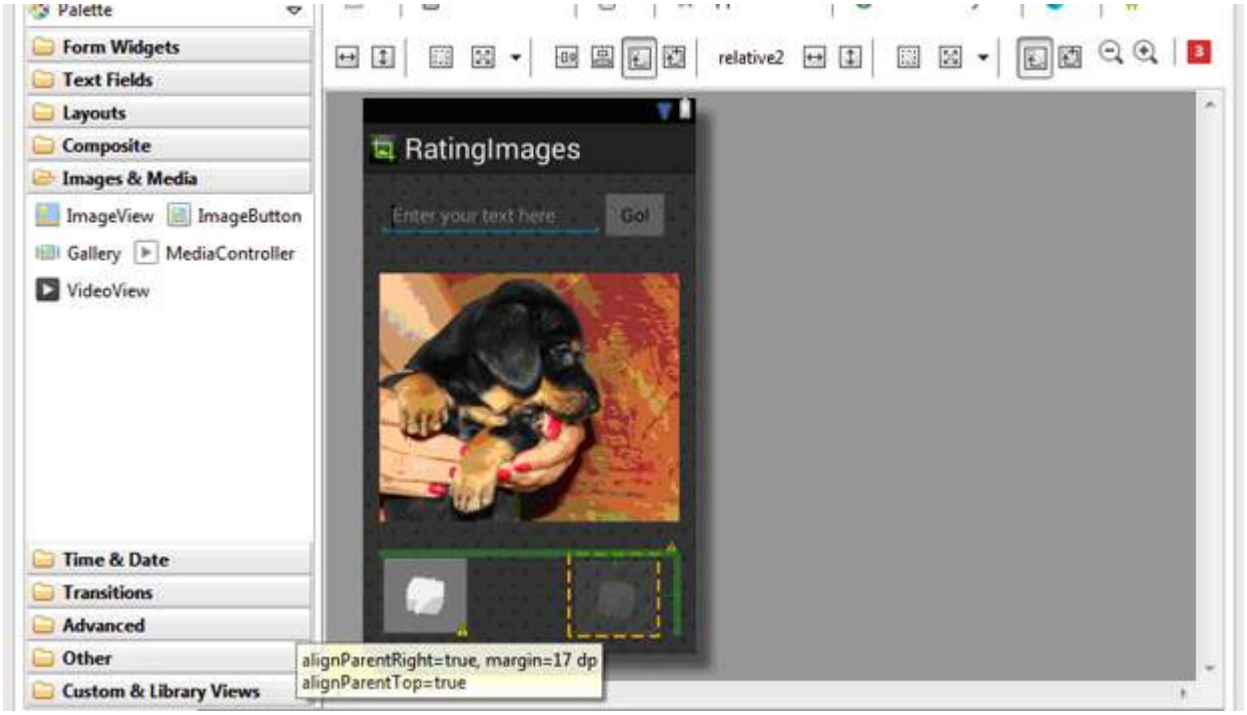
[увеличить изображение]
Рис. 6.45. Новое расположение элементов

Добавьте еще одну кнопку - кнопку "**Палец вниз**". Она "наложилась" на первую кнопку. Чтобы это исправить, проделайте с <RelativeLayout> то же самое, что и с <LinearLayout>: растяните элемент влево и вправо, до получения такого результата:



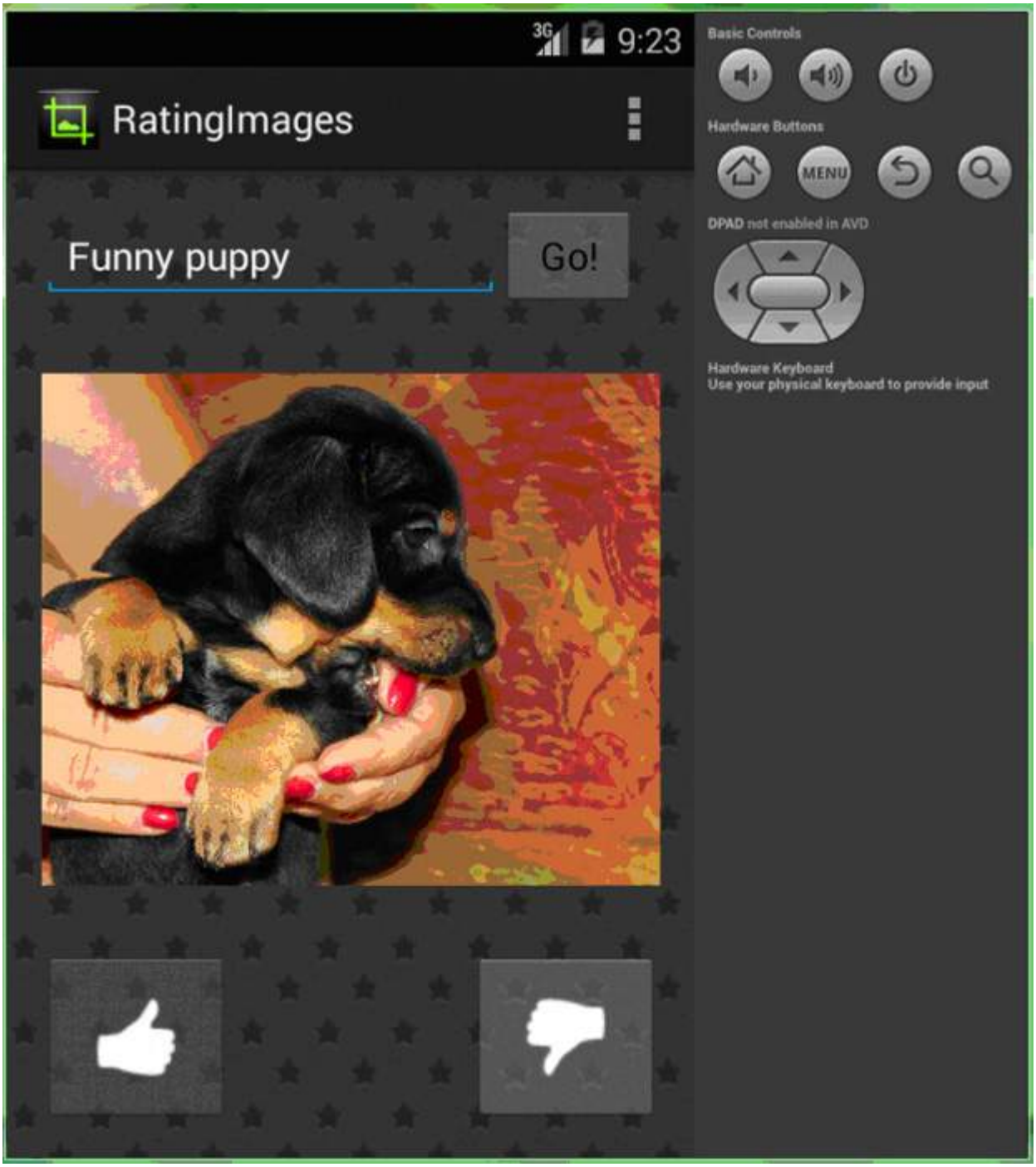
[увеличить изображение]
Рис. 6.46. Выравнивание области с кнопками

Теперь расставьте кнопки по краям так, чтобы они "прикрепились" к краям.



[увеличить изображение]
Рис. 6.47. Выравнивание кнопок

Готово! Теперь можно запустить эмулятор и посмотреть, что получилось.



увеличить изображение
Рис. 6.48. Главная активность

6.2.7 Листинги

1)

```
<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=".MainActivity"
        android:background="@drawable/background">
```

```
        <LinearLayout
                android:id="@+id/linear1"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_alignParentLeft="true"
                android:layout_alignParentRight="true"
                android:orientation="horizontal" >
                <EditText
                        android:id="@+id/edit_message"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:ems="10"
                        android:hint="@string/edit_message"
                        android:textColor="#ffffff" >
                        <requestFocus />
                </EditText>

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

        <FrameLayout
                android:id="@+id/imgview"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:layout_above="@+id/relative2"
                android:layout_below="@id/linear1" >
                <ImageView
                        android:id="@+id/puppy"
                        android:layout_width="wrap_content"
                        android:layout_height="match_parent"
                        android:src="@drawable/puppy"
                />
        </FrameLayout>

        <RelativeLayout
                android:id="@+id/relative2"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_alignLeft="@+id/imgview"
                android:layout_alignParentBottom="true"
                android:layout_alignRight="@+id/imgview" >
                <ImageButton
                        android:id="@+id/imageButton2"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:layout_alignParentRight="true"
                        android:layout_alignParentTop="true"
                        android:src="@drawable/dislike"
                />
                <ImageButton
                        android:id="@+id/imageButton1"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:layout_alignParentLeft="true"
                        android:layout_alignParentTop="true"
                        android:src="@drawable/like"
                />
        </RelativeLayout>

</RelativeLayout>
```

2)

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

                <string name="app_name">RatingImages</string>
                <string name="action_settings">Settings</string>
                <string name="hello_world">Hello world!</string>
                <string name="edit_message">Enter your text here</string>
                <string name="button_send">Go!</string>

</resources>
```

3)

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

<bitmap
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:src="@drawable/starring"
        android:tileMode="repeat" >

</bitmap>
```

6.3 BuildingBlocks или элементы для построения интерфейса

Среда разработки "**AndroidIDE**" открывает широкий выбор готовых к использованию элементов для создания выдающихся приложений.

Кнопка (Button) состоит из текста и/или изображения, которые дают понять пользователю, что произойдёт, если нажать на эту кнопку. Важно помнить, что человек по своей природе привык взаимодействовать с объектами, поэтому фон у кнопки абсолютно не обязателен.

Рис. 6.49a. Кнопка-изображение

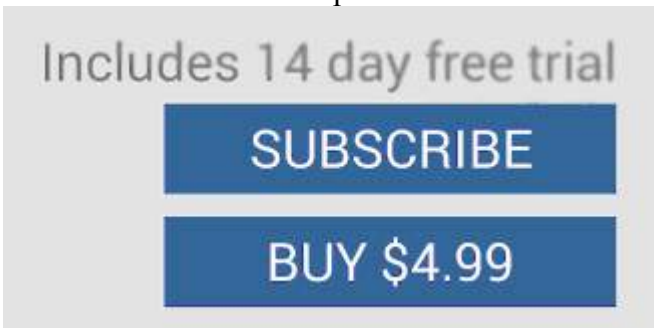
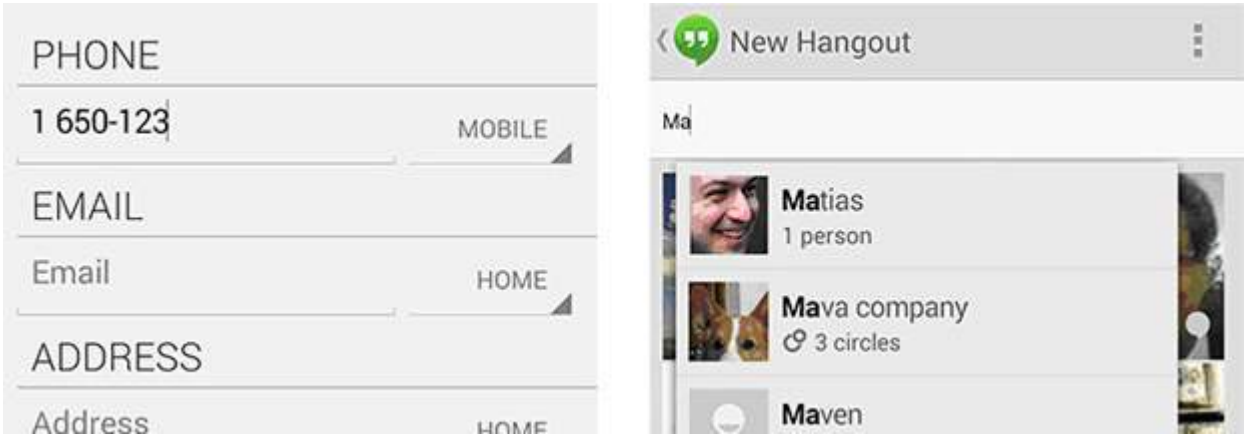


Рис. 6.49b. Кнопка-текст

Текстовые поля (TextFields)

Текстовые поля позволяют пользователю вводить текст в приложения. Они могут быть однострочными или многострочными. Одно касание текстового поля помещает курсор на поле ввода и автоматически выводит на экран клавиатуру. В дополнение к набору текста на клавиатуре текстовые поля позволяют выделять текст (вырезать, копировать, вставить). Поиск вариантов завершения слова помогает правописанию слов и упрощает поиск контактов в списке.



увеличить изображение
Рис. 6.50. 50 Текстовые поля, поиск контактов

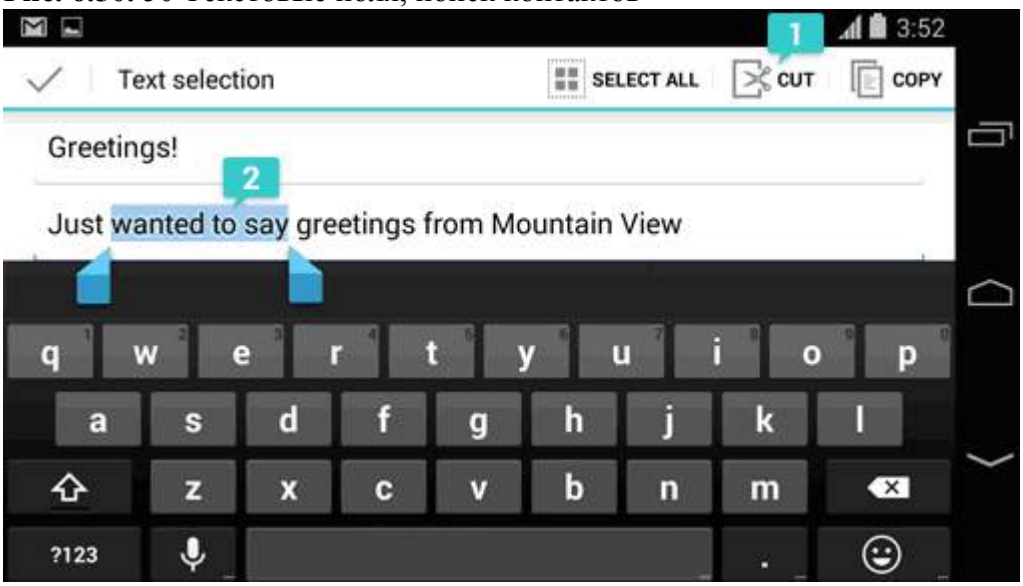


Рис. 6.51. Выделение текста

Ползунки и слайдеры (Seek Bars and Sliders)

Интерактивные ползунки позволяют выбирать значение из непрерывного или дискретного диапазона значений путем перемещения ползунка. Наименьшее значение находится слева, наибольшее справа.

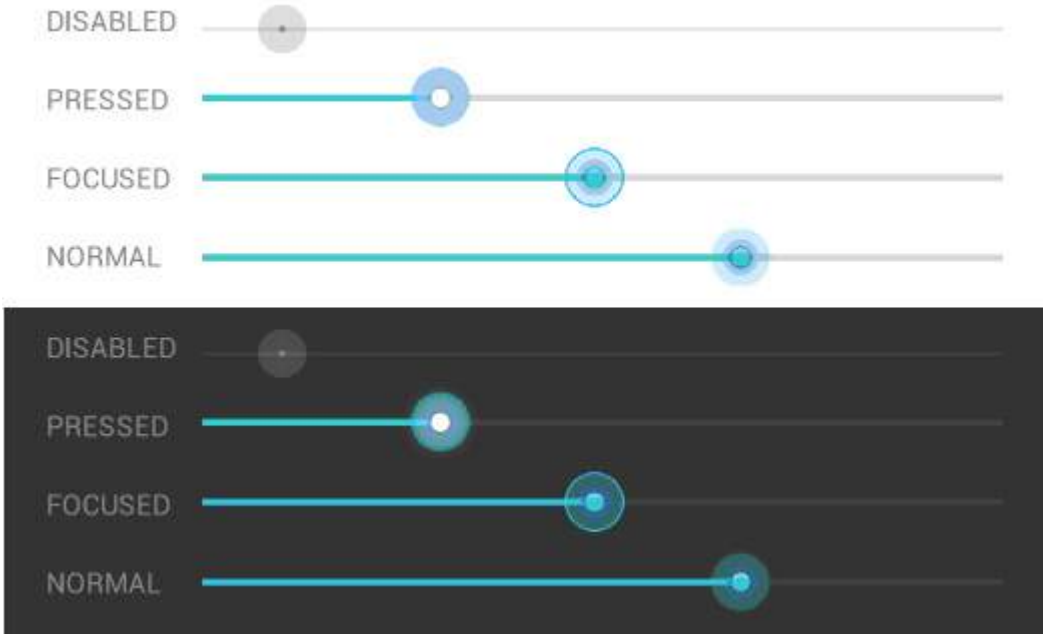


Рис. 6.52. Ползунки в светлой и темной темах

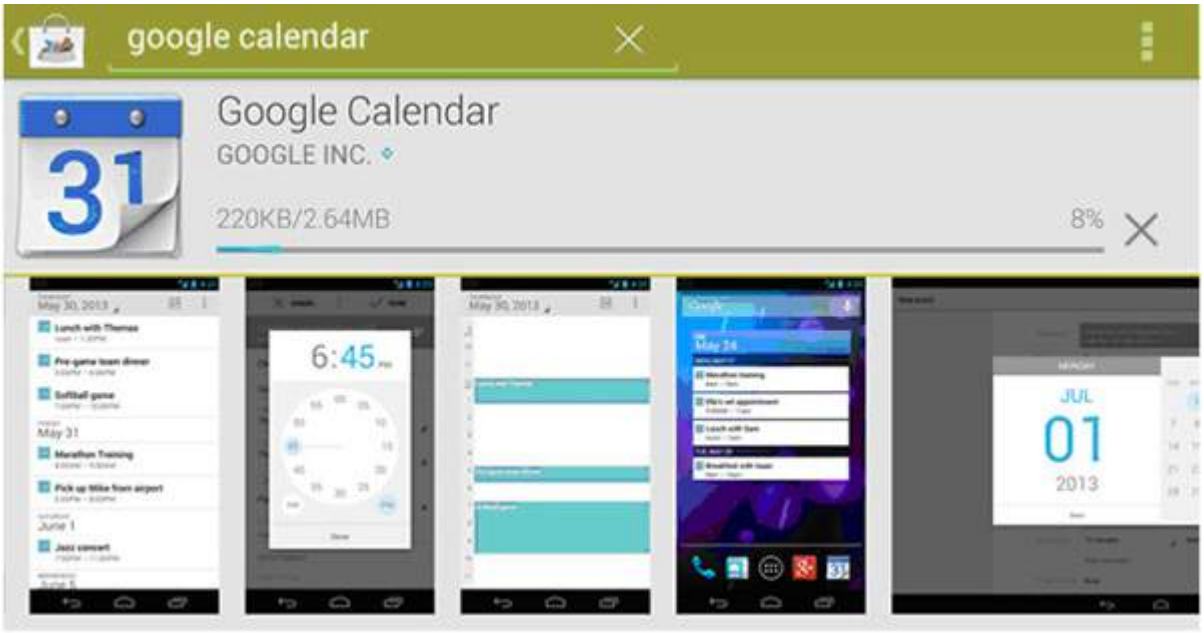
Интерактивный характер слайдера делает его удобным для настроек, которые отражают уровни интенсивности, такие как громкость, яркость, или насыщенность цвета.

Так, например, уровень громкости можно регулировать кнопками на корпусе устройства или при помощи жеста пальцем по экрану.

Рис. 6.53. Настройка звука

Прогресс-бары и активности (Progress&Activity)

Прогресс-бары и показатели активности сигнализируют пользователям о происходящем в данный момент времени длительном действии, что означает для пользователя подождать завершения процесса некоторое время.



увеличить изображение
Рис. 6.54. Процесс загрузки приложения

Переключатели (Switches) позволяют пользователю выбирать параметры.

Есть три вида переключателей: флажки, радио-кнопки, и включение/выключение выключателей.

Флажки используются в том случае, если из предлагаемого списка можно выбрать одновременно несколько вариантов.

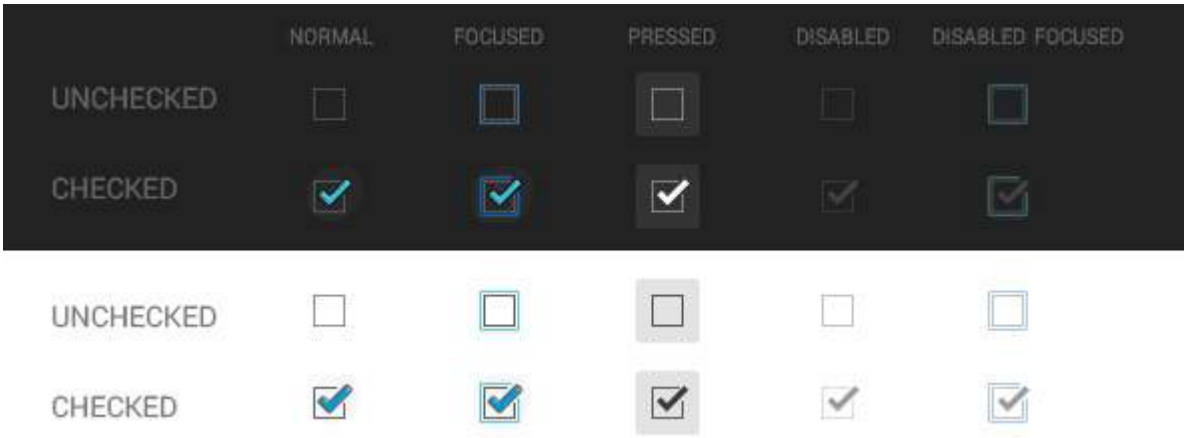


Рис. 6.55. Флажки

Радио-кнопки позволяют выбрать только один вариант из списка. Радио-кнопки формируются в группы.

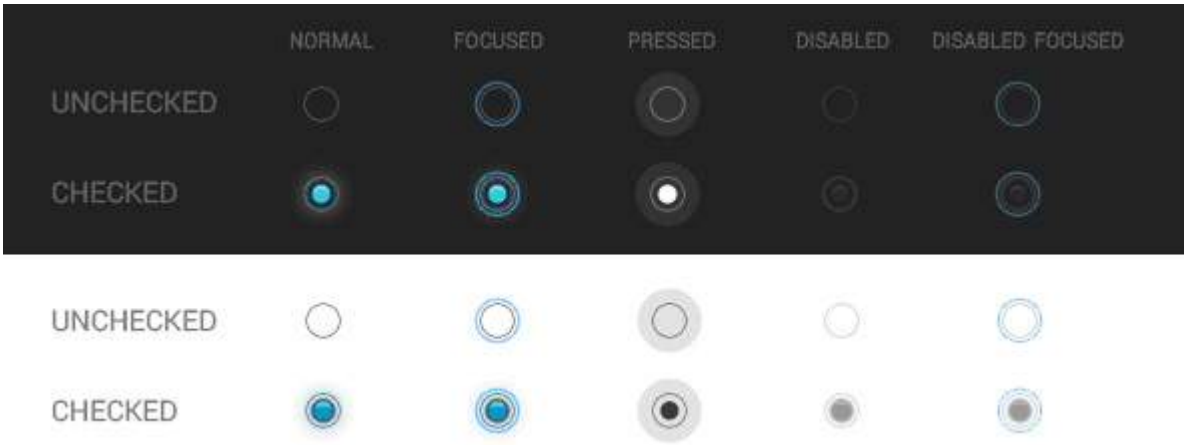


Рис. 6.56. Радио-кнопки

Выключатели дают возможность сделать флажок более наглядным, применив в качестве основы кнопку-значок, которая может фиксироваться в нажатом состоянии.

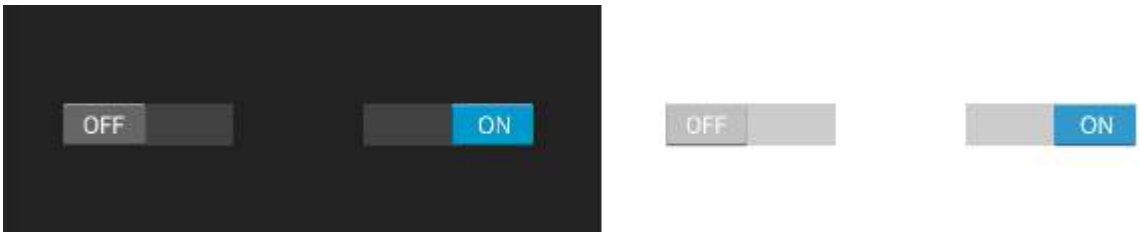
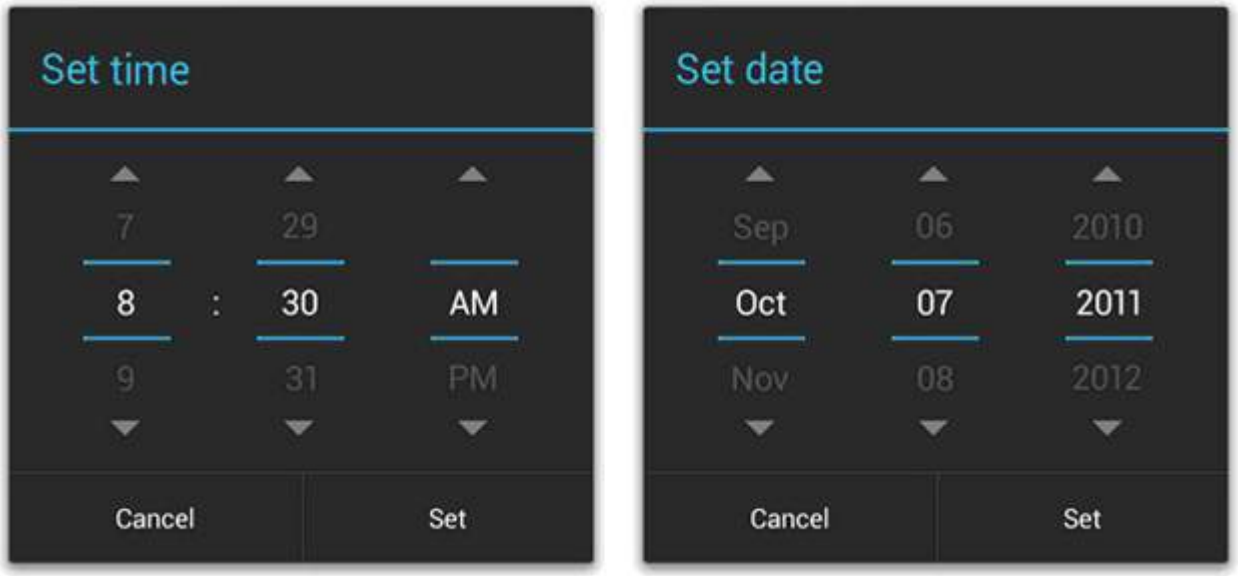


Рис. 6.57. Выключатели

Диалоговые окна (Dialogs) помогают приложению взаимодействовать с пользователем. Это может быть как банальный вопрос с двумя вариантами ответа (OK и Отмена), так и сложные окна с множеством настраиваемых пользователем параметров.

увеличить изображение
Рис. 6.58. Примеры диалоговых окон

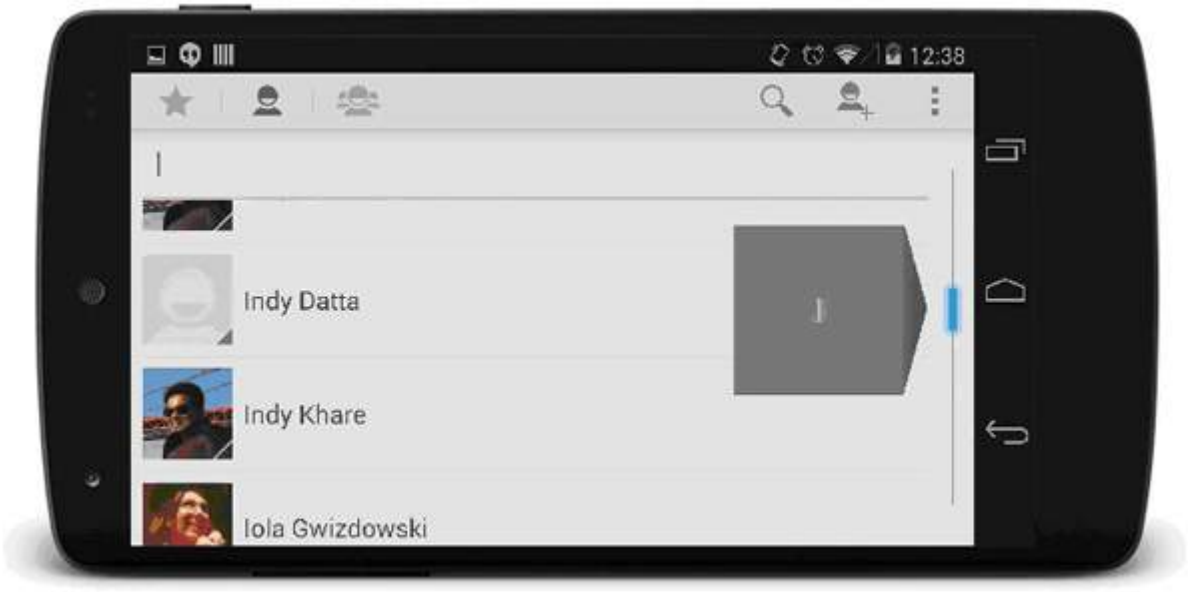
Средство выбора (Pickers) - это простой способ выбрать одно значение из набора путём последовательного перебора. Удобный перебор предлагаемых значений при помощи клика по стрелочкам или прокрутки, так же можно ввести значение с клавиатуры.



увеличить изображение
Рис. 6.59. Примеры пикеров

Прокрутка (Scrolling), как интуитивно ожидаемый элемент, уже давно плотно сидит в головах у разработчиков и пользователей. Плавное или быстрое перемещение по содержимому экрана и контенту, который "вылез" за рамки экрана, осуществляется прокруткой, достаточно лишь провести пальцем по экрану в нужную сторону с желаемой скоростью.

Для того чтобы индикатор прокрутки не занимал место у рабочей области экрана, после приостановки пользования прокруткой, индикатор исчезает, и появляется снова, стоит только начать "прокручивать" пальцем. Для удобства поиска чего-либо в алфавитном списке, рядом с индикатором прокрутки возникает индекс с указанием на то, в каком разделе находится индикатор.



увеличить изображение
Рис. 6.60. Индикатор и индекс прокрутки

Выпадающий список (Spinner) обеспечивает удобный способ выбора одного значения из набора.

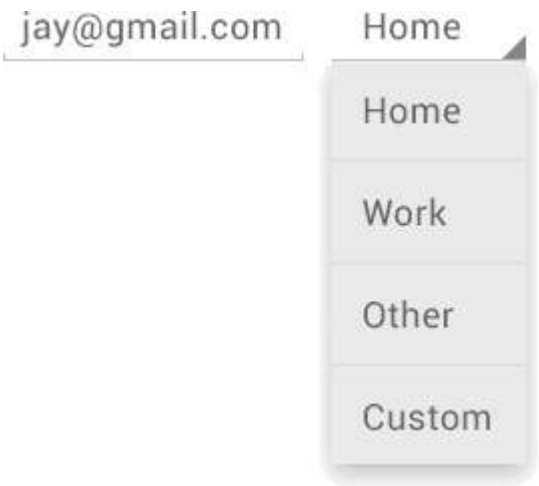


Рис. 6.61. Спиннер для выбора типа e-mail адреса

Вкладки (Tabs) позволяют легко просматривать и переключаться между различными окнами или функциональными элементами приложения. Вкладки, прокручиваемые движением пальца (ScrollableTabs), делают этот процесс еще прощё и приятнее.

Основные вкладки (FixedTabs) удобны при отображении трех и менее вкладок, поскольку ширина вкладок фиксируется по самому длинному названию вкладки.

увеличить изображение
Рис. 6.62. Вкладки

Списки (Lists) удобны для представления нескольких позиций информации в вертикальном расположении.

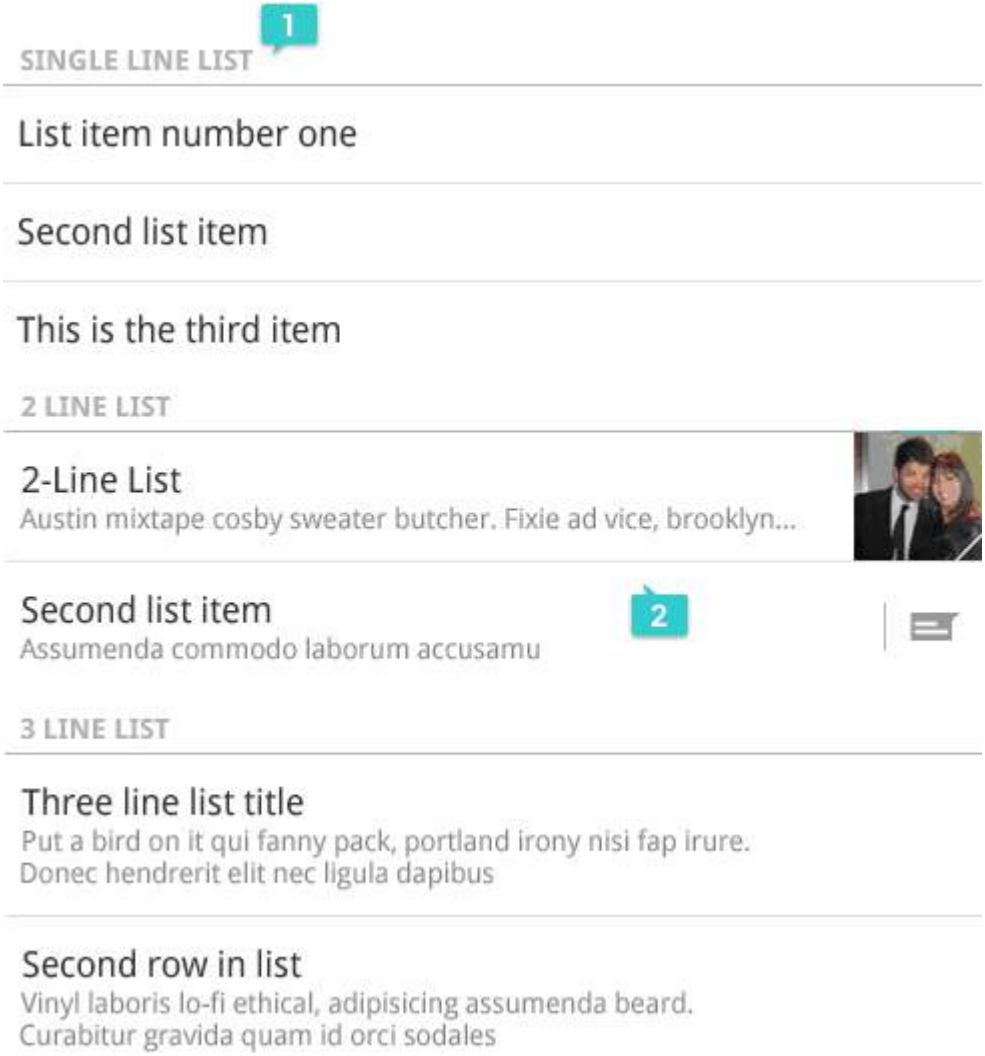


Рис. 6.63. Список

Сетки (GridLists) - это альтернатива стандартного списка. С помощью сетки можно создавать более удобное представление образных данных, нежели с помощью списка. Ещё одним плюсом сетки является то, что прокрутку можно осуществлять не только в вертикальном, но и в горизонтальном направлении.

Однако нужно стараться избегать создания сетки с возможностью прокрутки в обоих измерениях сразу.

Использование ярлычков для всех панелей сетки и полупрозрачных "занавесок" для неактивных панелей помогает пользователю быстрее понять, что скрывается под той или иной панелью.

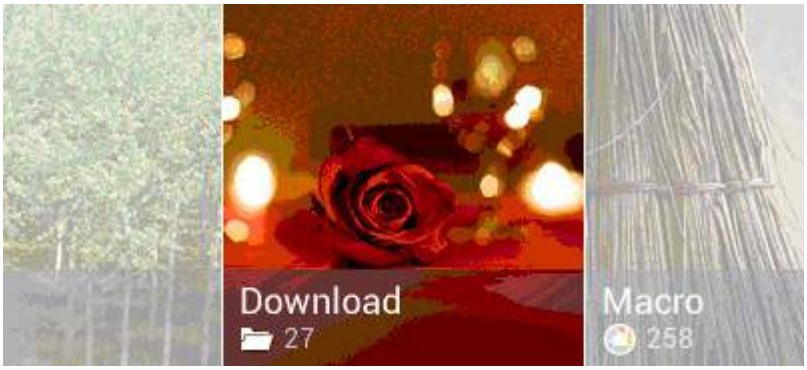


Рис. 6.64. Сетка с альбомами

Задание

1. Создайте в приложении на главной активности строку, в которой будет выводиться адрес сайта, откуда загружено изображение, и кнопку для перехода на этот сайт.
2. Подумайте над интерфейсом собственного приложения. Какие элементы управления оно может содержать? Попробуйте воплотить его интерфейс на практике.

Внимание! Если Вы увидите ошибку на нашем сайте, выделите её и нажмите Ctrl+Enter.
© Национальный Открытый Университет "ИНТУИТ", 2016 | www.intuit.ru

Самостоятельная работа 5. Демонстрации распознавания стандартных жестов

Цель лабораторной работы:

разработать простейшие приложения для демонстрации распознавания стандартных жестов.

Задачи лабораторной работы:

- рассмотреть распознавание всех поддерживаемых жестов;
- рассмотреть распознавание только части поддерживаемых жестов.

10.1 Введение

Для работы со стандартными жестами Android предоставляет класс GestureDetector. Этот класс содержит два вложенных интерфейса-слушателя: OnGestureListener и OnDoubleTapListener, эти интерфейсы задают методы, отслеживающие стандартные жесты. А также GestureDetector содержит вложенный класс SimpleOnGestureListener, который содержит пустые реализации, возвращающие значение false, где это необходимо, всех методов интерфейсов: OnGestureListener и OnDoubleTapListener.

В лабораторной работе рассмотрим две возможности распознавания жестов:

- случай распознавания всех поддерживаемых жестов, для этого реализуем в классе активности оба интерфейса;
- случай распознавания только некоторого набора поддерживаемых жестов, для этого в классе активности объявим внутренний класс-наследник класса GestureDetector.SimpleOnGestureListener.

10.2 Распознавание всех поддерживаемых жестов

Разработаем приложение, в котором продемонстрируем распознавание всех поддерживаемых жестов. Приложение содержит одну активность, одно информационное поле для вывода информации о распознанном жесте. Приложение работает следующим образом: пользователь выполняет один из поддерживаемых сенсорных жестов, в информационном поле отображается информация о распознанном жесте.

1. Создадим простое приложение и добавим на форму TextView для вывода информации.
2. Настроим логику приложения. В java класс, соответствующий активности внесем следующие дополнения.
    - Класс активности должен реализовывать интерфейсы: GestureDetector.OnGestureListener и GestureDetector.OnDoubleTapListener, для этого в объявление класса добавим конструкцию:
    - `implements GestureDetector.OnGestureListener,`
    - `GestureDetector.OnDoubleTapListener`

    - Нам понадобится экземпляр класса GestureDetectorCompat поэтому в качестве поля класса активности объявим следующую переменную:
    - `GestureDetectorCompat mDetector;`

В методе onCreate() класса активности, создадим экземпляр класса GestureDetectorCompat и присвоим его переменной mDetector:

```
mDetector = new GestureDetectorCompat(this,this);
```

одним из параметров конструктора является класс, который реализует интерфейс GestureDetector.OnGestureListener, в нашем случае использовано слово this, т. е. параметром является сам класс активности. Этот интерфейс уведомляет пользователей когда появляется определенное сенсорное событие.

В методе OnCreate() класса активности, следующая строка:

```
mDetector.setOnDoubleTapListener(this);
```

устанавливает слушатель событий, связанных с двойным касанием, это должен быть класс, реализующий интерфейс GestureDetector.OnDoubleTapListener. В нашем случае использовано слово this, т.е. слушателем будет опять сам класс активности.

1. Чтобы позволить вашему объекту GestureDetector получать события, необходимо переопределить метод onTouchEvent() для активности или элемента GUI. И передавать в экземпляр детектора все обнаруженные события.

```
2. public boolean onTouchEvent(MotionEvent event){
3.     this.mDetector.onTouchEvent(event);
4.     // Be sure to call the superclass implementation
5.     return super.onTouchEvent(event);
6. }
```

7. После проведенной подготовки пришло время реализовать все методы, объявленные в интерфейсах, отвечающих за прослушивание сенсорных событий.

Методы интерфейса GestureDetector.OnGestureListener:

onDown()	- отслеживает появление касания, т. е. палец прижат к экрану;
onFling()	- отслеживает появление жеста смахивания;
onLongPress()	- отслеживает удерживание пальца прижатым к экрану длительное время;
onScroll()	- отслеживает появление жеста прокрутки (пролистывания);
onShowPress()	- отслеживает, что произошло событие касания и больше никаких событий не происходит короткое время;
onSingleTapUp()	- отслеживает появление жеста одиночного нажатия (клик).

Методы интерфейса GestureDetector.OnDoubleTapListener:

onDoubleTap()	- отслеживает появление жеста двойного нажатия ("двойной клик");
onDoubleTapEvent()	- отслеживает появление события во время выполнения жеста двойного нажатия, включая касание, перемещение, подъем пальца.
onSingleTapConfirmed()	- отслеживает появление жеста одиночного нажатия (клик).

В листинге 10.1 представлен код приложения, в котором распознаются все поддерживаемые жесты, информация о появившемся и распознанном жесте выдается в информационное поле (TextView).

В качестве практики предлагается воспроизвести данное приложение и проверить, как система распознает тот или иной жест. Очень полезно для понимания, как выполняются основные жесты.

10.3 Распознавание только части поддерживаемых жестов

Разработаем приложение, в котором продемонстрируем распознавание только некоторой части поддерживаемых жестов по выбору программиста. Мы рассмотрим распознавание жеста смахивания (fling). Приложение содержит одну активность, одно информационное поле для вывода информации о распознанном жесте. Приложение работает следующим образом: пользователь выполняет один из поддерживаемых сенсорных жестов, в информационном поле отображается информация о распознанном жесте.

1. Создадим простое приложение и добавим на форму TextView для вывода информации.
2. Настроим логику приложения. В java класс, соответствующий активности внесем следующие дополнения.

Нам понадобится экземпляр класса GestureDetectorCompat поэтому в качестве поля класса активности объявим следующую переменную:

```
GestureDetectorCompat mDetector;
```

В методе onCreate() класса активности, создадим экземпляр класса GestureDetectorCompat и присвоим его переменной mDetector:

```
mDetector=new GestureDetectorCompat(this, new MyGestListener());
```

в конструкторе аргументом, отвечающим за отслеживание сенсорных событий, служит экземпляр класса MyGestListener() - внутренний класс, который является наследником класса GestureDetector.SimpleOnGestureListener.

Имеет смысл немного рассмотреть класс GestureDetector.SimpleOnGestureListener. Этот класс реализует интерфейсы GestureDetector.OnGestureListener и GestureDetector.OnDoubleTapListener, все методы заявленные в интерфейсах в этом классе имеют пустую реализацию и те, которые должны возвращать значение, возвращают false. Поэтому для распознавания какого-то события или некоторого подмножества событий достаточно написать реализацию соответствующих методов, в классе наследнике.

В листинге 10.2 представлен код приложения, в котором распознается только жест смахивания, т. е. реализован метод onFling(), информация о появившемся и распознанном жесте выдается в информационное поле (TextView). В качестве слушателя используется экземпляр класса MyGestListener(), являющийся наследником класса GestureDetector.SimpleOnGestureListener.

Получить больше информации о распознавании жестов можно по ссылке:
http://developer.android.com/training/gestures/index.html

```
package com.example.lab5_1_gestall;

import android.os.Bundle;
import android.app.Activity;
import android.support.v4.view.GestureDetectorCompat;
import android.view.*;
import android.widget.*;

public class MainActivity extends Activity
       implements GestureDetector.OnGestureListener,
                  GestureDetector.OnDoubleTapListener
{
  TextView tvOutput;
  GestureDetectorCompat mDetector;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
```

```
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);

  tvOutput = (TextView)findViewById(R.id.textView1);

  mDetector = new GestureDetectorCompat(this,this);
  mDetector.setOnDoubleTapListener(this);
}

public boolean onTouchEvent(MotionEvent event){
  this.mDetector.onTouchEvent(event);
  // Be sure to call the superclass implementation
  return super.onTouchEvent(event);
    }

@Override
public boolean onDown(MotionEvent event) {
    tvOutput.setText("onDown: " + event.toString());
    return false;
}

@Override
public boolean onFling(MotionEvent event1, MotionEvent event2,
               float velocityX, float velocityY) {
  tvOutput.setText("onFling: " + event1.toString()+event2.toString());

    return true;
}

@Override
public void onLongPress(MotionEvent event) {
   tvOutput.setText("onLongPress: " + event.toString());
}

@Override
public boolean onScroll(MotionEvent e1, MotionEvent e2, float distanceX,
                  float distanceY) {
   tvOutput.setText("onScroll: " + e1.toString()+e2.toString());
     return true;
}

@Override
public void onShowPress(MotionEvent event) {
   tvOutput.setText("onShowPress: " + event.toString());
}

@Override
public boolean onSingleTapUp(MotionEvent event) {
  tvOutput.setText("onSingleTapUp: " + event.toString());
    return true;
}

@Override
public boolean onDoubleTap(MotionEvent event) {
  tvOutput.setText("onDoubleTap: " + event.toString());
    return true;
}

@Override
public boolean onDoubleTapEvent(MotionEvent event) {
  tvOutput.setText("onDoubleTapEvent: " + event.toString());
    return true;
}

@Override
public boolean onSingleTapConfirmed(MotionEvent event) {
   tvOutput.setText("onSingleTapConfirmed: " + event.toString());
```

```
      return true;
  }
}
```

Листинг 10.1. Распознавание поддерживаемых жестов с помощью реализации интерфейсов

```
package com.example.lab5_1_gestsubset;

import android.os.Bundle;
import android.app.Activity;
import android.support.v4.view.*;
import android.view.*;
import android.widget.*;

public class SubsetGestActivity extends Activity {

private GestureDetectorCompat mDetector;
private TextView tvOut;

@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_subset_gest);
  mDetector = new GestureDetectorCompat(this, new MyGestListener());
  tvOut = (TextView)findViewById(R.id.textView1);
}

@Override
public boolean onTouchEvent(MotionEvent event){
this.mDetector.onTouchEvent(event);
return super.onTouchEvent(event);
}

class MyGestListener extends GestureDetector.SimpleOnGestureListener {

@Override
public boolean onFling(MotionEvent event1, MotionEvent event2,
                  float velocityX, float velocityY) {
tvOut.setText("onFling: " + event1.toString()+event2.toString());
   return true;
  }
}

}
```

Листинг 10.2. Распознавание жестов с использованием класса GestureDetector.SimpleOnGestureListener.

Внимание! Если Вы увидите ошибку на нашем сайте, выделите её и нажмите Ctrl+Enter.
© Национальный Открытый Университет "ИНТУИТ", 2016 | www.intuit.ru

Самостоятельная работа 6. Принципы работы с жестами вводимыми пользователями

Цель лабораторной работы:

Разработка приложения, помогающего понять принципы работы с жестами вводимыми пользователями.

Задачи лабораторной работы:

- создать набор жестов
- использовать созданные жесты в приложении

11.1 Введение

Начиная с версии 1.6, Android предоставляет API для работы с жестами, который располагается в пакете android.gesture и позволяет сохранять, загружать, создавать и распознавать жесты.

В данной работе рассмотрим процесс создания набора жестов, для создания жестов будем использовать приложение Gesture Builder, которое предустановлено на виртуальные устройства Android (AVD), начиная с версии 1.6.

После этого разработаем приложение, в котором предполагается распознавание и использование созданных жестов.

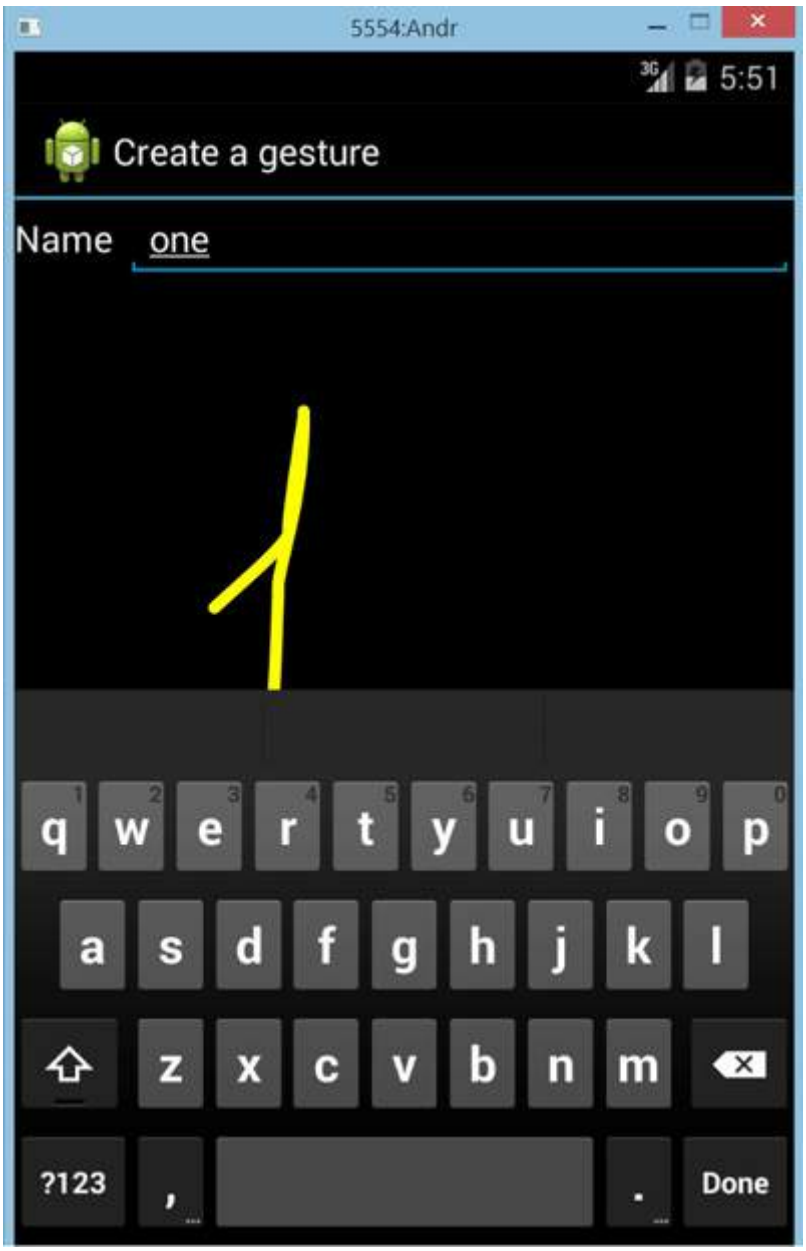
11.2 Создание набора жестов

Для начала создадим новое приложение.

Далее запустим эмулятор и используем приложение Gesture Builder для создания жестов "1", "2" и "S". Жест всегда связан с именем, но имя не обязательно должно быть уникальным, в действительности, для повышения точности в распознавании жеста рекомендуется сохранять несколько жестов с одним и тем же именем.

На рис. 11.1 можно увидеть приложение Gesture Builder в работе, чтобы добавить жест, необходимо нажать на кнопку **Add gesture**, в свободном пространстве изобразить жест (обычно он рисуется желтым цветом), в поле ввода задать имя жеста. Результат последовательного добавления жестов можно увидеть на рис. 11.2.

Жесты сохраняются на SD карте эмулятора, чтобы использовать их в приложении необходимо импортировать файл жестов в проект.



увеличить изображение

Рис. 11.1. Создание жестов с помощью приложения Gesture Builder

увеличить изображение
Рис. 11.2. Набор жестов, созданных в приложении Gesture Builder

Комментарий: Gesture Builder может сообщить, что ему некуда сохранять жесты, в этом случае необходимо запустить эмулятор с образом SD карты.

Сначала образ нужно создать с помощью утилиты **mksdcard** (расположена в папке **<AndrSDK>/sdk/tools**, где **<AndrSDK>** - путь, куда установлен Android SDK). В командной строке напишем:

```
mksdcard -l mySdCard 64M gesture.img
```

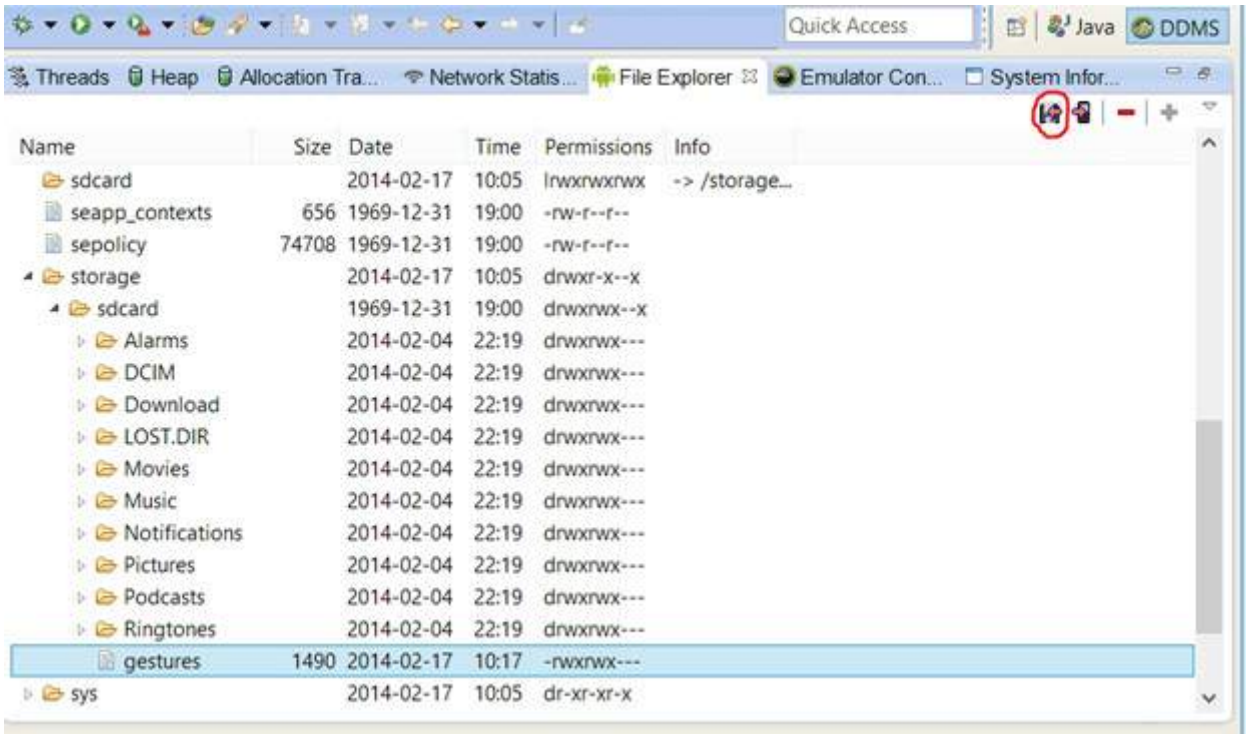
Следующим шагом будет создать и запустить эмулятор с образом gesture.img, для этого зайдем в папку **<AndrSDK>/sdk/tools** и выполним команду:

```
emulator -avd nameEmulator -sdcard gestures.img
```

nameEmulator - имя, которое присвоено эмулятору при создании.

Интересная статья на эту тему по ссылке: http://habrahabr.ru/post/120016/

Каждый раз, когда мы создаем или редактируем жесты с помощью Gesture Builder, создается файл gestures на SD карте эмулятора. Необходимо импортировать этот файл в директорию **res/raw/**, созданного проекта, в котором планируем использовать жесты.



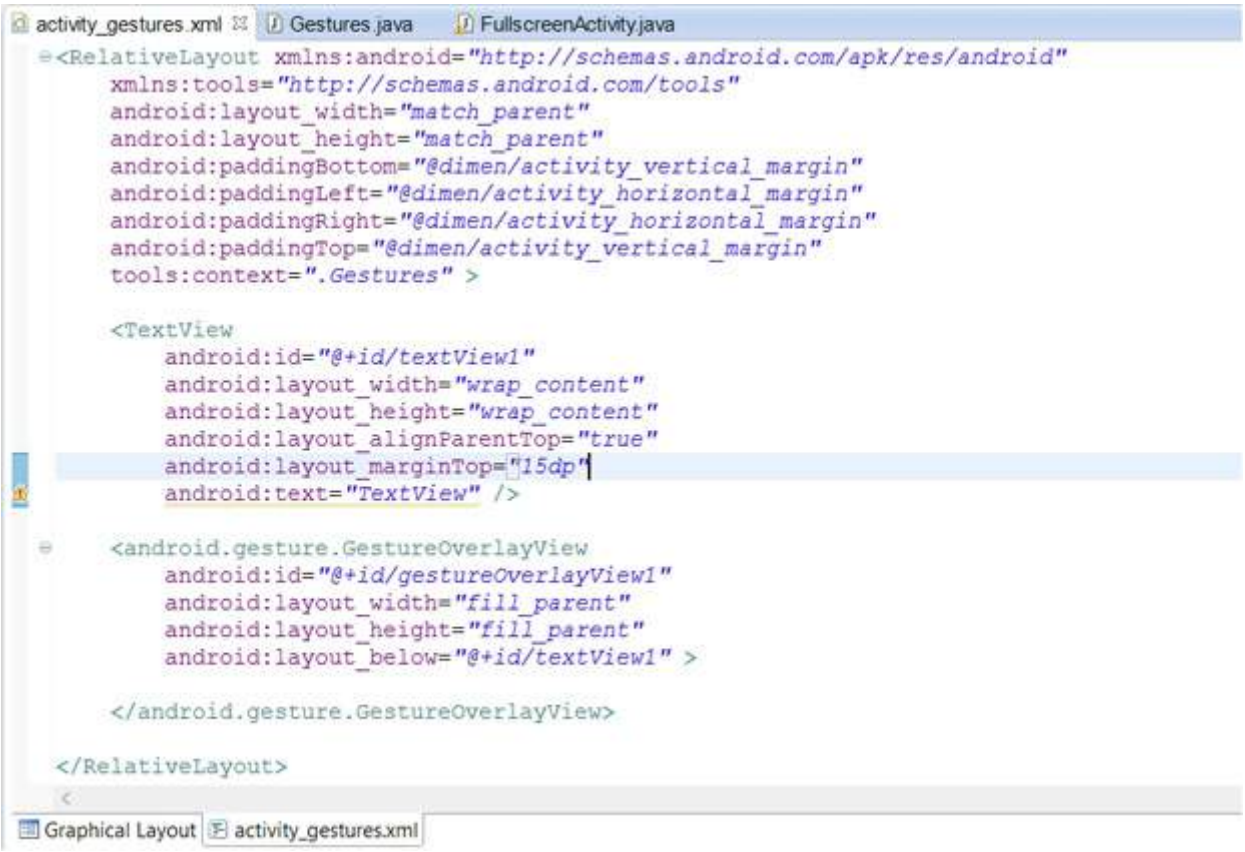
увеличить изображение
Рис. 11.3. Расположение файла gestures

Самый простой способ импортировать жесты в проект заключается использовании вкладки **File Explorer** в компоновке (perspective) DDMS. (Если компоновки DDMS нет найти ее можно следующим образом: **Window->Open Perspective->Other...->DDMS**. Если вкладки **File Explorer** нет, можно добавить: **Window-> Show View-> File Explorer**). На вкладке **File Explorer** найти директорию **sdcard/** (в нашем случае оказалась директория **storage/sdcard/**, имеет смысл при создании жестов обратить внимание в какую директорию Gesture Builder их сохраняет). На рис. 11.3 показана вкладка **File Explorer** в компоновке DDMS.

Чтобы скопировать файл жестов с эмулятора в проект, необходимо выбрать его и нажать кнопку "**Pull a file from the device**", выделенную на рис. 11.3 красным подобием окружности. Откроется диалог с предложением выбрать папку, в которую необходимо скопировать жесты, здесь надо найти папку проекта, в ней папку **res/raw/** (если папки **raw/** нет, ее необходимо создать) и нажать кнопку **Сохранить**. Теперь жесты есть в нашем проекте и их можно использовать.

11.3 Использование созданных жестов в приложении

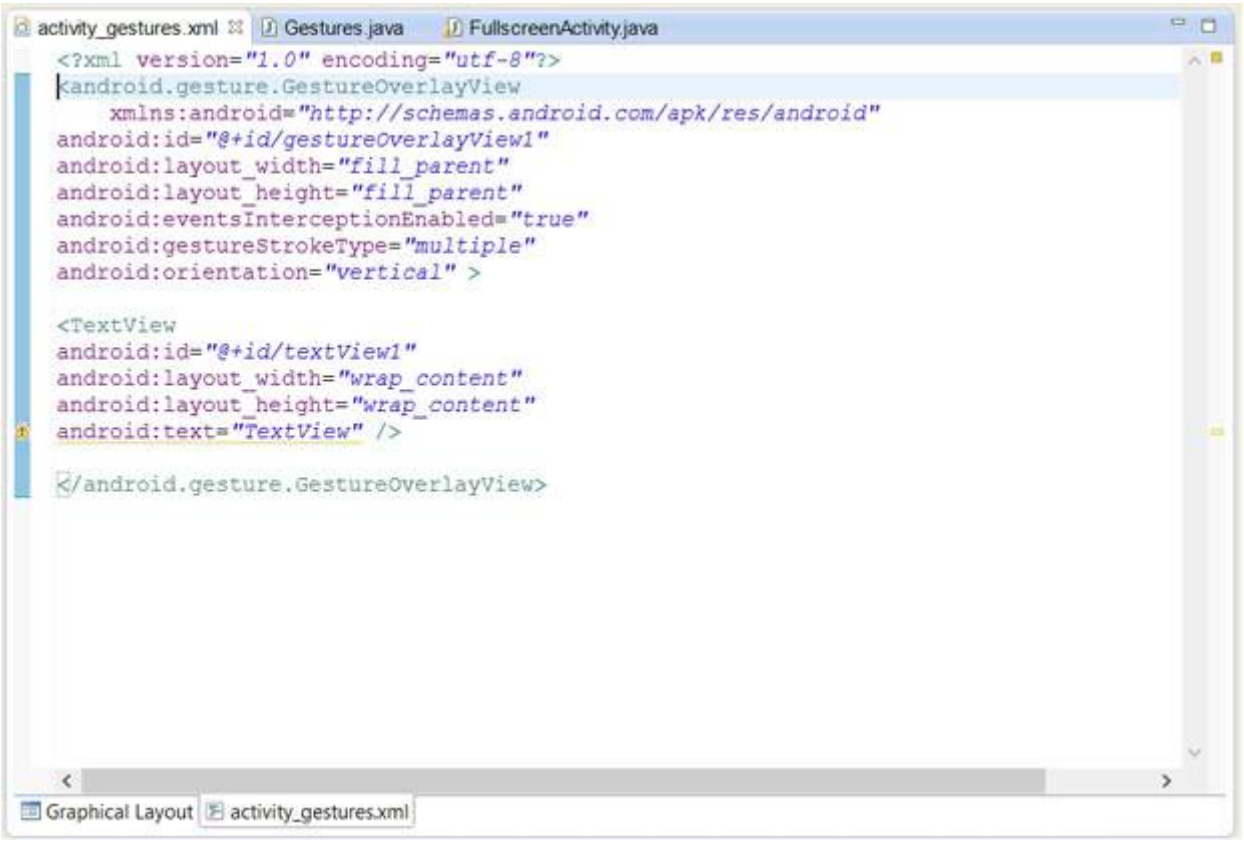
Для распознавания жестов необходимо добавить элемент GestureOverlayView в XML файл активности. И этот файл может выглядеть, например, как показано на рис. 11.4:



увеличить изображение

Рис. 11.4. XML файл активности приложения, элемент GestureOverlayView обычный компонент интерфейса пользователя

Можно добавить элемент GestureOverlayView поверх всех компонентов, как прозрачный слой, в этом случае XML файл активности может выглядеть так, как показано на рис. 11.5.



увеличить изображение

Рис. 11.5. XML файл активности приложения, элемент GestureOverlayView поверх всех компонентов интерфейса пользователя

Далее необходимо обработать ввод жеста пользователя, сравнить с загруженными жестами, и либо определить жест, либо сообщить пользователю, что такого жеста нет. Теперь вся работа будет выполняться в java файле, описывающем главную (и единственную) активность приложения. Внесем в этот класс следующие дополнения:

- Класс активности должен реализовывать интерфейс OnGesturePerformedListener, для этого в объявление класса добавим конструкцию:
- `implements OnGesturePerformedListener;`

- Нам понадобятся экземпляры классов GestureLibrary и GestureOverlayView, поэтому в качестве полей класса активности объявим следующие переменные:
- `GestureLibrary gLib;`
- `GestureOverlayView gestures;`

- В методе onCreate() выполним следующие действия:
- `gLib = GestureLibraries.fromRawResource(this, R.raw.gestures);`
- `if (!gLib.load()) {`
- `finish();`
- `}`

В первой строке выполнена инициализация переменной gLib жестами, загруженными из файла gestures папки **res/raw/**.

Оператор if выполняет проверку загружены ли жесты, если нет, выполняется выход из приложения.

- Добавим в метод onCreate() еще две строчки:
- `gestures = (GestureOverlayView) findViewById(R.id.gestureOverlayView1);`
- `gestures.addOnGesturePerformedListener(this);`

Для инициализации переменной gesture и подключения к ней слушателя событий появления жеста.

- И наконец напишем реализацию метода OnGesturePerformed(), который и будет вызываться при появлении события, соответствующего какому-либо жесту.

```
public void onGesturePerformed(GestureOverlayView overlay, Gesture gesture) {
    //Создаёт ArrayList с загруженными из gestures жестами
        ArrayList<Prediction> predictions = gLib.recognize(gesture);
        if (predictions.size() > 0) {
          //если загружен хотябы один жест из gestures
            Prediction prediction = predictions.get(0);
            if (prediction.score > 1.0) {
                if (prediction.name.equals("one"))
                tvOut.setText("1");
                else if (prediction.name.equals("stop"))
                  tvOut.setText("stop");
                else if (prediction.name.equals("two"))
                  tvOut.setText("2");
            }else{
              tvOut.setText("Жест неизвестен");
            }
        }
    }
```

В приложении всего лишь распознаются жесты и в информационное поле выводится информация о том, что за жест был использован. В листинге 11.1 представлен возможный код приложения.

11.4 Заключение

В качестве практического задания предлагаем реализовать жестами ввод чисел в приложении "Угадайка", разработанном в лабораторной работе второй темы. Создать жесты "0", "1", "2", "3", "4", "5", "6", "7", "8", "9" для ввода цифр и жест "S" для остановки ввода числа. В приложение добавить распознавание этих жестов, преобразование их в число и сравнение полученного числа с загаданным.

Еще варианты для самостоятельной работы:

1. разработать простой калькулятор с жестовым вводом чисел и операций;
2. разработать блокнотик для заметок с рукописным вводом текста.

```
package com.example.lab5_2_gestures;
```

```
import java.util.ArrayList;

import android.os.Bundle;
import android.app.Activity;
import android.gesture.Gesture;
import android.gesture.GestureLibraries;
import android.gesture.GestureLibrary;
import android.gesture.GestureOverlayView;
import android.gesture.GestureOverlayView.OnGesturePerformedListener;
import android.gesture.Prediction;
import android.view.Menu;
import android.widget.TextView;

public class Gestures extends Activity implements OnGesturePerformedListener{

  GestureLibrary gLib;
  GestureOverlayView gestures;
  TextView tvOut;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_gestures);
    tvOut=(TextView)findViewById(R.id.textView1);
        //Загрузка жестов (gestures) из res/raw/gestures
        gLib = GestureLibraries.fromRawResource(this, R.raw.gestures);
        if (!gLib.load()) {
          //Если жесты не загружены, то выход из приложения
            finish();
        }

       gestures = (GestureOverlayView)
findViewById(R.id.gestureOverlayView1);
       gestures.addOnGesturePerformedListener(this);
  }

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

  public void onGesturePerformed(GestureOverlayView overlay, Gesture gesture)
{
    //Создаёт ArrayList с загруженными из gestures жестами
        ArrayList<Prediction> predictions = gLib.recognize(gesture);
        if (predictions.size() > 0) {
          //если загружен хотя бы один жест из gestures
            Prediction prediction = predictions.get(0);
            if (prediction.score > 1.0) {
                if (prediction.name.equals("one"))
                tvOut.setText("1");
                else if (prediction.name.equals("stop"))
                  tvOut.setText("stop");
                else if (prediction.name.equals("two"))
                  tvOut.setText("2");
            }else{
              tvOut.setText("Жест неизвестен");
            }
        }
    }

}
```

Листинг 11.1. Распознавание жестов загруженных в файл res/raw/gestures
Внимание! Если Вы увидите ошибку на нашем сайте, выделите её и нажмите Ctrl+Enter.
© Национальный Открытый Университет "ИНТУИТ", 2016 | www.intuit.ru

Самостоятельная работа 7. Многооконное приложение

Цель лабораторной работы:

Разработка многооконного приложения, предоставляющего возможности: воспроизведения аудио и видео файлов, создания и отображения фотоснимков.

Задачи лабораторной работы:

- настроить интерфейс и реализовать логику активности для работы с камерой;
- настроить интерфейс и реализовать логику активности для воспроизведения аудио и видео;
- настроить интерфейс и реализовать логику активности для просмотра изображений;
- настроить интерфейс и реализовать логику главной активности приложения.

12.1 Введение

Для достижения цели, поставленной в лабораторной работе, сформулируем требования к разрабатываемому приложению, назовем его "Media".

Приложение предоставляет пользователю возможность выбора рода деятельности:

- работа с камерой для создания снимков;
- воспроизведение аудио и видео;
- просмотр изображений.

В приложении предполагается реализовать четыре активности:

- главная активность, предназначена для выбора рода деятельности, содержит три кнопки, нажатие на каждую кнопку вызывает к жизни соответствующую активность;
- активность для работы с камерой и создания снимков;
- активность для воспроизведения аудио и видео;
- активность для просмотра изображений.

12.2 Создание приложения

Создадим новое приложение, активность, полученную при создании проекта, оставим с именем **MainActivity**, она будет главной активностью приложения. Добавим в приложение еще три активности: **New->Other...->Android Activity**.

В нашем приложении созданные активности имеют следующие имена:

CameraActivity для работы с камерой и создания снимков;

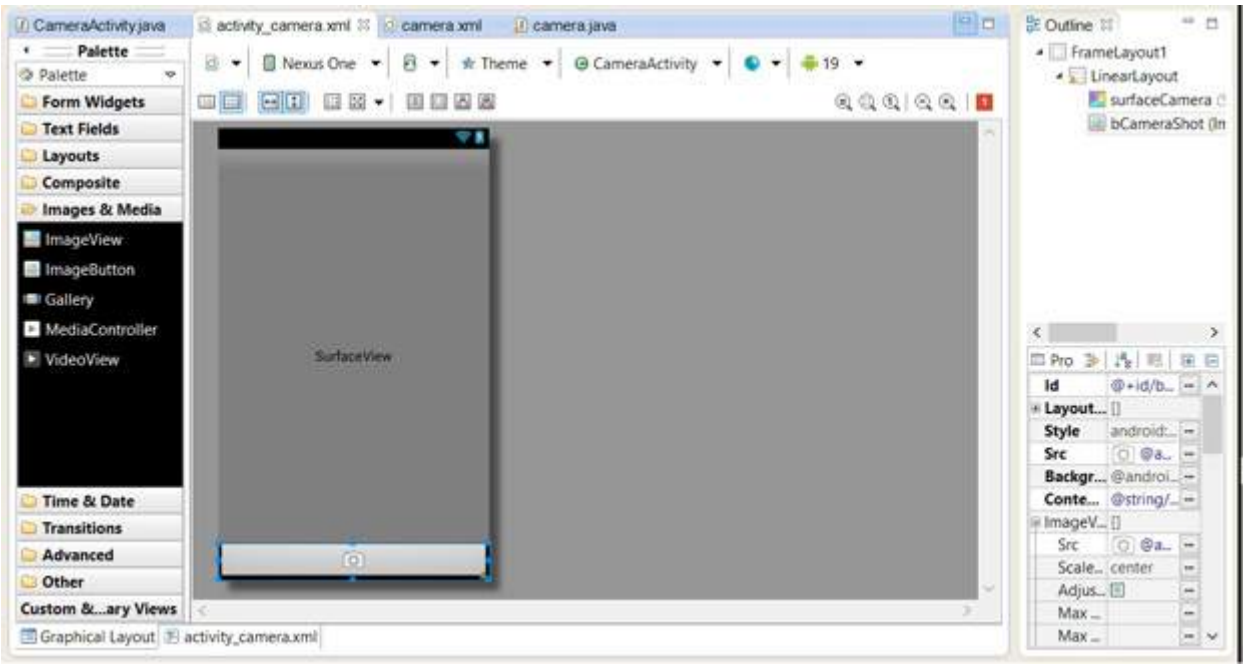
MediaActivity для воспроизведения видео и аудио;

GalleryActivity для просмотра изображений.

Далее продолжим работу с этими активностями, настроим интерфейс и реализуем логику для каждой из них. Начнем с активностей, отвечающих за тот или иной вид деятельности, главную активность приведем в порядок в самом конце работы.

12.3 Настройка интерфейса и реализация логики активности для работы с камерой

Настроим интерфейс активности для работы с камерой. Нам понадобится окно предварительного просмотра, добавим в окно активности элемент SurfaceView. А так же нам понадобится кнопка для выполнения снимков, добавим в окно активности элемент ImageButton. Далее предлагаем настроить интерфейс самостоятельно. В нашем случае активность выглядит так, как показано на рис. 12.1.



увеличить изображение
Рис. 12.1. Интерфейс активности для работы с камерой

Реализуем логику активности, в данном случае необходимо при нажатии на кнопку выполнить снимок. Работаем с java файлом, описывающим соответствующий класс активности, **CameraActivity.java**.

Объявим следующие поля класса активности:

```
private Camera camera; //для проведения всех операций с камерой
private SurfaceHolder surfaceHolder; //длязадания preview
private SurfaceView preview; //для отображения окна предпросмотра
private View shotBtn;  //для выполнения снимка (кнопка)
```

Для начала работы с камерой необходимо ее инициализировать, сделать это лучше в методе onResume() класса активности, для инициализации используется следующая конструкция:

```
camera = Camera.open();
```

После завершения работы с камерой, необходимо ее освободить для других приложений, сделать это лучше в методе onPause() класса активности, для освобождения камеры используется следующая конструкция:

```
Camera.release();
```

Для активности, работающей с камерой, имеет смысл сразу задать расположение экрана следующим образом

```
setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
```

в противном случае, приходится, например, в методе surfaceCreated() проверять расположение экрана и поворачивать окно предварительного просмотра, что не очень удобно, так как поворот экрана занимает некоторое время.

Выполним настройку окна предварительного просмотра:

```
preview = (SurfaceView) findViewById(R.id.surfaceCamera);
surfaceHolder = preview.getHolder();
surfaceHolder.addCallback(new MyCallback());
```

метод addCallback() используется для отслеживания изменений окна предпросмотра. Параметром этого метода служит экземпляр класса реализующего интерфейс SurfaceHolder.Callback, в нашем случае для реализации этого интерфейса создан внутренний класс активности MyCallback.

В этом классе необходимо реализовать методы:

```
public void surfaceCreated(SurfaceHolder holder);
public void surfaceChanged(SurfaceHolder holder, int format,
int width, int height);
public void surfaceDestroyed(SurfaceHolder holder);
```

Мы реализуем метод surfaceCreated(), оставшиеся два метода запишем с пустой реализацией.

В методе surfaceCreated() зададим окно предварительного просмотра:

```
camera.setPreviewDisplay(holder);
```

Выполним необходимые настройки окна предпросмотра (см. Листинг 12.1).

Запустим отображение окна предварительного просмотра:

```
camera.startPreview();
```

Пришло время обсудить и реализовать выполнение фотосъемки. Для того, чтобы сделать снимок необходимо вызвать метод takePicture(). Вызов этого метода можно выполнить из метода-обработчика события нажатия кнопки, тогда при нажатии на кнопку сразу будет

получена фотография, а можно "растянуть удовольствие" и воспользоваться предварительной автофокусировкой.

Сначала инициализируем кнопку и добавим слушателя события нажатия в методе onCreate() активности:

```
shotBtn = findViewById(R.id.bCameraShot);
shotBtn.setOnClickListener(new MyViewListener());
```

Вданномслучаеслушателемявляетсяэкземплярвнутреннегокласса MyViewListener, которыйреализуетинтерфейс View.OnClickListener.

```
class MyViewListener implements View.OnClickListener{
    @Override
    public void onClick(View v){
        if (v == shotBtn){
            camera.autoFocus(new MyAutoFocusCallback());
}
    }
}
```

В методе onClick() вызываем обработчик автофокуса, слушателем события автофокусировки в данном случае является экземпляр внутреннего класса MyAutoFocusCallback, который реализует интерфейс Camera.AutoFocusCallback.

```
class MyAutoFocusCallback implements Camera.AutoFocusCallback{
    @Override
    public void onAutoFocus(boolean paramBoolean, Camera
paramCamera){
    if (paramBoolean){
        // если удалось сфокусироваться, делаем снимок
paramCamera.takePicture(null, null, null,
                                        new MyPictureCallback());
        }
    }
}
```

Метод takePicture() имеетчетырепараметра:

Camera.ShutterCallback shutter	- вызывается в момент получения изображения с матрицы;
Camera.PictureCallback raw	- программе передаются для обработки raw данные (если поддерживается аппаратно);
Camera.PictureCallback postview	- программе передаются полностью обработанные данные (если поддерживается аппаратно);
Camera.PictureCallback jpg	- программе передается изображение в формате jpg. Здесь может быть организована запись изображения на карту памяти.

В нашем случае последним параметром является экземпляр класса MyPictureCallback(), который реализует интерфейс Camera.PictureCallback. В этом классе реализован единственный метод (см. Листинг 12.1):

```
public void onPictureTaken(byte[] paramArrayOfByte, Camera paramCamera){...}
```

В листинге 12.1 представлен код активности, реализующей работу с камерой и позволяющей выполнять снимки.

Чтобы разрешить приложению работать с камерой и сохранять фотографии на карте памяти, в манифест необходимо добавить следующие разрешения:

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

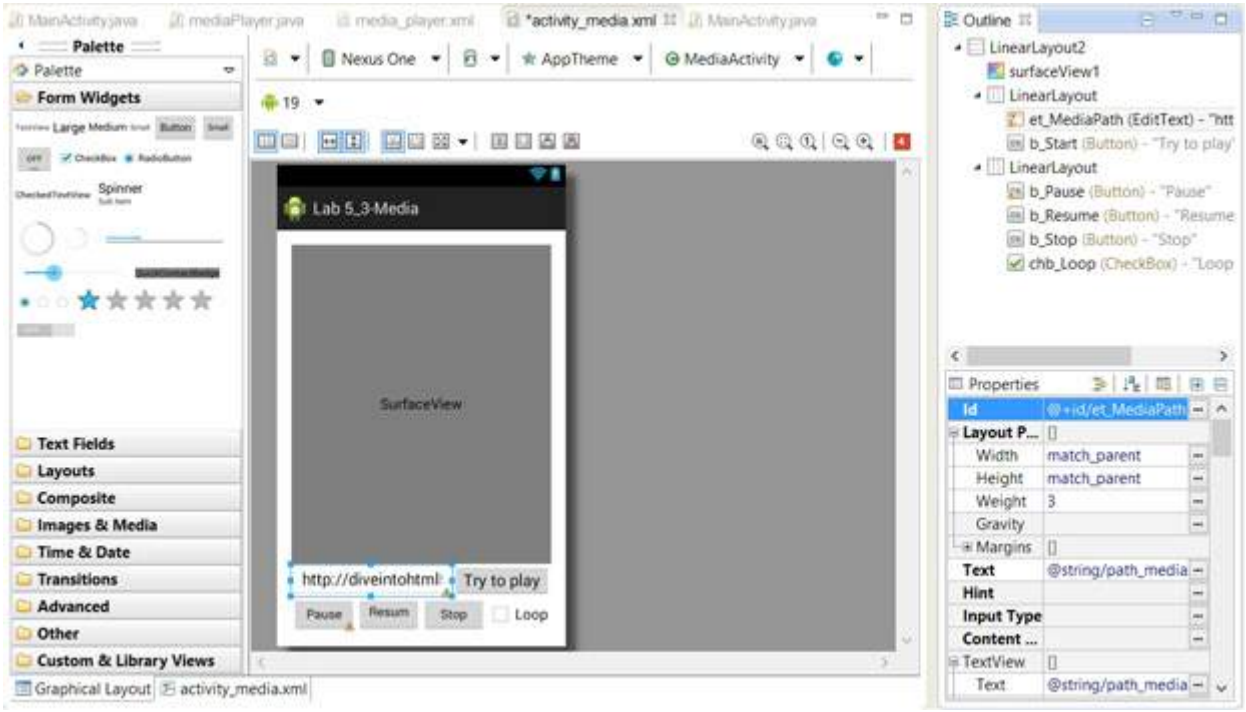
12.4 Настройка интерфейса и реализация логики активности для воспроизведения аудио и видео

Настроим интерфейс активности для воспроизведения аудио и видео. Нам понадобится окно предварительного просмотра, добавим в окно активности элемент SurfaceView. Для работы приложения необходимо поле для задания расположения медиа контента для проигрывания, добавим элемент EditText, а также потребуются несколько кнопок, добавим в окно активности следующие кнопки:

b_Start	- для запуска воспроизведения контента с начала, зададим свойству On Click этой кнопки значение onClickStart;
b_Pause	- для приостановки воспроизведения, зададим свойству On Click этой кнопки значение onClick;
b_Resume	- для продолжения воспроизведения с места приостановки, зададим свойству On Click этой кнопки значение onClick;
b_Stop	- для полной остановки воспроизведения, зададим свойству On Click этой кнопки значение onClick.

Для возможности выбора зацикленного воспроизведения медиа контента добавим в окно активности элемент CheckBox.

Далее предлагаем настроить интерфейс самостоятельно. В нашем случае активность выглядит так, как показано на рис. 12.2.



увеличить изображение
Рис. 12.2. Интерфейс активности для воспроизведения аудио и видео

Реализуем логику активности, в данном случае необходимо выполнять следующие действия:

- при нажатии на кнопку **Try to play**, запускается воспроизведение с начала контента, расположенного по адресу, указанному в поле ввода, расположенном слева от кнопки;
- при нажатии на кнопку **Pause**, воспроизведение контента приостанавливается;
- при нажатии на кнопку **Resume**, воспроизведение контента продолжается с места приостановки;
- при нажатии на кнопку **Stop**, воспроизведение контента останавливается и может быть возобновлено только с начала, т. е. необходимо снова нажать **Try to play**.

Элемент Loop управляет возможностью повтора воспроизведения.

Работать будем с файлом **MediaActivity.java**, описывающим соответствующий класс активности.

Объявим следующие поля класса активности:

```
MediaPlayer mediaPlayer; //управляет воспроизведением
CheckBox chbLoop;  //управляет режимом повтора
```

В методе onCreate() активности настроим чек-бокс так, чтобы он включал/выключал режим повтора для плеера.

```
chbLoop = (CheckBox) findViewById(R.id.chb_Loop);
chbLoop.setOnCheckedChangeListener(new
                        OnCheckedChangeListener() {
    @Override
```

```
    public void onCheckedChanged(CompoundButton buttonView,
                                 boolean isChecked) {
        if (mediaPlayer != null)
        mediaPlayer.setLooping(isChecked);
    }
});
```

Метод onClickStart() используется для обработки нажатий на кнопку **Try to play**. В этом метода сначала освобождаем ресурсы текущего проигрывателя, используя вызов метода:

```
releaseMP();
```

Этот метод рекомендуется вызывать по окончанию использования плеера, а также при onPause/onStop, если нет острой необходимости держать объект.

Реализация метода releaseMP():

```
private void releaseMP() {
    if (mediaPlayer != null) {
        try {
            mediaPlayer.release();
            mediaPlayer = null;
        } catch (Exception e) {
          e.printStackTrace();
        }
    }
}
```

Продолжим рассмотрение метода onClickStart(). В нем после освобождения ресурсов, занимаемся подготовкой плеера к воспроизведению.

```
mediaPlayer = new MediaPlayer();
mediaPlayer.setDataSource(DATA);
mediaPlayer.setDisplay(((SurfaceView)
                findViewById(R.id.surfaceView1)).getHolder());
mediaPlayer.setOnPreparedListener(this);
mediaPlayer.prepareAsync();
```

Создаем новый объект класса MediaPlayer.

Используем метод setDataSource() для задания источника данных, в качестве параметра передаем строку из поля ввода, сохраненную в переменной DATA.

Используя метод setDisplay(), задаем экран для воспроизведения, в нашем случае элемент SurfaceView, добавленный в окно активности на этапе формирования интерфейса.

Метод prepareAsync() выполняет асинхронную подготовку плеера к воспроизведению и когда подготовка будет завершена сообщает об этом слушателю, указанному в методе setPreparedListener(). В нашем случае слушателем является сам класс активности, для этого он объявлен, как класс реализующий интерфейс OnPreparedListener. В случае готовности плеера вызывается метод onPrepared(MediaPlayer mp), объявленный в указанном интерфейсе и реализованный в классе активности, в этом методе выполняется запуск воспроизведения:

```
mp.start();
```

Существует еще метод prepare(), он также выполняет подготовку плеера, но в синхронном режиме. Для прослушивания файлов из интернета, необходимо использовать асинхронный режим, что мы и сделали.

Последние две строки метода onClickStart():

```
mediaPlayer.setLooping(chbLoop.isChecked());
mediaPlayer.setOnCompletionListener(this);
```

В первой из них определяется возможность циклического воспроизведения в зависимости от значения чек-бокса.

Во второй - задается слушатель для получения сообщения о достижении конца воспроизводимого контента. В нашем случае этим слушателем будет сам класс активности для этого он объявлен, как класс реализующий интерфейс OnCompletionListener. Полную версию метода onClickStart() можно найти в листинге 12.2.

Метод onClick() вызывается при нажатии на любую кнопку: **Pause**, **Resume**, **Stop**. В этом методе выполняется проверка какая кнопка была нажата и после этого выполняются соответствующие действия.

Если нажата кнопка **Pause**, выполняется следующая конструкция:

```
if (mediaPlayer.isPlaying())   mediaPlayer.pause();
```

Если нажата кнопка **Release**, выполняется следующая конструкция:

```
if (!mediaPlayer.isPlaying())  mediaPlayer.start();
```

Если нажата кнопка **Stop**, выполняется следующая конструкция:

```
mediaPlayer.stop();
```

Имеет смысл обратить внимание на метод активности:

```
protected void onDestroy() {
  super.onDestroy();
  releaseMP();
}
```

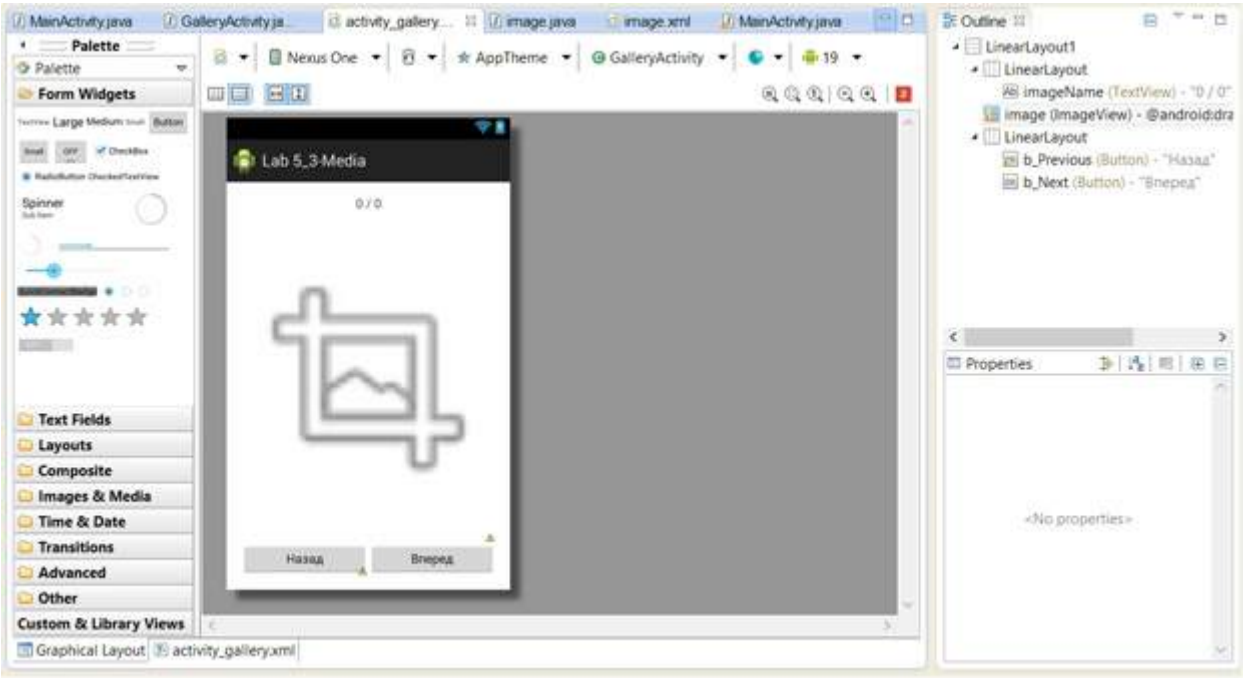
В этом методе обязательно необходимо освободить ресурсы, что и выполняется вызовом метода releaseMP().

Полный код класса MediaActivity представлен в листинге 12.2.

12.5 Настройка интерфейса и реализация логики активности для просмотра изображений

Настроим интерфейс активности для просмотра изображений. Нам потребуется элемент для просмотра изображений, добавим в окно активности элемент ImageView. Добавим информационное поле, т. е. элемент TextView, для отображения информации об общем количестве изображений в папке и номере просматриваемого изображения. Добавим две кнопки, для перемещения от одного изображения к другому вперед и назад.

Предлагаем настроить интерфейс самостоятельно. В нашем случае активность выглядит так, как показано на рис. 12.3. Но это, разумеется, не единственно возможный вариант.



увеличить изображение
Рис. 12.3. Интерфейс активности для просмотра изображений

Реализуем логику активности, данная активность ориентирована на просмотр снимков, сделанных в этом же приложении и сохраненных в папке: **/sdcard/TrainingMedia/**. Кнопка **Назад** выводит в окно просмотра предыдущий снимок, по отношению к уже отображенному, зададим свойству On Click этой кнопки значение onPrevious. Кнопка **Вперед** выводит в окно просмотра следующий снимок, по отношению к уже отображенному, зададим свойству On Click этой кнопки значение onNext.

Работать будем с файлом **GalleryActivity.java**, описывающим класс соответствующей активности.

Объявим поля класса активности:

```
int currentImage=0;
ArrayList<String> images;
ImageView imageView;
TextView nameView;
```

Настройку основных элементов для вывода изображений на экран выполним в методе onResume() активности, этот метод вызывается каждый раз, перед выводом активности на передний план (см. жизненный цикл активности в "Виды приложений и их структура").

```
images=new ArrayList<String>();

imageView=((ImageView)findViewById(R.id.image));
try{
    File imagesDirectory=new File("/sdcard/TrainingMedia/");
    images=searchImage(imagesDirectory);
    updatePhoto(Uri.parse(images.get(currentImage)));
}catch(Exception e){
    nameView.setText("Ошибка: Папка '/sdcard/TrainingMedia/' не
                                                        найдена");
}
```

Для получения списка изображений в переменную images используется метод searchImage(), этот метод, используя переданный в него в качестве параметра адрес директории с изображениями, находит файл с расширением .jpg, .png или .jpeg и добавляет его к списку изображений. Метод возвращает список доступных файлов с изображениями. Код метода представлен в листинге 12.3.

Метод updatePhoto() выполняет обновление счетчика фотографий в информационном поле и выводит на экран изображение, соответствующее переданному в метод URI. Код метода представлен в листинге 12.3.

В методе onPause() активности выполняется очистка списка изображений и освобождение памяти. Этот метод вызывается каждый раз, как активность теряет фокус ввода. Код метода представлен в листинге 12.3.

Осталось рассмотреть еще два метода: onPrevious() и onNext().

Первый метод вызывается, когда нажата кнопка **Назад**. В нем уменьшается номер текущего изображения и вызывается метод updatePhoto().

Второй метод вызывается, когда нажата кнопка **Вперед**. В нем увеличивается номер, текущего изображения и вызывается метод updatePhoto().

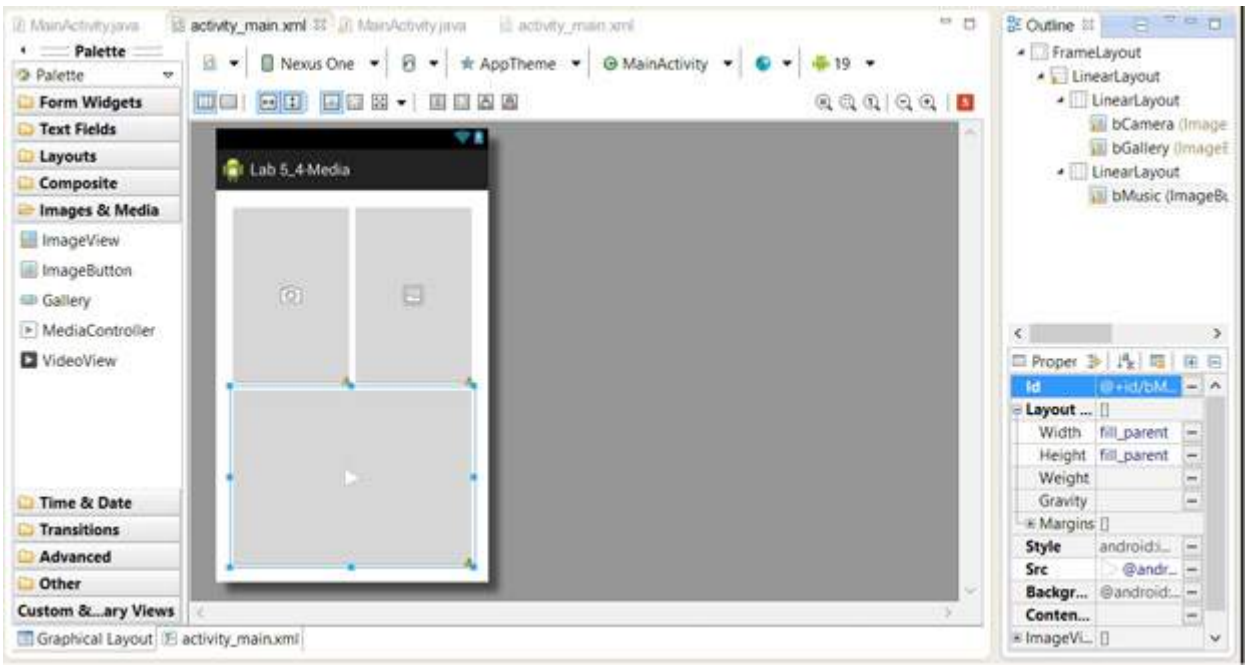
Полный код класса GalleryActivity представлен в листинге 12.3.

12.6 Настройка интерфейса и реализация логики главной активности приложения

Настроим интерфейс главной активности приложения. Эта активность содержит три кнопки (**Image Button**):

bCamera	- для вызова активности, предоставляющей возможности работы с камерой;
bGallery	- для вызова активности, предоставляющей возможности просмотра изображений;
bMusic	- для вызова активности, предоставляющей возможности воспроизведения аудио и видео.

Предлагаем настроить интерфейс самостоятельно. В нашем случае активность выглядит так, как показано на рис. 12.4. Но это, разумеется, не единственно возможный вариант.



[увеличить изображение]

Рис. 12.4. Интерфейс главной активности приложения

Реализуем логику главной активности. В данном случае достаточно настроить обработку событий нажатия на кнопки, таким образом, чтобы при нажатии на кнопку запускалась соответствующая активность. Работаем с java файлом, описывающим класс активности.

- Создадим переменную btnClick, как реализацию интерфейса-слушателя OnClickListener:

```
OnClickListener btnClick=new OnClickListener() {
    @Override
    public void onClick(View v) {
        Click(v.getId());
    }
};
```

- Добавим слушателя события нажатия к каждой кнопке активности:

```
((ImageButton)findViewById(R.id.bMusic)).setOnClickListener(btnClick);
((ImageButton)findViewById(R.id.bCamera)).setOnClickListener(btnClick);
((ImageButton)findViewById(R.id.bGallery)).setOnClickListener(btnClick)
;
```

- В методе onClick() слушателя вызываем метод Click(), в котором выполняется запуск соответствующей активности:

```
protected void Click(int view){
    Intent intent=null;
    switch (view){
        case R.id.bMusic: intent=new Intent(this,mediaPlayer.class);
break;
        case R.id.bGallery: intent=new
Intent(this,GalleryActivity.class); break;
        case R.id.bCamera: intent=new
Intent(this,CameraActivity.class); break;
```

```
        default: break;
    }
    if(intent!=null){
        startActivity(intent);
    }
}
```

В листинге 12.4 представлен код главной активности.

```
package com.example.lab5_4_media;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import android.hardware.Camera;
import android.hardware.Camera.Size;
import android.os.Bundle;
import android.app.Activity;
import android.content.pm.ActivityInfo;
import android.content.res.Configuration;
import android.util.Log;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.View;
import android.view.Window;
import android.view.WindowManager;
import android.view.ViewGroup.LayoutParams;

public class CameraActivity extends Activity {

  private Camera camera;
  private SurfaceHolder surfaceHolder;
  private SurfaceView preview;
  private View shotBtn;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // если хотим, чтобы приложение постоянно имело портретную ориентацию
    setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);

    // если хотим, чтобы приложение было полноэкранным
    getWindow().addFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN);

    // и без заголовка
    requestWindowFeature(Window.FEATURE_NO_TITLE);

    setContentView(R.layout.activity_camera);

    preview = (SurfaceView) findViewById(R.id.surfaceCamera);
    surfaceHolder = preview.getHolder();
    surfaceHolder.addCallback(new MyCallback(this));
    surfaceHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);

    shotBtn = findViewById(R.id.bCameraShot);
    shotBtn.setOnClickListener(new MyViewListener());
  }

  @Override
  protected void onResume() {
```

```
    super.onResume();
    camera = Camera.open();
  }

  @Override
  protected void onPause(){
    super.onPause();
    if (camera != null){
      camera.setPreviewCallback(null);
      camera.stopPreview();
      camera.release();
      camera = null;
    }
  }

  class MyCallback implements SurfaceHolder.Callback{

    Activity host;

    MyCallback(Activity act){
      host=act;
    }

    @Override
    public void surfaceChanged(SurfaceHolder holder, int format, int width,
    int height){}
    @Override
    public void surfaceCreated(SurfaceHolder holder){
      try {
        camera.setPreviewDisplay(holder);
        camera.setPreviewCallback(new MyPreviewCallback());
      }
      catch (IOException e){
        Log.d("myLogs","Ошибка камеры");
        e.printStackTrace();
      }
      Size previewSize = camera.getParameters().getPreviewSize();
      float aspect = (float) previewSize.width / previewSize.height;

      int previewSurfaceWidth = preview.getWidth();
      int previewSurfaceHeight = preview.getHeight();

      LayoutParams lp = preview.getLayoutParams();

      // здесь корректируем размер отображаемого previev, чтобы не было
искажений

      if (host.getResources().getConfiguration().orientation !=
              Configuration.ORIENTATION_LANDSCAPE){
        // портретный вид
        camera.setDisplayOrientation(90);
        lp.height = previewSurfaceHeight;
        lp.width = (int) (previewSurfaceHeight / aspect);
      }
      else {
        // ландшафтный
        camera.setDisplayOrientation(0);
        lp.width = previewSurfaceWidth;
        lp.height = (int) (previewSurfaceWidth / aspect);
      }
      preview.setLayoutParams(lp);
      camera.startPreview();
    }
```

```
    @Override
    public void surfaceDestroyed(SurfaceHolder holder){}
  }

  class MyViewListener implements View.OnClickListener{

    @Override
    public void onClick(View v) {
      if (v == shotBtn) {
        // либо делаем снимок непосредственно здесь
        // либо включаем обработчик автофокуса

        //camera.takePicture(null, null, null, this);
        camera.autoFocus(new MyAutoFocusCallback());
      }
    }
  }

  class MyAutoFocusCallback implements Camera.AutoFocusCallback{

    @Override
    public void onAutoFocus(boolean paramBoolean, Camera paramCamera){
      if (paramBoolean){
        // если удалось сфокусироваться, делаем снимок
        paramCamera.takePicture(null, null, null, new MyPictureCallback());
      }
    }
  }

  class MyPictureCallback implements Camera.PictureCallback{

    @Override
    public void onPictureTaken(byte[] paramArrayOfByte, Camera paramCamera){
      // сохраняем полученные jpg в папке /sdcard/CameraExample/
      // имя файла - System.currentTimeMillis()
      try {
        File saveDir = new File("/sdcard/CameraExample/");
        if (!saveDir.exists()) {
          saveDir.mkdirs();
        }

        FileOutputStream os = new
FileOutputStream(String.format("/sdcard/CameraExample/%d.jpg",
System.currentTimeMillis()));
        os.write(paramArrayOfByte);
        os.close();
      }
      catch (Exception e) {}

      // после того, как снимок сделан, показ превью отключается.
      // Необходимо включить его
      paramCamera.startPreview();
    }
  }

  class MyPreviewCallback implements Camera.PreviewCallback{

    @Override
    public void onPreviewFrame(byte[] paramArrayOfByte, Camera paramCamera) {
      // здесь можно обрабатывать изображение, показываемое в preview
    }
  }
}
```

Листинг 12.1. Класс CameraActivity для работы с камерой

```
package com.example.lab5_4_media;

import android.media.MediaPlayer;
import android.media.MediaPlayer.OnCompletionListener;
import android.media.MediaPlayer.OnPreparedListener;
import android.os.Bundle;
import android.view.Gravity;
import android.view.SurfaceView;
import android.view.View;
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.EditText;
import android.widget.Toast;
import android.widget.CompoundButton.OnCheckedChangeListener;
import android.app.Activity;

public class MediaActivity extends Activity implements OnPreparedListener,
OnCompletionListener{

  MediaPlayer mediaPlayer;
  CheckBox chbLoop;

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

      chbLoop = (CheckBox) findViewById(R.id.chb_Loop);
    chbLoop.setOnCheckedChangeListener(new OnCheckedChangeListener() {
      @Override
      public void onCheckedChanged(CompoundButton buttonView,
          boolean isChecked) {
        if (mediaPlayer != null)
          mediaPlayer.setLooping(isChecked);
      }
    });
    }

    public void onClickStart(View view) {
    releaseMP();

    String
DATA=((EditText)findViewById(R.id.et_MediaPath)).getText().toString();
    try {
      mediaPlayer = new MediaPlayer();
      mediaPlayer.setDataSource(DATA);
      mediaPlayer.setDisplay(((SurfaceView)
          findViewById(R.id.surfaceView1)).getHolder());
      //mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
      mediaPlayer.setOnPreparedListener(this);
      mediaPlayer.prepareAsync();
    } catch (Exception e) {
      showMessage("Ошибка воспроизведения");
    }

    if (mediaPlayer == null)
      return;

    mediaPlayer.setLooping(chbLoop.isChecked());
    mediaPlayer.setOnCompletionListener(this);
  }

    private void showMessage(String text){
```

```
    Toast toast = Toast.makeText(getApplicationContext(), text,
        Toast.LENGTH_SHORT);
    toast.setGravity(Gravity.CENTER, 0, 0);
    toast.show();
  }

    private void releaseMP() {
    if (mediaPlayer != null) {
      try {
        mediaPlayer.release();
        mediaPlayer = null;
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
  }

    public void onClick(View view) {
    if (mediaPlayer == null)
      return;
    switch (view.getId()) {
    case R.id.b_Pause:
      if (mediaPlayer.isPlaying())
        mediaPlayer.pause();
      break;
    case R.id.b_Resume:
      if (!mediaPlayer.isPlaying())
        mediaPlayer.start();
      break;
    case R.id.b_Stop:
      mediaPlayer.stop();
      break;
    }
  }

      @Override
  public void onPrepared(MediaPlayer mp) {
    mp.start();
  }

  @Override
  public void onCompletion(MediaPlayer mp) {}

  @Override
  protected void onDestroy() {
    super.onDestroy();
    releaseMP();
  }

}
```

Листинг 12.2. Класс MediaActivity

```
package com.example.lab5_4_media;

import java.io.File;
import java.util.ArrayList;
import android.net.Uri;
import android.os.Bundle;
import android.app.Activity;
import android.util.Log;
import android.view.View;
import android.widget.ImageView;
import android.widget.TextView;
```

```
public class GalleryActivity extends Activity {

  int currentImage=0;
  ArrayList<String> images;
  ImageView imageView;
  TextView nameView;

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

  @Override
  public void onResume(){
    super.onResume();

    currentImage=0;

    Log.d("myLogs","onResume cI="+currentImage);

    nameView=((TextView)findViewById(R.id.imageName));
    images=new ArrayList<String>();

    imageView=((ImageView)findViewById(R.id.image));
    try{
      File imagesDirectory=new File("/sdcard/TrainingMedia/");
      images=searchImage(imagesDirectory);
      updatePhoto(Uri.parse(images.get(currentImage)));
    }catch(Exception e){
      nameView.setText("Ошибка: Папка '/sdcard/TrainingMedia/' не найдена");
      Log.d("myLogs","Ошибка");
    }
  }

  @Override
    protected void onPause()
    {
        super.onPause();
        images.clear();
        Log.d("myLogs","onPause cI="+currentImage);
    }

  private ArrayList<String> searchImage(File dir){
    ArrayList<String> imagesFinded=new ArrayList<String>();
    for(File f:dir.listFiles()){
      if(!f.isDirectory()){
        String fileExt=getFileExt(f.getAbsolutePath());
        if(fileExt.equals("png") || fileExt.equals("jpg") ||
fileExt.equals("jpeg")){
          Log.d("myLogs","Файлнайден "+f.getAbsolutePath());
          imagesFinded.add(f.getAbsolutePath());
        }
      }
    }
    return imagesFinded;
  }

  public static String getFileExt(String filename){
    return filename.substring(filename.lastIndexOf(".") + 1);
  }

  public void updatePhoto(Uri uri){
    try{
```

```
      nameView.setText((currentImage+1)+"/"+images.size());
      imageView.setImageURI(uri);
    }catch(Exception e){
      nameView.setText("Ошибказагрузкифайла");
    }
  }

  public void onNext(View v){
    if(currentImage+1<images.size() && images.size()>0){
      currentImage++;
      updatePhoto(Uri.parse(images.get(currentImage)));
    }
  }

  public void onPrevious(View v){
    if(currentImage>0 && images.size()>0){
      currentImage--;
      updatePhoto(Uri.parse(images.get(currentImage)));
    }
  }



}
```

Листинг 12.3. Класс GalleryActivity

```
package com.example.lab5_4_media;

import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.util.Log;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ImageButton;

public class MainActivity extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    OnClickListener btnClick=new OnClickListener() {
      @Override
      public void onClick(View v) {
        Log.d("myLogs",v.getId()+"");
        Click(v.getId());
      }
    };
    ((ImageButton)findViewById(R.id.bMusic)).setOnClickListener(btnClick);
    ((ImageButton)findViewById(R.id.bCamera)).setOnClickListener(btnClick);
    ((ImageButton)findViewById(R.id.bGallery)).setOnClickListener(btnClick);
  }
  protected void Click(int view){
    Intent intent=null;
    Log.d("myLogs",view+"");
    switch (view){
      case R.id.bMusic: intent=new Intent(this,MediaActivity.class);   break;
      case R.id.bGallery: intent=new Intent(this,GalleryActivity.class);
break;
      case R.id.bCamera: intent=new Intent(this,CameraActivity.class); break;
      default: break;
    }
```

```
    if(intent!=null){
      Log.d("myLogs","Интент = "+intent.toString());
      startActivity(intent);
    }
  }
}
```

Листинг 12.4. Класс MainActivity

Внимание! Если Вы увидите ошибку на нашем сайте, выделите её и нажмите Ctrl+Enter.
© Национальный Открытый Университет "ИНТУИТ", 2016 | www.intuit.ru

Самостоятельная работа 8. Геолокационные возможности

Цель лабораторной работы:

разработка приложения, демонстрирующего геолокационные возможности.

Задачи лабораторной работы:

- Разработать приложение, получающее координаты устройства и отслеживающее их изменения.

13.1 Введение

В данной работе рассмотрим процесс создания приложения.

13.2 Разработка приложения, получающего координаты устройства и отслеживающего их изменение

Создадим приложение.

Далее будем править java файл, определяющий класс активности приложения. Внесем в этот класс следующие дополнения:

- Нам потребуется экземпляр класса LocationManager, поэтому в методе onCreate() запишем следующую конструкцию:

```
LocationManager mlocManager =
  (LocationManager) getSystemService(Context.LOCATION_SERVICE);
```

экземпляр класса LocationManager, как и большинство системных сервисов, создается с помощью вызова метода getSystemService().

- Укажем, что в приложении необходимо получать обновления координат, сделаем это с помощью вызова метода requestLocationUpdates() класса LocationManager:

```
mlocManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
              0, 0, mlocListener);
```

первый параметр метода указывает способ получения координат, для этого используются константы класса LocationManager:

GPS_PROVIDER	- сообщает, что координаты определяются с помощью GPS;
NETWORK_ PROVIDER	- сообщает, что координаты определяются с использованием сигналов сотовых вышек и WiFi.

В случае, когда необходимо получать координаты и с GPS, и от сотовых вышек необходимо вызвать метод requestLocationUpdates() дважды: один раз первый параметр должен быть равен GPS_PROVIDER, второй раз - NETWORK_ PROVIDER.

Второй и третий параметры метода управляют частотой обновлений. Второй определяет минимальное время между извещениями об обновлениях, третий - минимальное изменение расстояния между извещениями об обновлениях. Если оба эти параметра имеют нулевое значение, то извещения об обновлениях появляются настолько часто, насколько это возможно.

Последний параметр указывает на слушателя, который получает вызовы при обновлениях координат. В нашем случае слушателем является переменная mlocListener - реализация интерфейса LocationListener.

- Теперь необходимо объявить переменную mlocListener:

```
LocationListener mlocListener = new LocationListener(){
  public void onLocationChanged(Location location) {
    tvLat.append(" "+location.getLatitude());
    tvLon.append(" "+location.getLongitude());
  }
  public void onStatusChanged(String provider, int status,
                Bundle extras) {}
  public void onProviderEnabled(String provider) {}
  public void onProviderDisabled(String provider) {}
};
```

Переменная mlocListener инициализируется реализацией интерфейса LocationListener. Этот интерфейс содержит 4 метода, каждый из которых должен быть определен в реализации интерфейса. Нас интересует метод onLocationChanged(), который вызывается каждый раз при изменении показаний GPS датчика, в этом методе всего лишь выполняется вывод полученных координат в информационные поля.

- Чтобы разрешить получать обновления координат с помощью сигналов от сотовых вышек (NETWORK_PROVIDER) или с GPS датчика (GPS_PROVIDER), необходимо в файле манифеста добавить строку:

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

(для работы с сигналами сотовых вышек) или

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

(для работы GPS датчиком).

Без этих полномочий приложение "сломается" во время выполнения, когда попробует запросить обновление координат. Если в приложении в качестве источника координат используются и NETWORK_PROVIDER, и GPS_PROVIDER, в манифесте достаточно запросить только полномочия на ACESS_FINE_LOCATION. Тогда как ACCESS_COARSE_LOCATION позволяет работать только с NETWORK_PROVIDER.

В листинге 13.1 представлен код приложения, позволяющего получать координаты устройства и отслеживать их изменения, используя GPS приемник.

После создания приложения, разумеется, необходимо протестировать его работу. Проще всего это сделать, используя реальное устройство под управлением Android, но если устройства под рукой нет, можно использовать виртуальное устройство. Однако при этом необходимо решить вопрос, каким образом имитировать передачу данных о местоположении на эмулятор. Проще всего воспользоваться для этого DDMS.

В Eclipse откройте компоновку (Perspective) DDMS, в этой компоновке выберите вкладку **Emulator Control**. С помощью инструмента DDMS, можно имитировать обновление данных о местоположении несколькими способами:

- вручную передать значения широты и долготы на виртуальное устройство;
- использовать GPX файл, описывающий маршрут для считывания эмулятором;
- использовать KML файл, описывающий отдельные метки местности для последовательной передачи на виртуальное устройство.

13.3 Заключение

В этой части работы рассмотрели вопросы разработки приложений, способных получать координаты устройства и отслеживать их изменения. Разработанное приложение реагирует на изменения координат устройства и выдает новые координаты в соответствующие информационные поля.

```
package com.example.lab5_4_geolocation;

import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.app.Activity;
import android.content.Context;
import android.widget.TextView;

public class MainActivity extends Activity {

  TextView tvOut;
  TextView tvLon;
  TextView tvLat;

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

    tvOut = (TextView)findViewById(R.id.textView1);
    tvLon = (TextView)findViewById(R.id.longitude);
    tvLat = (TextView)findViewById(R.id.latitude);

    //Получаем сервис
    LocationManager mlocManager = (LocationManager)
                  getSystemService(Context.LOCATION_SERVICE);

    LocationListener mlocListener = new LocationListener(){
      public void onLocationChanged(Location location) {

      //Called when a new location is found by the network location provider.
        tvLat.append(" "+location.getLatitude());
        tvLon.append(" "+location.getLongitude());
      }
```

```
      public void onStatusChanged(String provider, int status, Bundle extras)
{}

      public void onProviderEnabled(String provider) {}

      public void onProviderDisabled(String provider) {}

    };

    //Подписываемся на изменения в показаниях датчика

    mlocManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0,
                mlocListener);

    //Если gps включен, то ... , иначе вывести "GPS is not turned on..."
    if (mlocManager.isProviderEnabled(LocationManager.GPS_PROVIDER)) {
      tvOut.setText("GPS is turned on...");

    } else {
      tvOut.setText("GPS is not turned on...");
    }
  }
}
```

Листинг 13.1. Получение геолокационных данных

Внимание! Если Вы увидите ошибку на нашем сайте, выделите её и нажмите Ctrl+Enter.
© Национальный Открытый Университет "ИНТУИТ", 2016 | www.intuit.ru

Самостоятельная работа 9. Использование сторонних библиотек

Цель лабораторной работы:

Научиться писать приложения с использованием сторонних библиотек

Задачи лабораторной работы:

- Научиться подключать библиотеки
- Научиться использовать библиотеки в своих приложениях

15.1 Использование библиотеки для построения графиков AChartEngine

Рассмотрим пример подключения библиотеки AChartEngine, предназначенной для построения графиков. На сайте разработчика (http://code.google.com/p/achartengine) помимо самой библиотеки можно найти подробную документацию, оформленную в стиле Javadoc pages, примеры использования библиотеки, а также ее исходный код.

В разделе **Downloads** также можно скачать демонстрационные примеры использования библиотеки.

1. Создайте проект **Graphics**.
2. Чтобы подключить библиотеку, нужно скачать ее с сайта разработчика, потом просто перетащить из проводника в папку **libs** вашего проекта в Eclipce.
3. Теперь нужно добавить строку <activity android:name="org.achartengine.GraphicalActivity"/> в раздел <application> в файле манифеста вашего проекта. Библиотека подключена.
4. Теперь перейдем к файлу **MainActivity.java**. Создайте и инициализируйте три массива (в методе onCreate()). В первом будут содержаться цвета, во втором - подписи, в третьем - значения.

```
5. int[] values = new int[] { 25,25,25,25 };
6. String[] bars = new String[] {"Bananas",  "Kiwi", "Oranges", "Cream"};
7. int[] colors = new int[] { Color.YELLOW,  Color.GREEN, Color.RED,
   Color.WHITE };
```

8. Создайте объект CategorySeries и загрузите в него массивы строк с подписями и значения. При построении диаграмм будут использоваться именно эти данные.

```
9. CategorySeries series = new CategorySeries("Pie Chart");
```

10. Создайтеобъект DefaultRenderer. CategorySeries и DefaultRenderer определенывбиблиотеке AChartEngine.

```
11.  DefaultRenderer dr = new DefaultRenderer();
```

12. Для каждого цвета в диаграмме создайте SimpleSeriesRenderer и установите его в этот цвет. Затем каждый SimpleSeriesRenderer добавьте в DefaultRenderer. Таким образом, имеем один Render на каждый Series. К слову, Render может включать другой Render, таким образом можно создавать подкатегории.

```
13.  for (int v=0; v<4; v++){
14.    series.add(bars[v], values[v]);
15.  SimpleSeriesRenderer r = new SimpleSeriesRenderer();
```

```
16.     r.setColor(colors[v]);
17.    dr.addSeriesRenderer(r);
18.  }
```

19. Создайте объект PieChartIntent и передайте ему CategorySeries и DefaultRenderer.

```
20.  return ChartFactory.getPieChartIntent(this, series, dr, "Fruit
   Salad");
```

С помощью PieChartIntent можно показать диаграмму, как отдельную деятельность.

21. С помощью методов библиотеки можно изменить размер и цвет подписей, включить зум:

```
22.  dr.setZoomEnabled(true);
23.  dr.setChartTitleTextSize(20);
24.  dr.setLegendTextSize(TEXT_SIZE);
25.  dr.setChartTitleTextSize(20);
26.  dr.setZoomButtonsVisible(false);
27.  dr.setLabelsTextSize(TEXT_SIZE);
28.  dr.setLegendTextSize(TEXT_SIZE);
29.  dr.setLabelsColor(Color.BLACK);
```

30. Для удобства можно создать дополнительную переменную, в которой будет храниться размер текста.

```
31.  private static final int TEXT_SIZE = 40;
32.
33.    Intent intent = buildIntent();
34.      startActivity(intent);
35.      }
36.
37.      public Intent buildIntent() {
38.        int[] values = new int[] { 25,25,25,25 };
39.      . . .
40.
41.
42.        dr.addSeriesRenderer(r);
43.      }
44.      dr.setChartTitleTextSize(20);
45.      . . .
46.      dr.setLabelsColor(Color.BLACK);
47.    return ChartFactory.getPieChartIntent(
48.    this, series, dr, "Fruit Salad");
49.      }
```

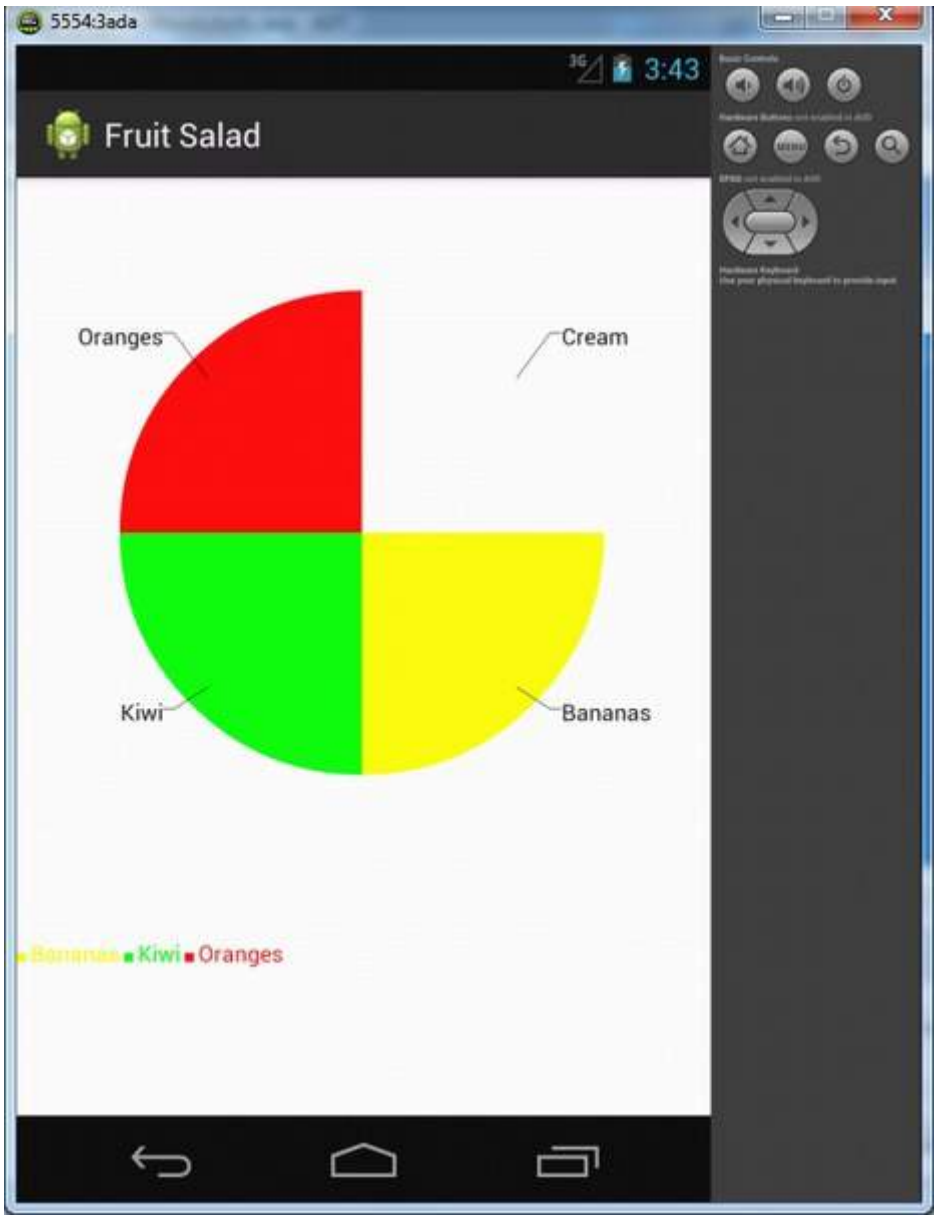
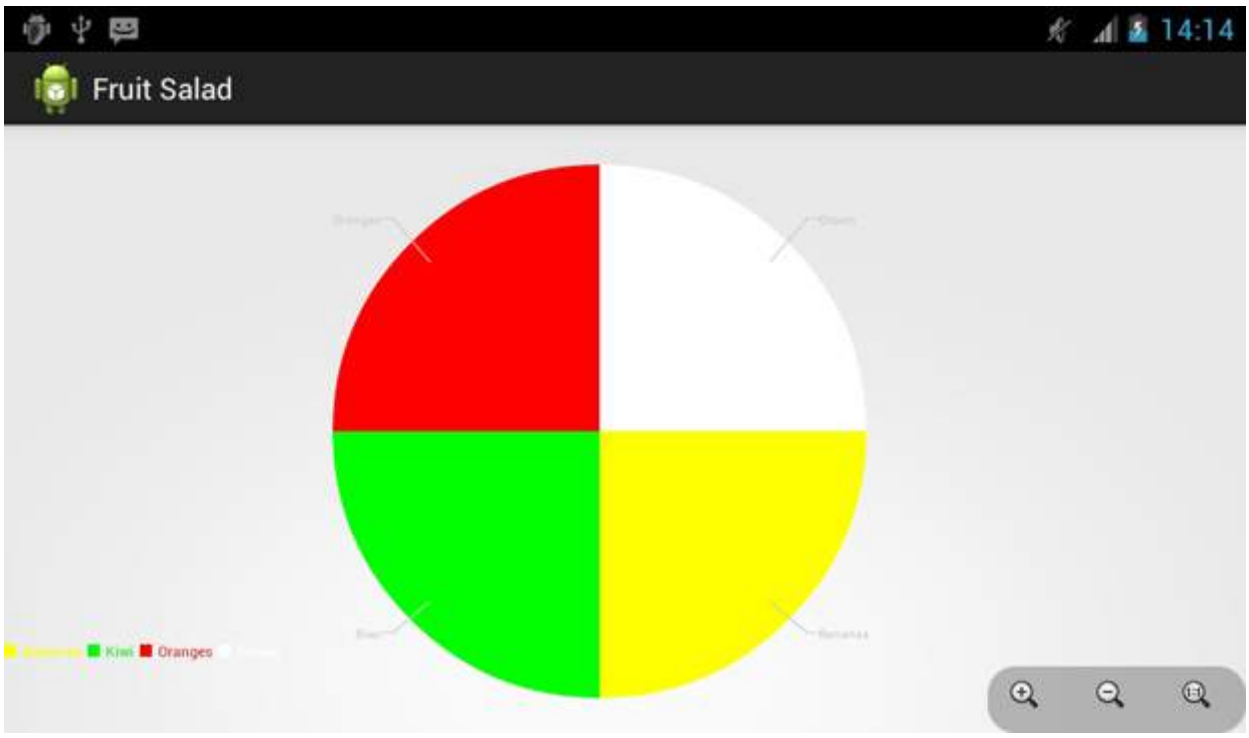


Рис. 15.1. Приложение, запущенное на эмуляторе



увеличить изображение

Рис. 15.2. Приложение, запущенное на устройстве

```
package com.mypackage.graphics;


import org.achartengine.ChartFactory;
import org.achartengine.chart.PieChart;
import org.achartengine.model.CategorySeries;
import org.achartengine.renderer.DefaultRenderer;
import org.achartengine.renderer.SimpleSeriesRenderer;
import android.content.Context;
import android.content.Intent;
import android.graphics.Color;
import android.app.Activity;
import android.os.Bundle;

public class MainActivity extends Activity {
private static final int TEXT_SIZE = 40;
    @Override
    public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
          Intent intent = buildIntent();
        startActivity(intent);
    }

    public Intent buildIntent() {
        int[] values = new int[] { 25,25,25,25 };
        String[] bars = new String[] {"Bananas",  "Kiwi", "Oranges",
"Cream"};
        int[] colors = new int[] { Color.YELLOW,  Color.GREEN, Color.RED,
Color.WHITE };

        CategorySeries series = new CategorySeries("Pie Chart");
        DefaultRenderer dr = new DefaultRenderer();

        for (int v=0; v<4; v++){
            series.add(bars[v], values[v]);
            SimpleSeriesRenderer r = new SimpleSeriesRenderer();
```

```
            r.setColor(colors[v]);
            dr.addSeriesRenderer(r);
        }
dr.setZoomButtonsVisible(true);
        dr.setZoomEnabled(true);
        dr.setChartTitleTextSize(20);
        dr.setLegendTextSize(TEXT_SIZE);
        dr.setChartTitleTextSize(20);
        dr.setZoomButtonsVisible(false);
        dr.setLabelsTextSize(TEXT_SIZE);
        dr.setLegendTextSize(TEXT_SIZE);
        dr.setLabelsColor(Color.BLACK);
        return ChartFactory.getPieChartIntent(this, series, dr, "Fruit
Salad");
}

}
```

Листинг 15.1. Код MainActivity.java

Внимание! Если Вы увидите ошибку на нашем сайте, выделите её и нажмите Ctrl+Enter.
© Национальный Открытый Университет "ИНТУИТ", 2016 | www.intuit.ru

Самостоятельная работа 10. Работа с базами данных в Android

Цель лабораторной работы:

Разработка Android приложения, демонстрирующего возможности работы с базой данных SQLite.

Задачи лабораторной работы:

- создать приложение;
- настроить интерфейс приложения;
- реализовать логику приложения.

17.1 Введение

Для достижения цели, поставленной в лабораторной работе, сформулируем требования к разрабатываемому приложению. Приложение демонстрирует возможности работы с базой данных, предполагает реализацию следующих действий:

- добавление записей в базу данных;
- считывание строк и вывод на экран;
- удаление базы данных.

17.2 Создание приложения

Создадим новое Android приложение:

Project Name: Lab7_1-SQLite;

Package Name: com.example.lab7_1_sqlite;

Activity Files: SQLiteActivity.java, activity_sqlite.xml.

17.3 Настройка интерфейса

Разрабатываемое приложение является учебно-тренировочным, предполагает демонстрацию возможностей работы с базой данных: создание, добавление записей, просмотр записей, удаление базы данных. Поэтому интерфейс будет максимально простым: нам понадобятся два поля для ввода данных (в таблице базы данных два столбца), поле для вывода записей базы данных и три кнопки, по одной на каждое действие: добавление записей, вывод записей, удаление базы.

В нашем случае активность приложения имеет вид, показанный на рис. 17.1. Разумеется, не обязательно в точности воспроизводить предложенный внешний вид приложения, а на самом деле даже имеет смысл настроить интерфейс по своему усмотрению, чтобы потренировать навыки работы с компонентами и настройками пользовательского интерфейса.

Рис. 17.1. Внешний вид приложения для демонстрации основных возможностей базы данных SQLIte в системе Android

17.4 Реализация логики

Для создания и обновления базы данных, Android предоставляет класс SQLiteOpenHelper. Для работы с этим классом в приложении необходимо создать класс-наследник, в котором обязательно реализовать методы: onCreate() и onUpgarde().

Создадим новый класс, назовем его, например, MyOpenHelper:

```
public class MyOpenHelper extends SQLiteOpenHelper{...}
```

Реализуем метод onCreate():

```
public void onCreate(SQLiteDatabase DB) {
```

```
  String query="create table " + TABLE_NAME + " (_id integer primary key
autoincrement, " + field_name_1 + "
TEXT, " + field_name_2 + " TEXT)";
  DB.execSQL(query);
}
```

Параметром метода onCreate() является объект класса SQLiteDatabase, позволяющего работать с базой данных напрямую.

В строке query формируется запрос на создание таблицы, где

TABLE_NAME	- константа, содержащая имя таблицы,
FIELD_NAME_1	- константа, содержащая имя первого столбца,
FIELD_NAME_2	- константа, содержащая имя второго столбца.

Все эти константы объявлены в классе MyOpenHelper:

```
public String TABLE_NAME="first_table";
public String FIELD_NAME_1="first_field";
public String FIELD_NAME_2="second_field";
```

Метод execSQL() класса SQLiteDatabase запускает запрос на выполнение.

Еще необходимо реализовать метод onUpgrade(), в нашем случае он ничего существенного не делает, но необходимо прописать реализацию этого метода даже, если это будет пустая реализация:

```
public void onUpgrade(SQLiteDatabase DB, int oldVersion,
        int newVersion) {
  Log.d("myLogs","| Upgrade |"+DB.toString());
}
```

Система потребует создать конструктор класса MyOpenHelper, т. к. для его предка конструктор без параметров не определен, поэтому добавим в код класса следующие строчки:

```
MyOpenHelper(Context ct, String nm,
        SQLiteDatabase.CursorFactory cf, int vs){
  super(ct, nm, cf, vs);
}
```

Полученный конструктор не делает ничего особенного, просто вызывает конструктор предка, т. е. класса SQLiteOpenHelper.

Мы описали создание вспомогательного класса, необходимого нам для создания и открытия базы данных, полный код этого класса представлен в листинге 17.1.

Пришло время описать реализацию заявленных функций приложения. Далее будем работать с классом активности, описание которого находится в файле **SQLiteActivity.java**.

Выполним необходимую подготовку, зададим переменные, как поля класса активности:

```
EditText field1, field2, result;
MyOpenHelper myHelper = null;
SQLiteDatabase DB;
```

Вторая строка определяет объект класса MyOpenHelper, а третья - объект класса SQLiteDatabase, далее в программе все взаимодействия с базой данных после ее создания будут выполняться через этот объект.

Далее в методе onCreate() активности создадим экземпляр класса MyOpenHelper, для создания, открытия и возможно управления базой данных, имя которой myDB:

```
myHelper=new MyOpenHelper(this, "myDB", null, 1);
```

Для работы приложения необходимо задать действия, которые будут выполняться при нажатии на кнопки, для этого настроим свойство On Click кнопок:

- для кнопки **Ввод данных** свойству On Click присвоим значение insertIntoDatabase, при нажатии на кнопку будет вызываться метод с этим именем, реализованный в классе активности;
- для кнопки **Вывод данных** свойству On Click присвоим значение readDatabase, при нажатии на кнопку будет вызываться метод с этим именем, реализованный в классе активности;
- для кнопки **Delete** свойству On Click присвоим значение deleteDatabase, при нажатии на кнопку будет вызываться метод с этим именем, реализованный в классе активности.

Рассмотрим реализацию метода insertIntoDatabase().

В этом методе после проверки того, что в поля введены значения, получаем экземпляр базы данных с разрешением на запись:

```
DB = myHelper.getWritableDatabase();
```

Далее формируем запись, добавляемую в таблицу:

```
ContentValues CV = new ContentValues()
CV.put(myHelper.FIELD_NAME_1,field1.getText().toString());
CV.put(myHelper.FIELD_NAME_2,field2.getText().toString());
```

После этого добавляем запись в таблицу и закрываем экземпляр базы данных:

```
DB.insert(myHelper.TABLE_NAME,null,CV);
DB.close();
```

Полный код рассмотренного метода можно найти в листинге 17.2.

Рассмотрим реализацию метода readDatabase().

В этом методе сначала получаем экземпляр базы данных с разрешением на чтение:

```
DB = myHelper.getReadableDatabase();
```

Далее формируем и выполняем запрос к базе данных на получение всех значений из базы, результат запроса помещается в объект класса Cursor:

```
String columns[]={"_id",myHelper.FIELD_NAME_1,
        myHelper.FIELD_NAME_2};
Cursor cursor=DB.query(myHelper.TABLE_NAME, columns, null,
        null, null, null, "_id");
```

После этого выведем содержимое класса Cursor в поле на форме, предназначенное для вывода информации:

```
if(cursor!=null){
  cursor.moveToFirst();
  if (cursor.moveToFirst()) {
    do {
      result.setText(result.getText().toString()+"\n" +cursor.getString(0)+")
"+cursor.getString(1)+",
      "+cursor.getString(2));
    } while (cursor.moveToNext());
  }
}
```

В этом кусочке кода основное внимание можно обратить на работу с классом курсор, очень похоже на работу со списками. Начиная с начала списка получаем значения элементов и передвигаемся на следующий элемент, пока не достигнем конца.

После всех манипуляций с базой данных необходимо ее закрыть:

```
DB.close();
```

Полный код рассмотренного метода можно найти в листинге 17.2.

Рассмотрим реализацию метода deleteDatabase.

В этом методе сначала получаем экземпляр базы данных с разрешением на запись:

```
DB = myHelper.getWritableDatabase();
```

Удалим таблицу, с помощью метода delete() класса SQLiteDatabase:

```
DB.delete(myHelper.TABLE_NAME, null, null);
```

После всего закроем базу:

```
DB.close();
```

Полный код рассмотренного метода можно найти в листинге 17.2.

17.5 Заключение

В работе на примере простого приложения рассмотрели выполнение основных операций с базами данных: создание, добавление записей, просмотр всех записей таблицы, удаление. Для выполнения выборок из таблицы, необходимо познакомиться с основами построения запросов в SQLite и сформировать запрос для метода query() класса SQLiteDatabase.

```
package com.example.lab7_1_sqlite;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;

public class MyOpenHelper extends SQLiteOpenHelper {

  public String TABLE_NAME="first_table";
  public String FIELD_NAME_1="first_field";
  public String FIELD_NAME_2="second_field";

  MyOpenHelper(Context ct, String nm, SQLiteDatabase.CursorFactory cf, int
vs){
    super(ct, nm, cf, vs);
  }

  @Override
  public void onUpgrade(SQLiteDatabase DB, int oldVersion, int newVersion) {
    Log.d("myLogs","| Upgrade |"+DB.toString());
  }

  @Override
  public void onCreate(SQLiteDatabase DB) {
    Log.d("myLogs","| Create |"+DB.toString());
    String query="create table " + TABLE_NAME + " ( _id integer primary key
      autoincrement, " + FIELD_NAME_1 + " TEXT, " + FIELD_NAME_2 + " TEXT)";
    DB.execSQL(query);
  }
}
```

Листинг 17.1. Класс MyOpenHelper

```
package com.example.lab7_1_sqlite;

import android.os.Bundle;
import android.app.Activity;
import android.content.ContentValues;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.util.Log;
import android.view.View;
import android.widget.EditText;

public class SQLiteActivity extends Activity {

  MyOpenHelper myHelper = null;
  EditText field1, field2, result;
  SQLiteDatabase DB;

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

```
    myHelper=new MyOpenHelper(this, "myDB", null, 1);

    field1=(EditText)findViewById(R.id.field1);
    field2=(EditText)findViewById(R.id.field2);
    result=(EditText)findViewById(R.id.dbResult);
  }

  public void insertIntoDatabase(View v){

    if(!field1.getText().toString().equals("") &&
        !field2.getText().toString().equals("")){
      Log.d("myLogs","Insert INTO DB ("+field1.getText().toString()+ "," +
field2.getText().toString()+")");

      DB = myHelper.getWritableDatabase();
      String query="create table if not exist " + myHelper.TABLE_NAME +
      " (_id integer primary key autoincrement, " + myHelper.FIELD_NAME_1 + "
TEXT, " + myHelper.FIELD_NAME_2 + " TEXT)";

      ContentValues CV = new ContentValues();
      CV.put(myHelper.FIELD_NAME_1,field1.getText().toString());
      CV.put(myHelper.FIELD_NAME_2,field2.getText().toString());
      DB.insert(myHelper.TABLE_NAME,null,CV);
      DB.close();
      field1.setText("");
      field2.setText("");
    }
  }

  public void readDatabase(View v){

    result.setText("");

    Log.d("myLogs","READ FROM DB");
    DB = myHelper.getReadableDatabase();

    String columns[]={"_id",myHelper.FIELD_NAME_1, myHelper.FIELD_NAME_2};
    Cursor cursor=DB.query(myHelper.TABLE_NAME, columns, null, null, null,
    null, "_id");
    if(cursor!=null){
      cursor.moveToFirst();
      if (cursor.moveToFirst()) {
        do {
          result.setText(result.getText().toString()+ "\n" +
                cursor.getString(0) + ") " + cursor.getString(1) + "," +
cursor.getString(2));
        } while (cursor.moveToNext());
      }
    }

    DB.close();
  }

  public void deleteDatabase(View v){
    Log.d("myLogs","Delete Database");
    DB = myHelper.getWritableDatabase();

    DB.delete(myHelper.TABLE_NAME, null, null);
    DB.close();
  }
}
```

Листинг 17.2. Класс SQLiteActivity

Внимание! Если Вы увидите ошибку на нашем сайте, выделите её и нажмите Ctrl+Enter.
© Национальный Открытый Университет "ИНТУИТ", 2016 | www.intuit.ru

Самостоятельная работа 11. Установка и настройка среды программирования Intel XDK. Создание первых приложений

Цель лабораторной работы:

Установка и настройка среды Intel XDK и создание простейших приложений на основе пустого шаблона и встроенных примеров.

Задачи лабораторной работы:

- Установить и настроить среду программирования Intel XDK.
- Создать первое приложение Hello, world!.
- Создать приложение "Часы".
- Рассмотреть работу приложения "Orientation" из встроенных примеров.
- Научиться запускать приложение на эмуляторе мобильного устройства.
- Научиться осуществлять тестирование приложения на мобильном устройстве, собирать и компилировать apk-файлы.

19.1 Введение

Лабораторная работа посвящена различным аспектам разработки мобильных приложений для ОС Android с использованием среды Intel XDK. В ней содержатся указания по установке и настройке среды, созданию приложений как "с нуля", так и из готовых шаблонов, запуску приложений в эмуляторе и на мобильных устройствах. Имеются задания для самостоятельной работы.

19.2 Установка и настройка среды

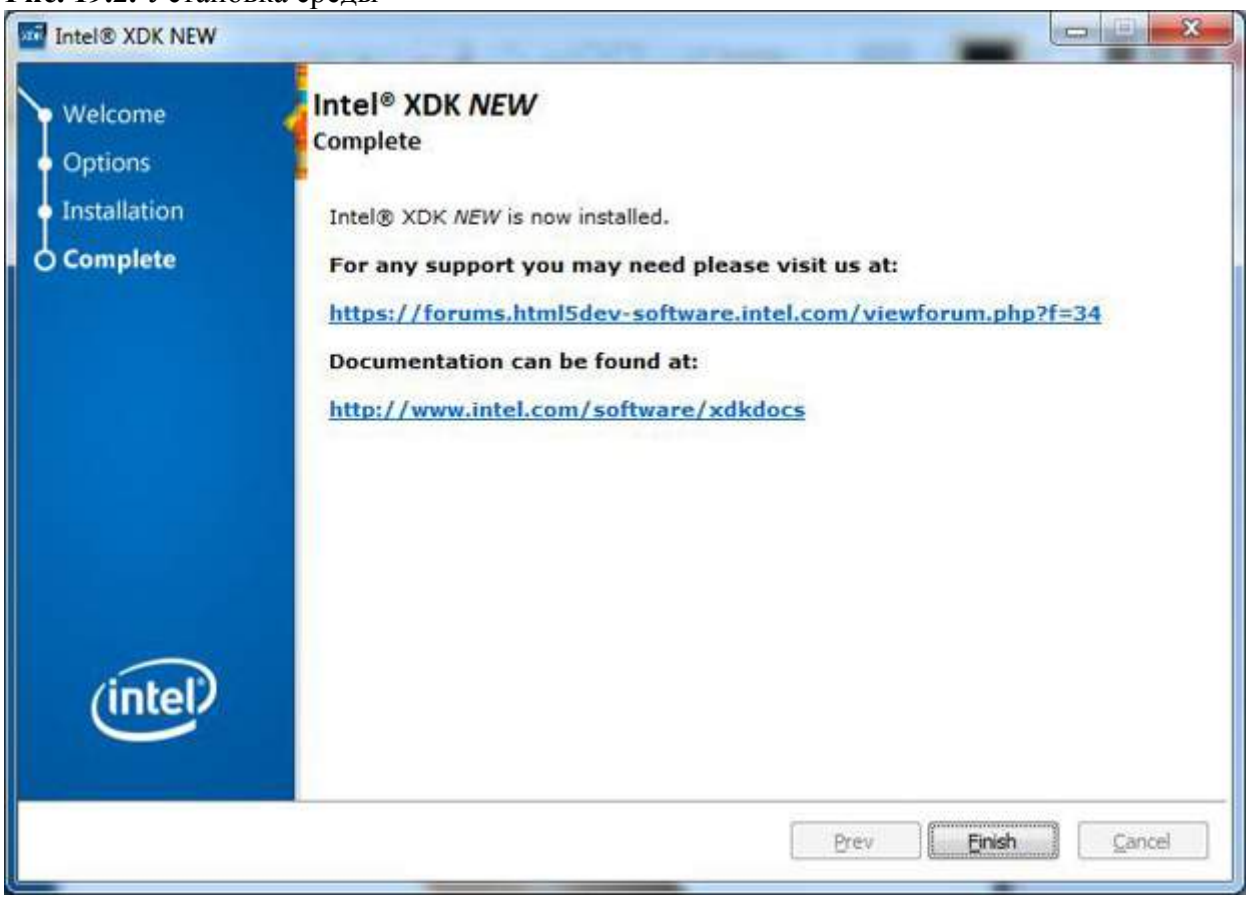
Скачать среду можно на сайте разработчика http://xdk-software.intel.com/ (рис. 19.1). Чтобы приступить к загрузке, достаточно выбрать "**Download Intel XDK NEW**". Вы будете переведены на страницу скачивания дистрибутива для Вашей операционной системы, например, Intel® XDK download for Microsoft Windows, где через 5 секунд начнется загрузка.

увеличить изображение
Рис. 19.1. Сайт разработчика

Установка среды разработки не представляет собой ничего сложного - типичный набор действий для установки приложения (рис. 19.2). После завершения установки инсталлятор предложит вам две гиперссылки на сайты, где можно найти документацию по среде и задать свои вопросы по работе с Intel XDK (рис. 19.3).

увеличить изображение

Рис. 19.2. Установка среды



увеличить изображение
Рис. 19.3. Поддержка пользователей

Итак, среда Intel XDK установлена, ярлык на нее добавлен в главное меню операционной системы (рис. 19.4).

При запуске приложение предложит авторизоваться (рис. 19.5). Если учетной записи Intel XDK еще нет, среда предложит пройти стандартную процедуру регистрации (рис. 19.6). При следующем запуске окна авторизации достаточно будет выбрать пункт "**Keep me logged in on this computer**" для автоматической авторизации на данном компьютере (рис. 19.4).

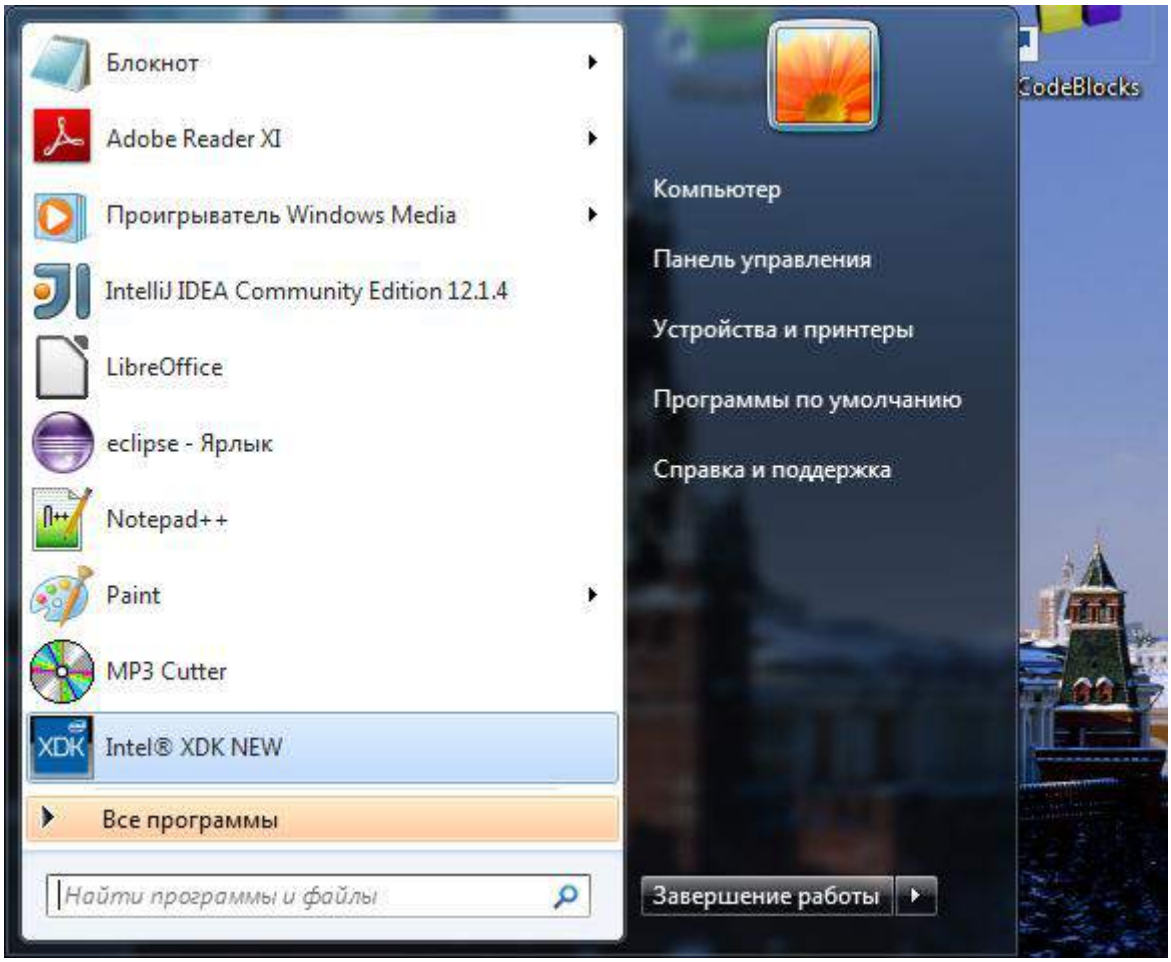
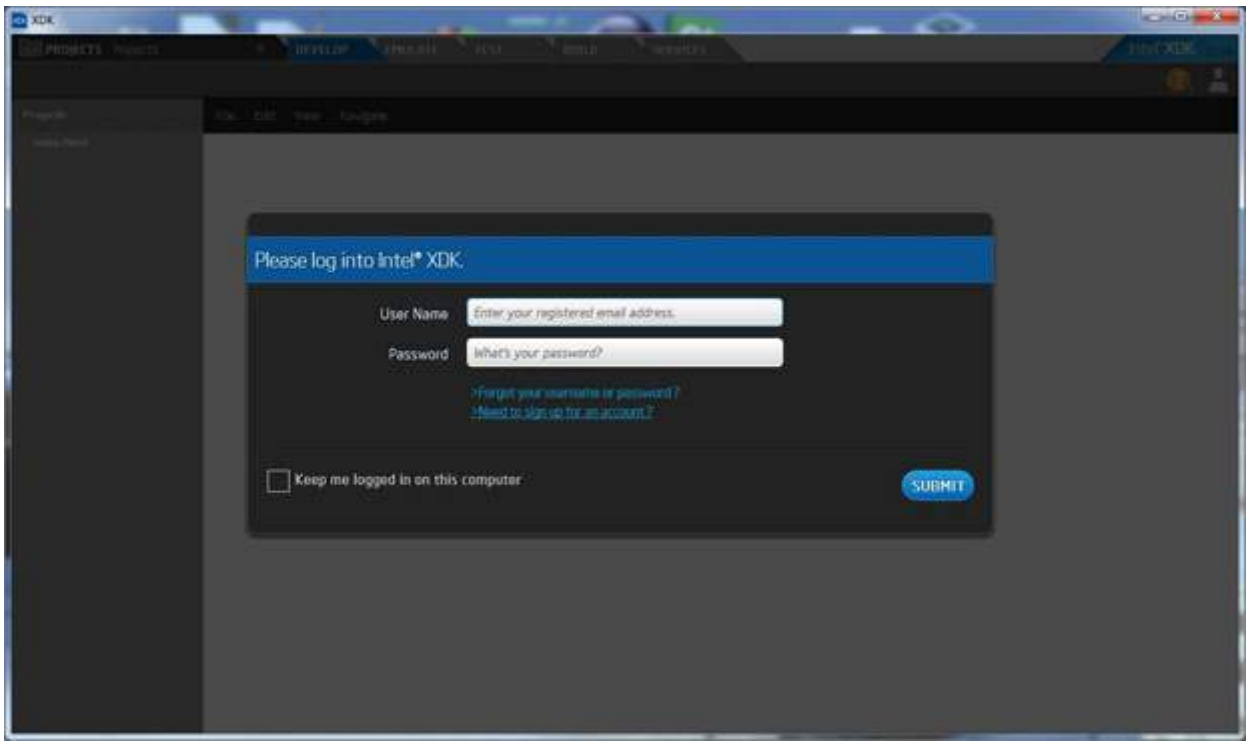
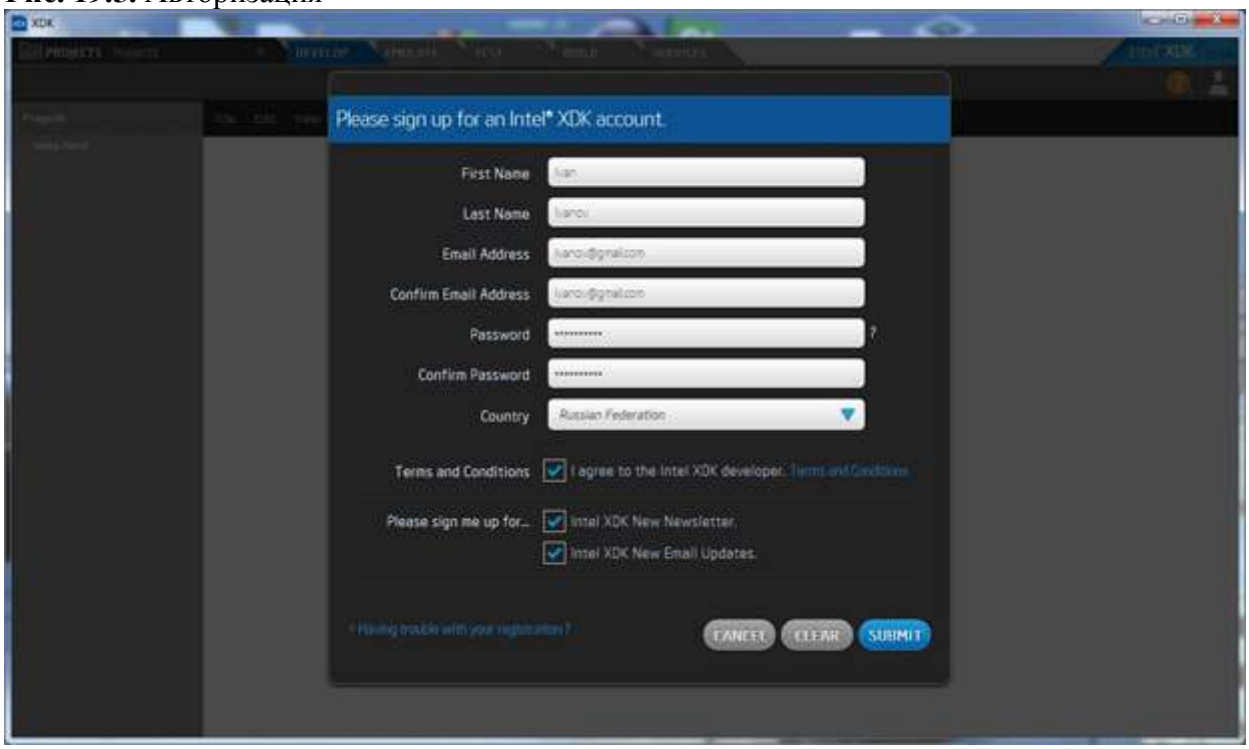


Рис. 19.4. Intel XDK в меню пуск WIN7

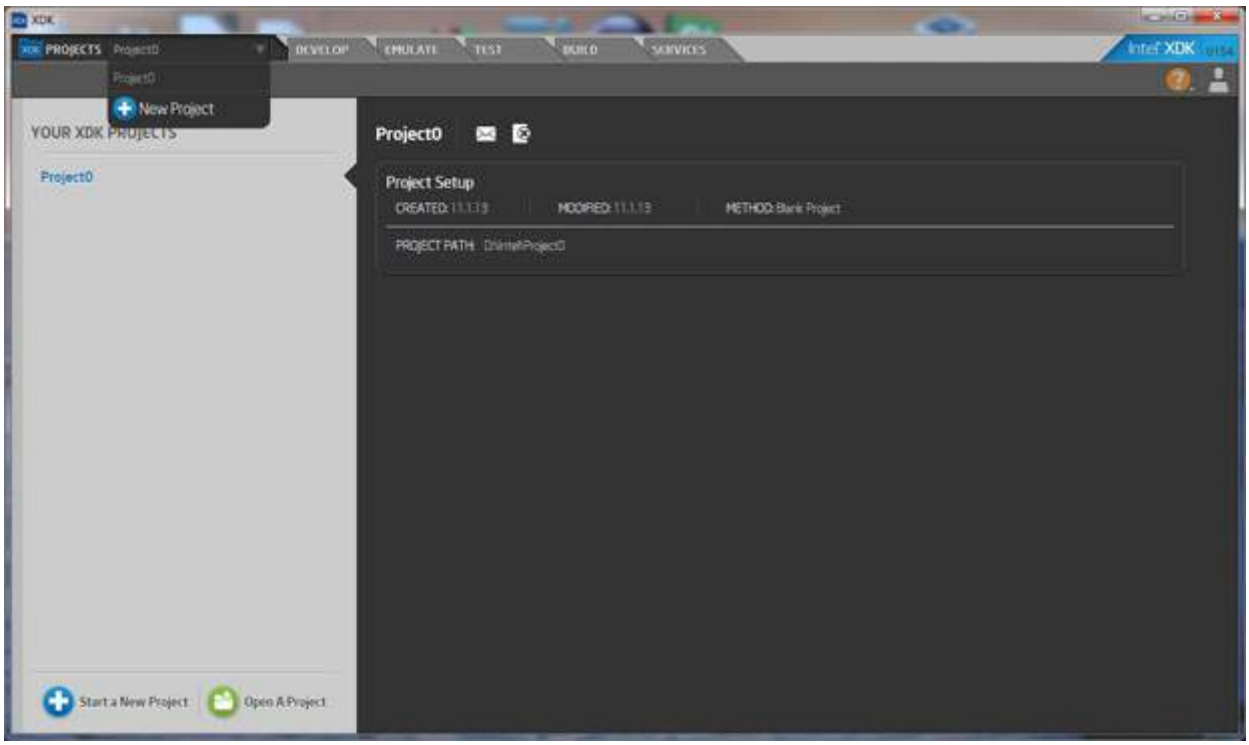


увеличить изображение
Рис. 19.5. Авторизация



увеличить изображение
Рис. 19.6. Регистрация

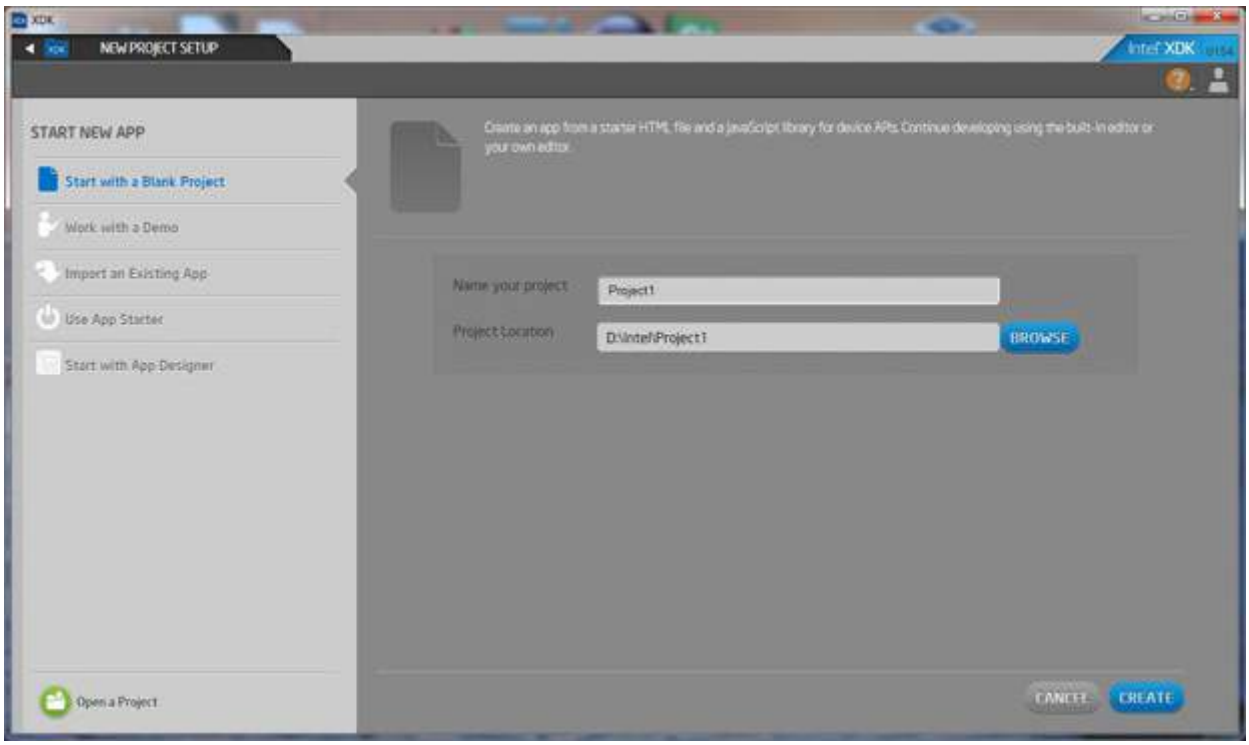
После авторизации мы увидим основное окно среды, проекты, над которыми уже велась работа, с указанием даты создания и модификации, а также места их расположения, и пункты меню для работы в панели управления (рис. 19.7).



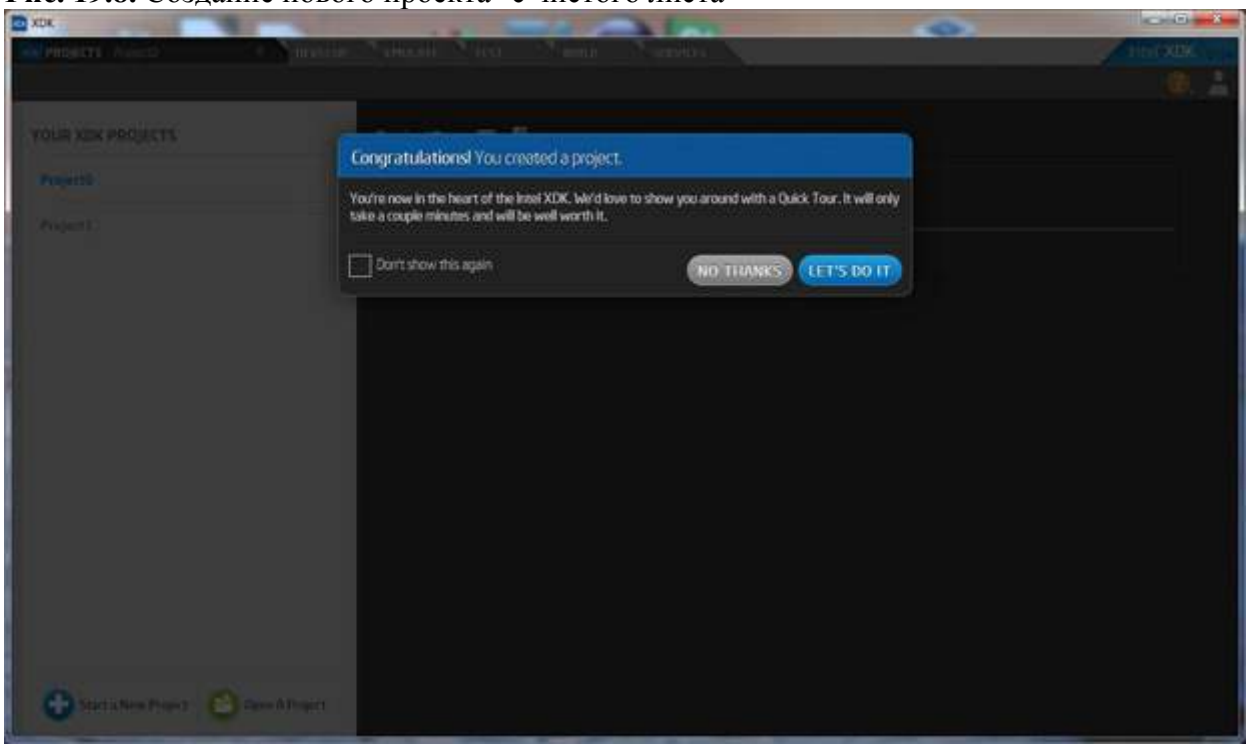
[увеличить изображение]
Рис. 19.7. Главное окно среды

19.3 Создание проекта на основе пустого шаблона

Выбрав пункт "**New Project**", мы попадаем в окно для выбора вида создаваемого приложения. Здесь мы можем создать приложения "с чистого листа": стартового шаблона HTML-файла и сценария Javascript. В этом случае достаточно лишь указать имя создаваемого проекта и директорию для его хранения (рис. 19.8). Система дает советы по работе с создаваемым приложением (рис. 19.9).

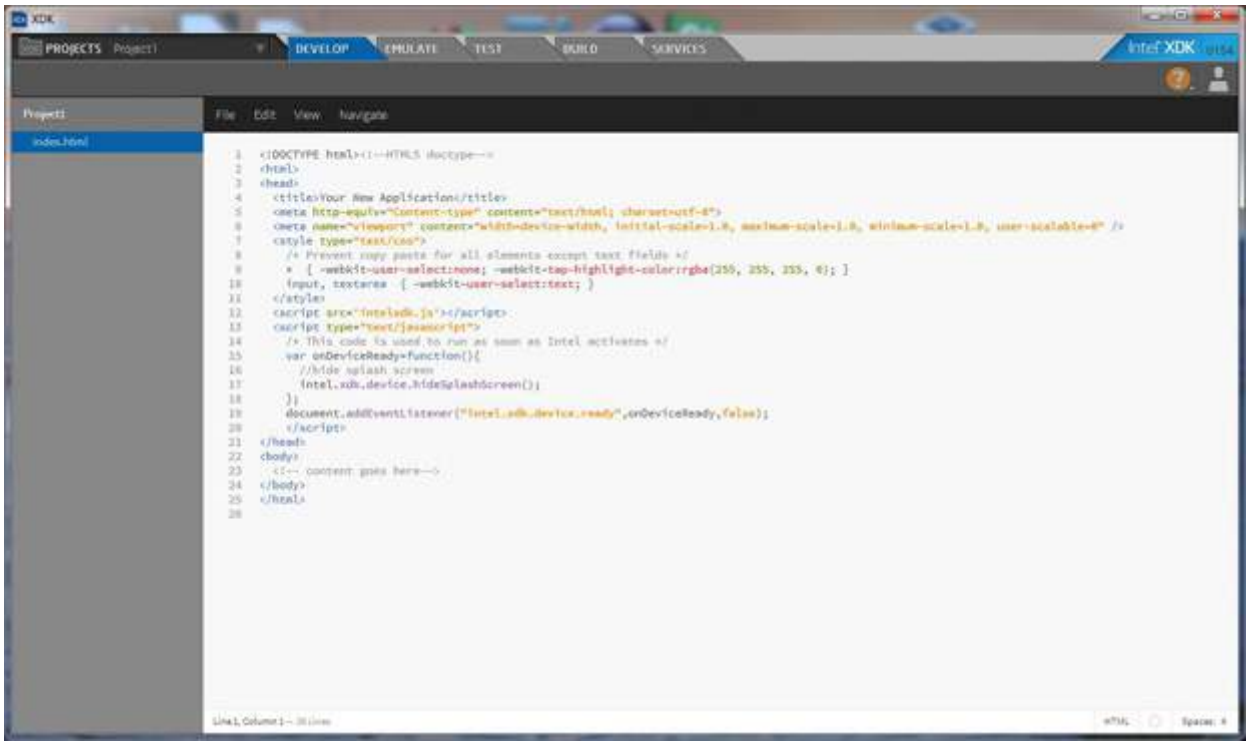


увеличить изображение
Рис. 19.8. Создание нового проекта "с чистого листа"



увеличить изображение
Рис. 19.9. Подсказки среды

После этого мы попадаем во вкладку "**Develop**", где можем видеть заготовку нашего будущего приложения (рис. 19.10).



увеличить изображение
Рис. 19.10. Код заготовки будущего проекта

Рассмотрим этот код подробнее. Это практически стандартная web-страница с подключенным скриптом intelxdk.js, отвечающим за основную функциональность приложения.

Для того чтобы web-страница функционировала как мобильное приложение, по умолчанию задан ряд настроек.

Так, сделаны мета-определения для корректной работы в браузере телефона:

```
<meta name="viewport" content="width=device-width,
initial-scale=1.0">
```

Данный код определяет ширину страницы (в данном случае приложения), равной ширине устройства, и задает начальный масштаб.

Кроме того, поставлен запрет на любое масштабирование - как начальное при загрузке, так и пользовательское - уже после загрузки (maximum-scale=1.0, minimum-scale=1.0, user-scalable=0).

С помощью свойств каскадных таблиц стилей реализован запрет на выделение любых компонентов, за исключением элементов ввода и редактирования текстовых полей:

```
-webkit-user-select:none; и -webkit-user-select:text;
```

За запуск всех функциональных возможностей, реализованных в скрипте intelxdk.js, отвечают строки, реализованные во втором скрипте, явно прописанном в коде приложения:

```
/* This code is used to run as soon as Intel activates */
        var onDeviceReady=function(){
        //hide splash screen
        intel.xdk.device.hideSplashScreen();
        };
document.addEventListener("intel.xdk.device.ready",onDeviceReady,false);
```

Программирование на JavaScript - программирование каких-либо действий, основанных на событиях. Модель событий DOM2, реализованная в виде иерархической структуры объектов документа HTML, позволяет реализовать стандартные способы создания, захвата, обработки и отмены событий в древовидной структуре типа иерархии.

События начинают жизненный цикл на вершине иерархии (на уровне Document) и движутся вниз через вложенные объекты к цели. Этот процесс называется фазой захвата.

За это в DOM2 отвечает метод addEventListener с тремя аргументами: строковое название события, у нас это intel.xdk.device.ready, обработчик события - onDeviceReady, и логический параметр - фаза захвата. Если значение фазы равно true - то обработчик сработает по мере перемещения вглубь по иерархии, если false - на обратном "пути".

Как только Javascript будет загружен, событие intel.xdk.device.ready будет обработано.

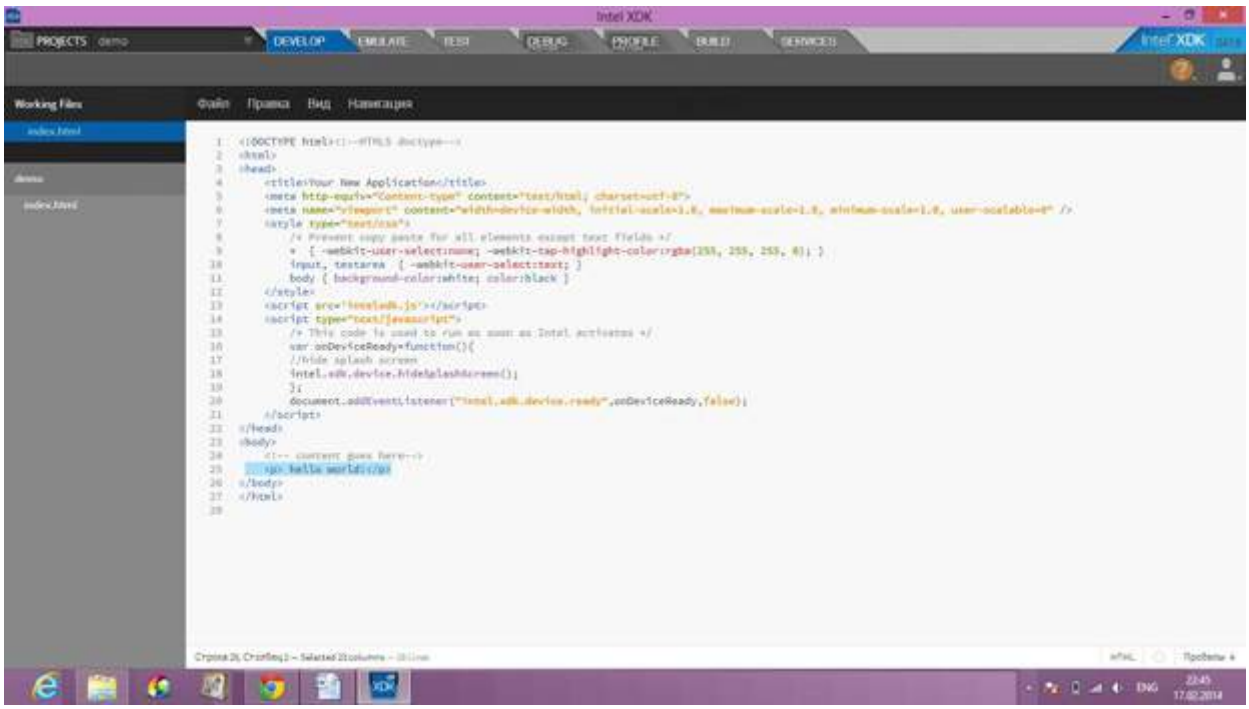
Итак, заготовка приложения есть, можно приступать к разработке пользовательских приложений.

19.4 Разработка первого приложения

В основе мобильного приложения, разрабатываемого с использованием Intel XDK, как мы уже выяснили ранее, обычная web-страница с таблицей стилей и скриптами javascript для реализации реакций на события. А значит, все наши знания по HTML5, CSS и javascript вполне применимы.

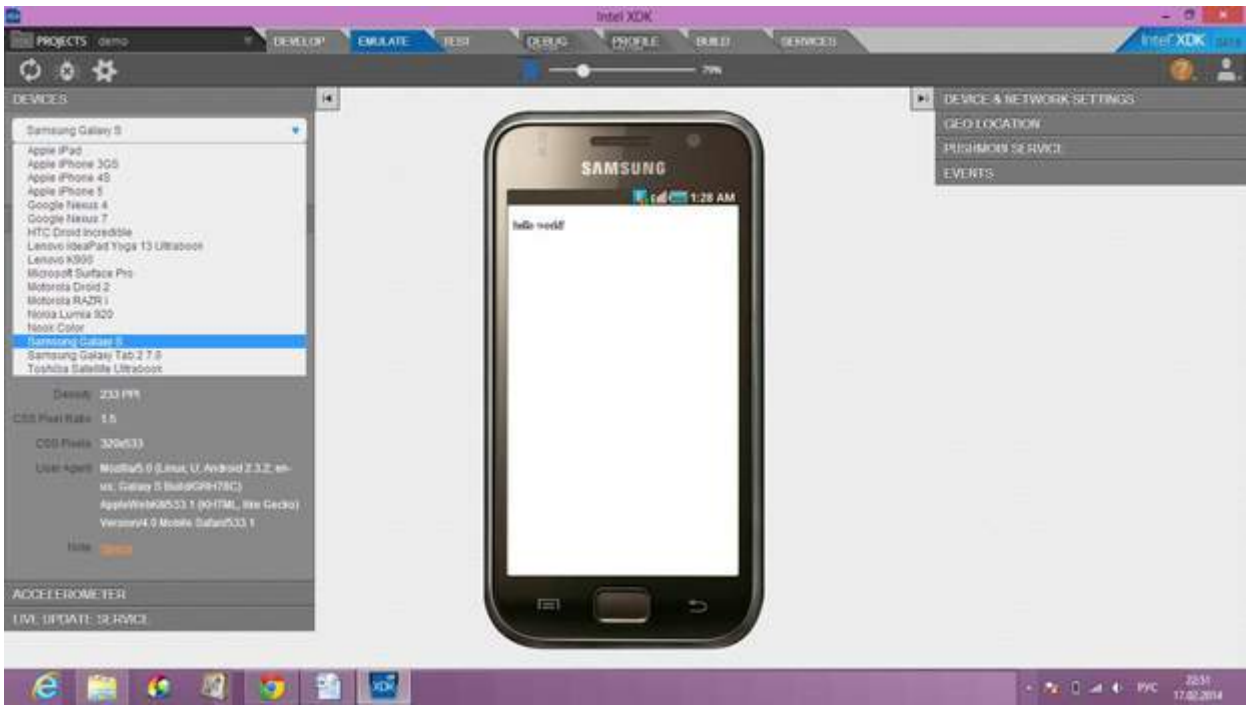
Начнем с классического "**Hello world!**". Для этого вполне достаточно прописать в теле страницы, в данном случае приложения, например, абзац текста (рис. 19.11):

```
<p> Hello world!</p>
```



увеличить изображение
Рис. 19.11. Код приложения "Hello world!"

Проверить полученный результат можно с помощью вкладки "**Emulate**", где среда позволяет сымитировать работу на самых популярных устройствах (рис. 19.12).



увеличить изображение
Рис. 19.12. Простейшее приложение на эмуляторе

Всем, кто знаком с тройкой **HTML+CSS+Javascript**, открыты огромные просторы для творчества!

Например, почему бы не создать свои "авторские" часы? Отображаться время будет с помощью заголовка 1 уровня, которому мы зададим размер, цвет и выравнивание по центру:

```
<h1 id="times"></h1>
```

Стиль прописан в таблице стилей

```
#times {color:red;font-size:50pt;text-align:center;}
```

За "автоматику" часов будет отвечать две функции Javascript.

Функция startTime получает текущее системное время и задает его элементу с идентификатором times.

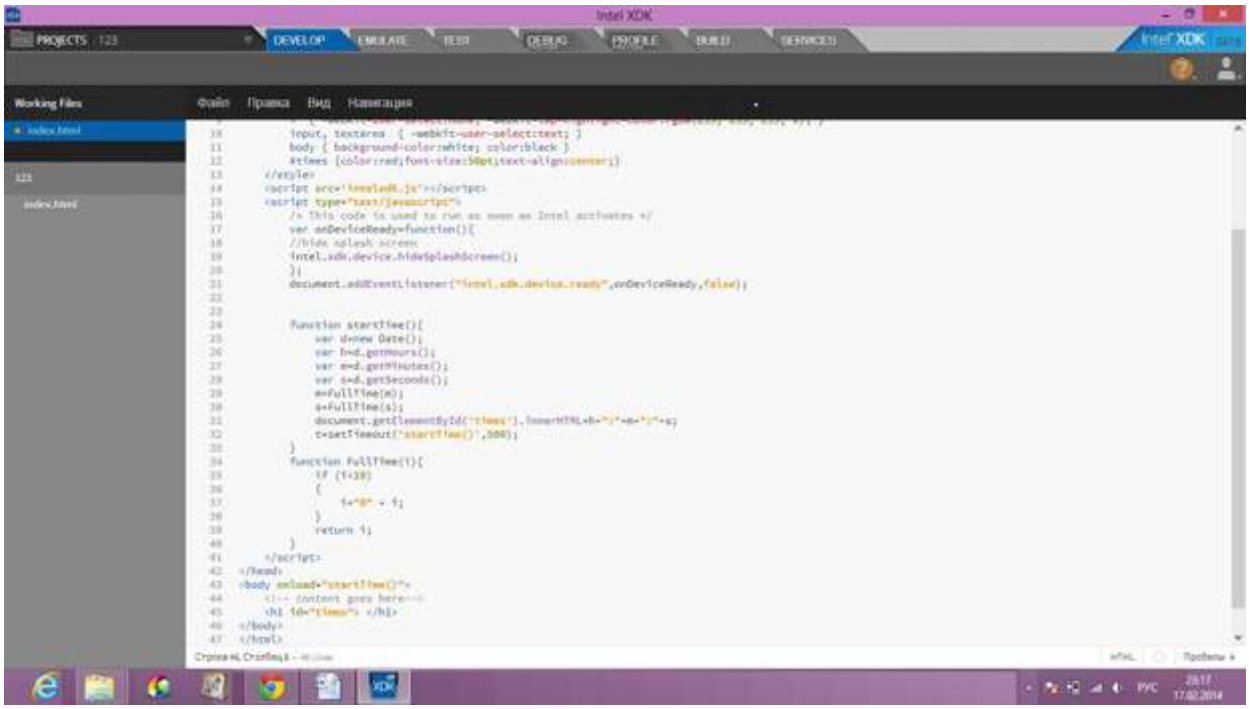
```
function startTime(){
            var d=new Date();
            var h=d.getHours();
            var m=d.getMinutes();
            var s=d.getSeconds();
            m=FullTime(m);
            s=FullTime(s);
            document.getElementById('times').innerHTML=h+":"+m+":"+s;
t=setTimeout('startTime()',500);
        }
```

Функция FullTime позволяет корректно отобразить время, значение минут и секунд которого занимает 1 разряд (добавляет ведущий нуль)

```
function FullTime(i){
        if (i<10)
        {
                i="0" + i;
        }
        return i;
}
```

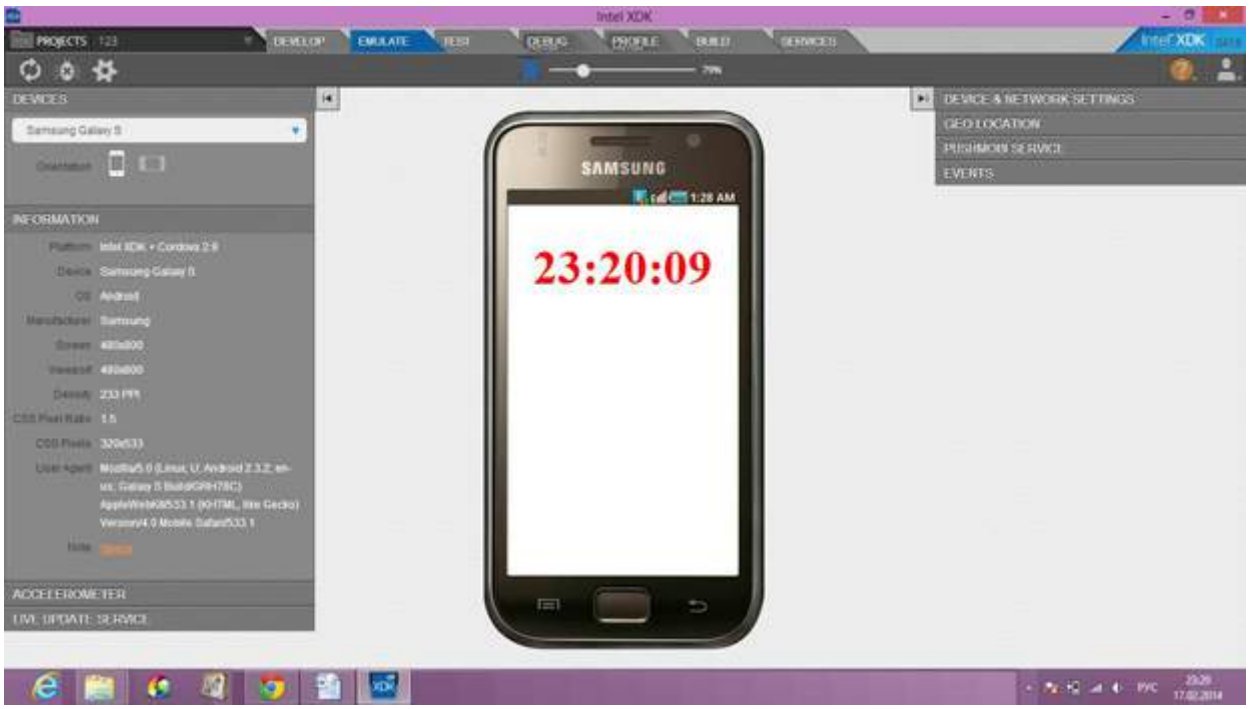
Остается добавить эти функции в тег <script type="text/javascript"> и прописать, чтобы функция была выполнена при загрузке приложения (рис. 19.13).

```
<body onload="startTime()">
```



увеличить изображение
Рис. 19.13. Код приложения "Часы"

Проверяем результат работы (рис. 19.14)



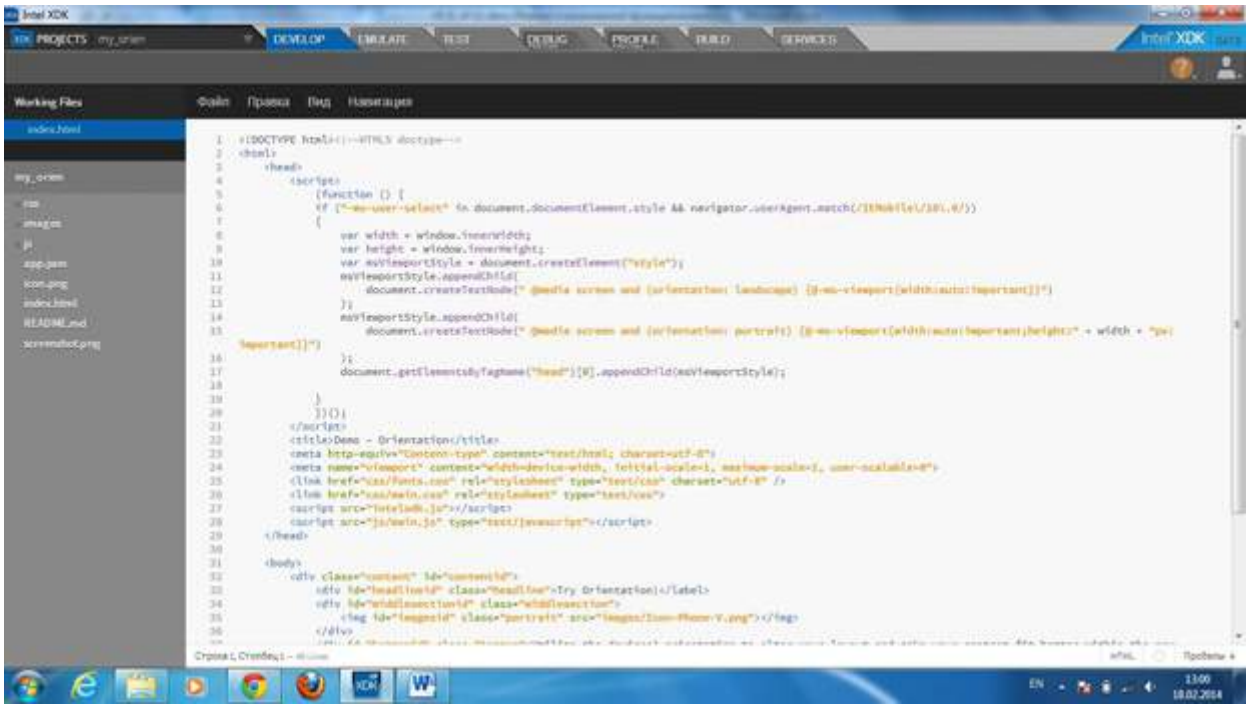
[увеличить изображение]
Рис. 19.14. Приложение "Часы" на эмуляторе

19.5 Разработка приложения на основе шаблона

Разрабатывать приложения можно не только на основе пустого шаблона, но и воспользоваться встроенными примерами и создать на их базе собственное. Для этого достаточно в окне создания приложения выбрать пункт "**Work with a Demo**" выбрать понравившийся пример (рис. 19.15).

[увеличить изображение]
Рис. 19.15. Выбор встроенного примера

Рассмотрим, например, приложение "**Orientation**". Для этого выберем его в списке и нажмем на кнопку "**Next**". Зададим имя нашему проекту и нажмем кнопку "**Create**". Получим следующий код (рис. 19.16). Рассмотрим его поподробнее.



увеличить изображение
Рис. 19.16. Код приложения "Orientation"

Для начала заметим, что автоматически были созданы 3 директории: для таблиц стилей, для изображений и для скриптов, а также ряд вспомогательных файлов.

Остановимся подробнее на скрипте **main.js**. Остальные два особого интереса не представляют: intelxdk.js нам уже знаком, встроенный скрипт отвечает за начальные соотношения ширины и длины, настройки портретной и альбомной ориентации.

Итак, функция init начинает свою работу в момент, когда приложение уже загружено, а библиотека JavaScript еще нет и передает значения ширины и высоты для встроенного скрипта.

Далее прописан запрет на изменения размера приложения (скроллинг) как реакция на событие touchmove.

```
preventDefaultScroll = function(event)
{
    // Prevent scrolling on this element
    event.preventDefault();
    window.scroll(0,0);
    return false;
};
window.document.addEventListener("touchmove", preventDefaultScroll, false);
/*
```

Как только все библиотеки Javascript, в том числе Intel XDK, загружены, возникает событие intel.xdk.device.ready. В ее обработчике onDeviceReady() помимо описанных в

"чистом" шаблоне команд по скрытию окна заставки прописана фиксация текущей ориентации (intel.xdk.device.setRotateOrientation("any")) и возможность отслеживания поворотов (intel.xdk.device.setAutoRotate(true)). Команда intel.xdk.device.managePower(true,false) не дает приложению перейти в спящий режим.

```
function onDeviceReady()
{
    //lock orientation
    intel.xdk.device.setRotateOrientation("any");
    intel.xdk.device.setAutoRotate(true);

    //manage power
    intel.xdk.device.managePower(true,false);

    //hide splash screen
    intel.xdk.device.hideSplashScreen();
}
```

Теперь можно заняться отслеживанием события "**Поворот**" и изменением свойств изображения imagesid и трех текстовых элементов-слоев headlineid, bottomed, endingid. За это у нас будет отвечать обработчик intel.xdk.device.orientation.change.

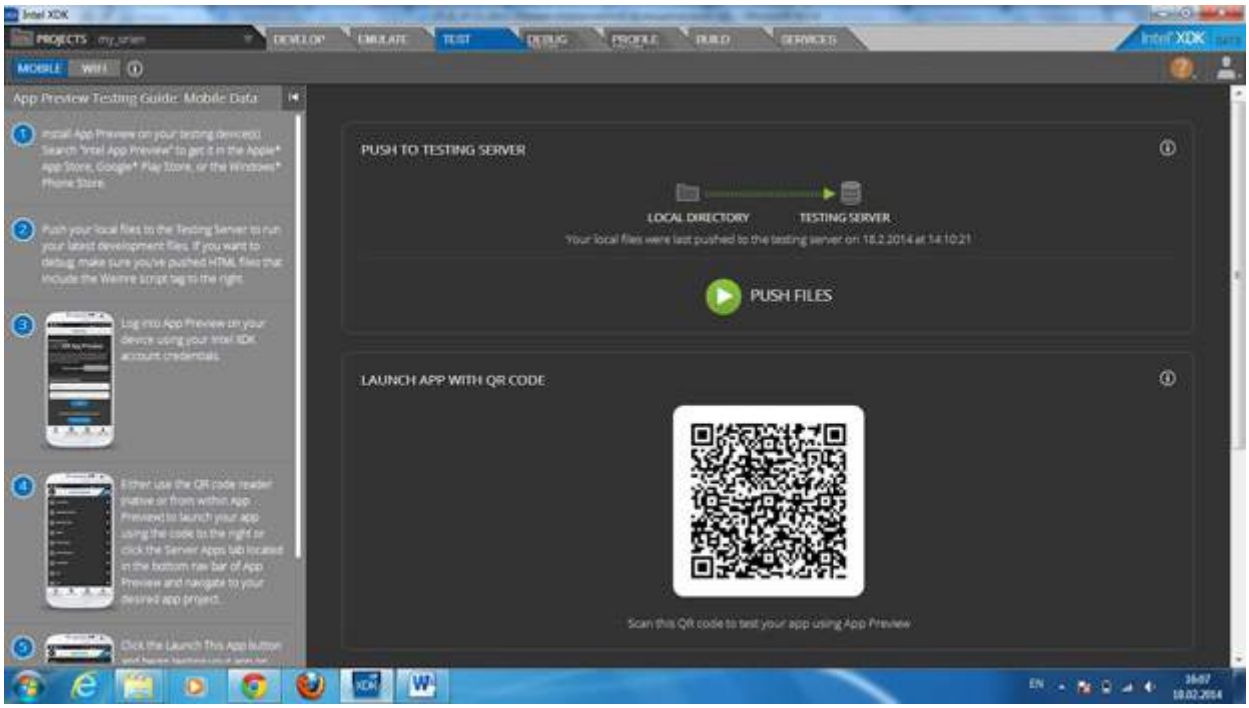
За портретную ориентацию отвечают значения 0 и 180, за альбомную 90 и 270 в соответствии со значением градусов поворота. Остается в зависимости от текущего значения ориентации задать соответствующие адреса изображений для imagesid и имена классов для объектов imagesid, headlineid, bottomed, endingid - одни для альбомной, другие для портретной. Все стилевые свойства прописаны классами в подключенной таблице стилей **main.css**.

```
function orientationchange(value)
{
    //Portrait orientation
    if(value.orientation == 0 || value.orientation == 180)
    {
document.getElementById("imagesid").className="portrait";
document.getElementById("imagesid").src="./images/Icon-Phone-V.png";
document.getElementById("headlineid").className="headline";
document.getElementById("bottomid").className="bottom";
document.getElementById("endingid").className="ending";
    }
    //Landscape orientation
    else
    {
document.getElementById("imagesid").src="./images/Icon-Phone-H.png";
document.getElementById("imagesid").className="imglandscape";
document.getElementById("headlineid").className="headline headlinelandscape";
document.getElementById("bottomid").className="bottom bottomlandscape";
document.getElementById("endingid").className="ending endinglandscape";
    }
}
```

Проект готов, проверен в режиме эмуляции, можно приступать к его тестированию и отладке, сборке и компиляции.

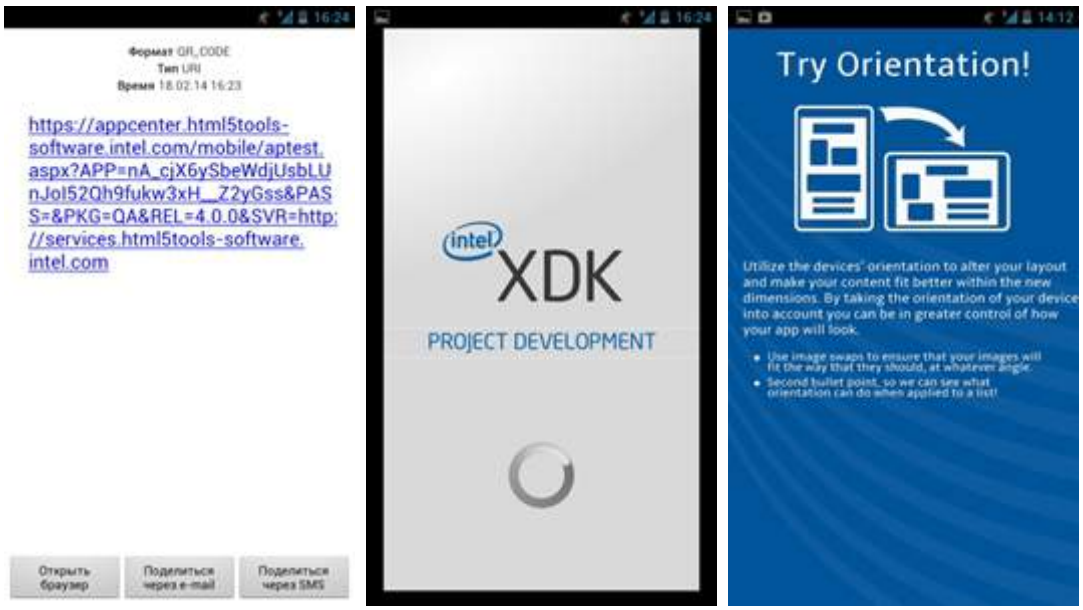
19.6 Тестирование и отладка на мобильном устройстве

Для реализации тестирования и отладки на мобильном устройстве необходимо установить на последнем приложение App Preview. Скачать его можно с помощью Apple AppStore, Google Play Store, Windows Phone Store. Также необходимо наличие на мобильном устройстве приложения по чтению QR-кодов. Остается перейти во вкладку "**Test**", нажать "**Push files**" для отправки файлов на сервер intel XDK и прочитать QR-код мобильным устройством (рис. 19.17).



увеличить изображение
Рис. 19.17. Код приложения "Orientation"

Код распознает гиперссылку, ресурс скачивается и автоматически запускается программой App Preview. В зависимости от скорости Интернет-соединения это займет некоторое время. После это приложение будет запущено. Можно экспериментировать (рис. 19.18).

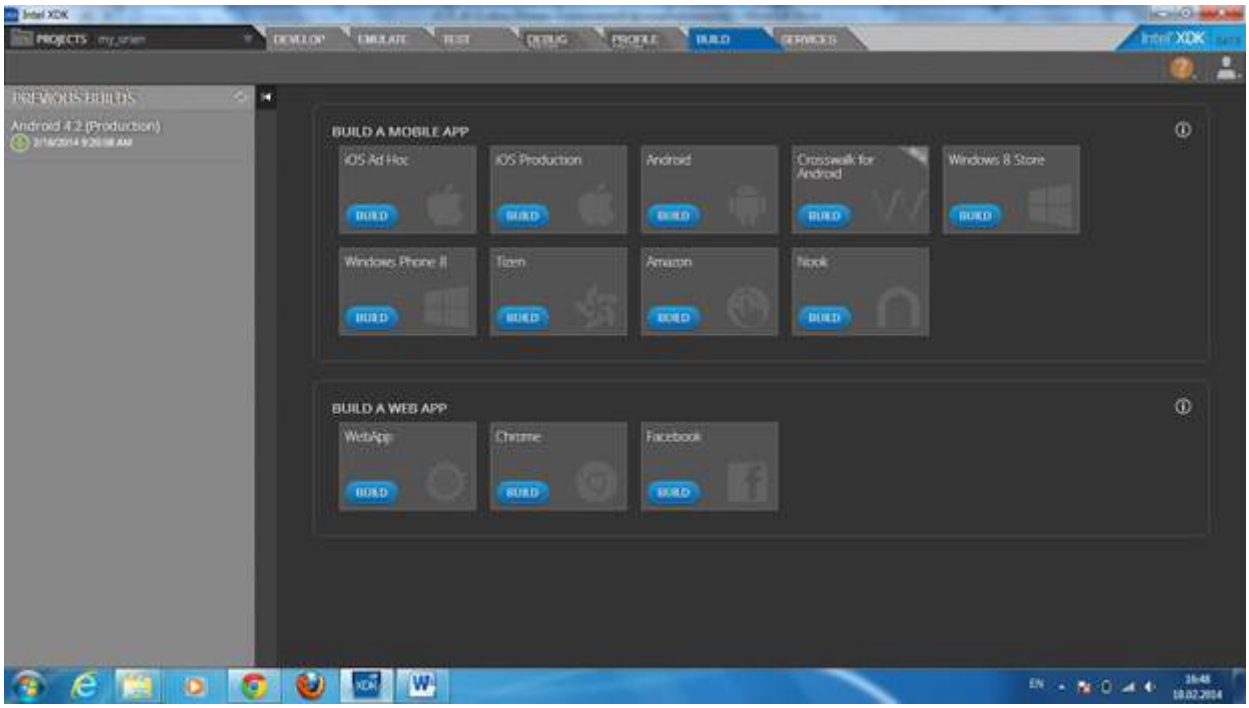


увеличить изображение увеличить изображение увеличить изображение

Рис 19.18. Запуск приложения на мобильном устройстве

19.7 Сборка и компиляция приложения

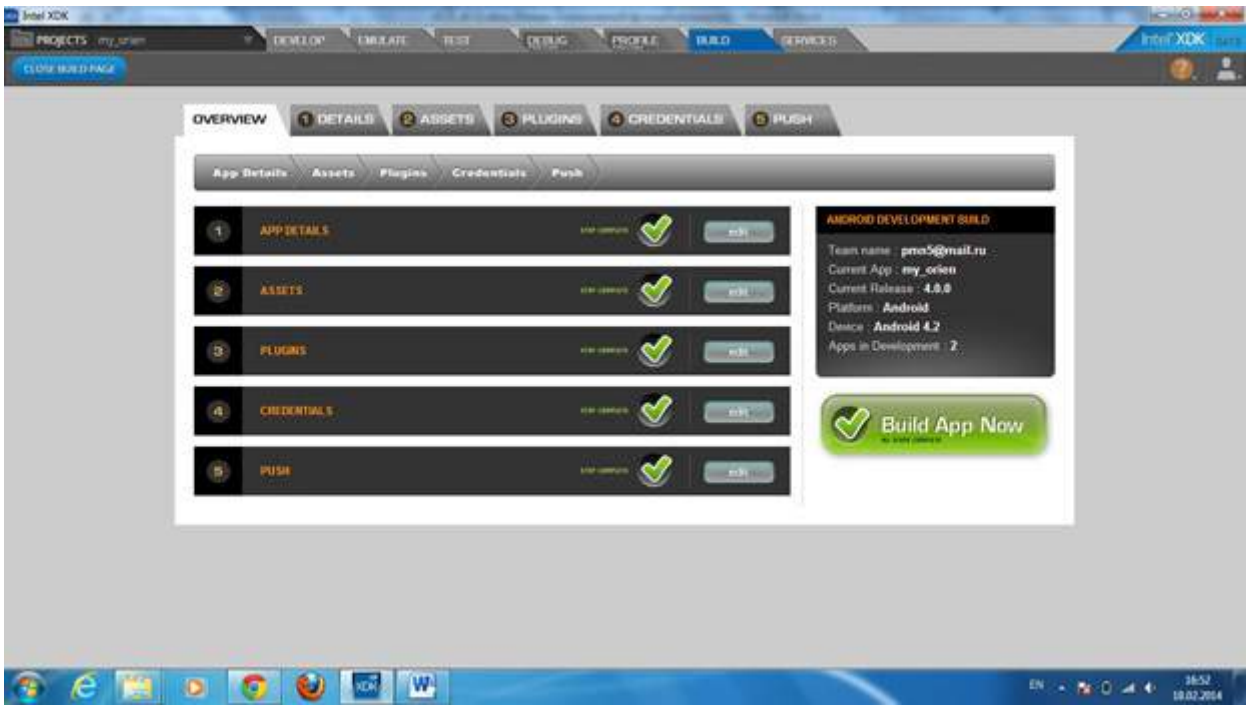
Сборка и компиляция приложения предполагает сборку apk-файла. Для этого переходим во вкладку "**Build**" и выбираем тип операционной системы на нашем мобильном устройстве (рис. 19.19).



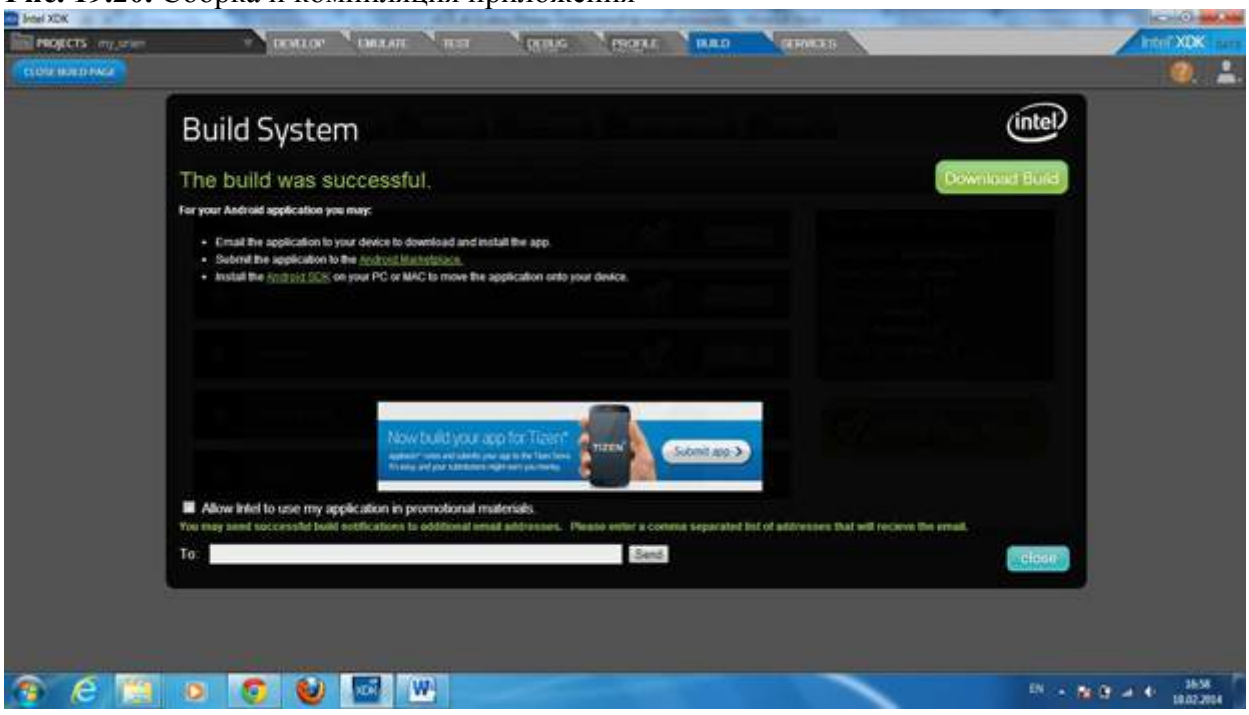
увеличить изображение

Рис. 19.19. Выбор ОС мобильного устройства

Нажимаем кнопку "**Build App Now**" и начинаем процесс сборки приложения. Здесь существует возможность тонкой настройки создаваемого приложения, ограничимся настройками по умолчанию (рис. 19.20). После завершения процесса сборки и компиляции скачиваем полученный apk-файл на мобильное устройство любым из способов (рис. 19.21).



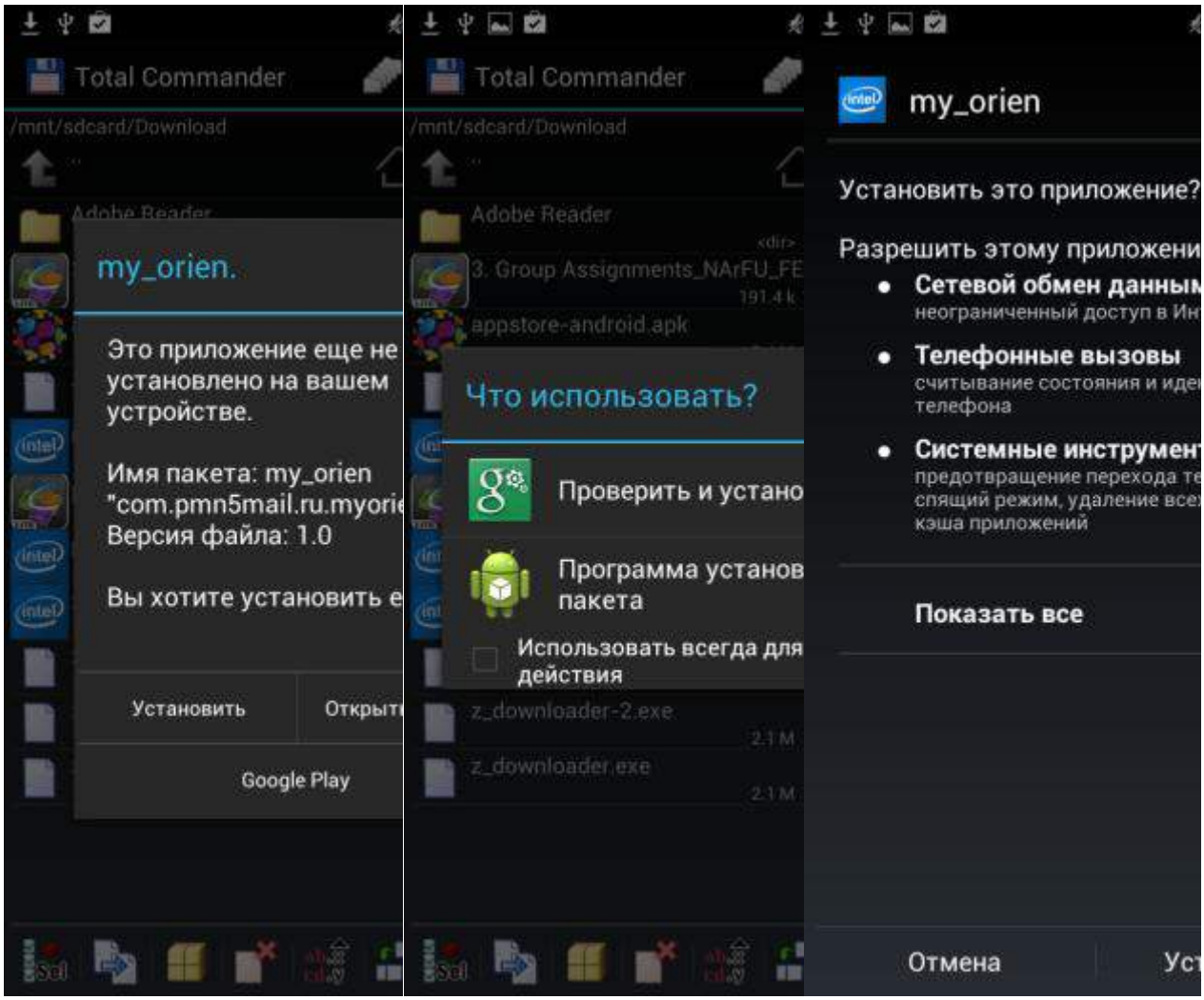
[увеличить изображение]
Рис. 19.20. Сборка и компиляция приложения



[увеличить изображение]
Рис. 19.21. Проект скомпилирован

Установка и запуск приложения на мобильном устройстве не представляет большого труда.

Для установки приложения выполняем стандартный набор действий (рис. 19.22).


Total Commander
/mnt/sdcard/Download
Adobe Reader
my_orien.
Это приложение еще не установлено на вашем устройстве.
Имя пакета: my_orien
"com.pmn5mail.ru.myorie
Версия файла: 1.0
Вы хотите установить е
Установить
Открыт
Google Play
Total Commander
/mnt/sdcard/Download
Adobe Reader
3. Group Assignments_NArFU_FE
appstore-android.apk
Что использовать?
Проверить и устано
Программа установ пакета
Использовать всегда для действия
z_downloader-2.exe
z_downloader.exe
my_orien
Установить это приложение?
Разрешить этому приложени
Сетевой обмен данным
неограниченный доступ в Ин
Телефонные вызовы
считывание состояния и иде телефона
Системные инструмент
предотвращение перехода те спящий режим, удаление все кэша приложений
Показать все
Отмена
Уст

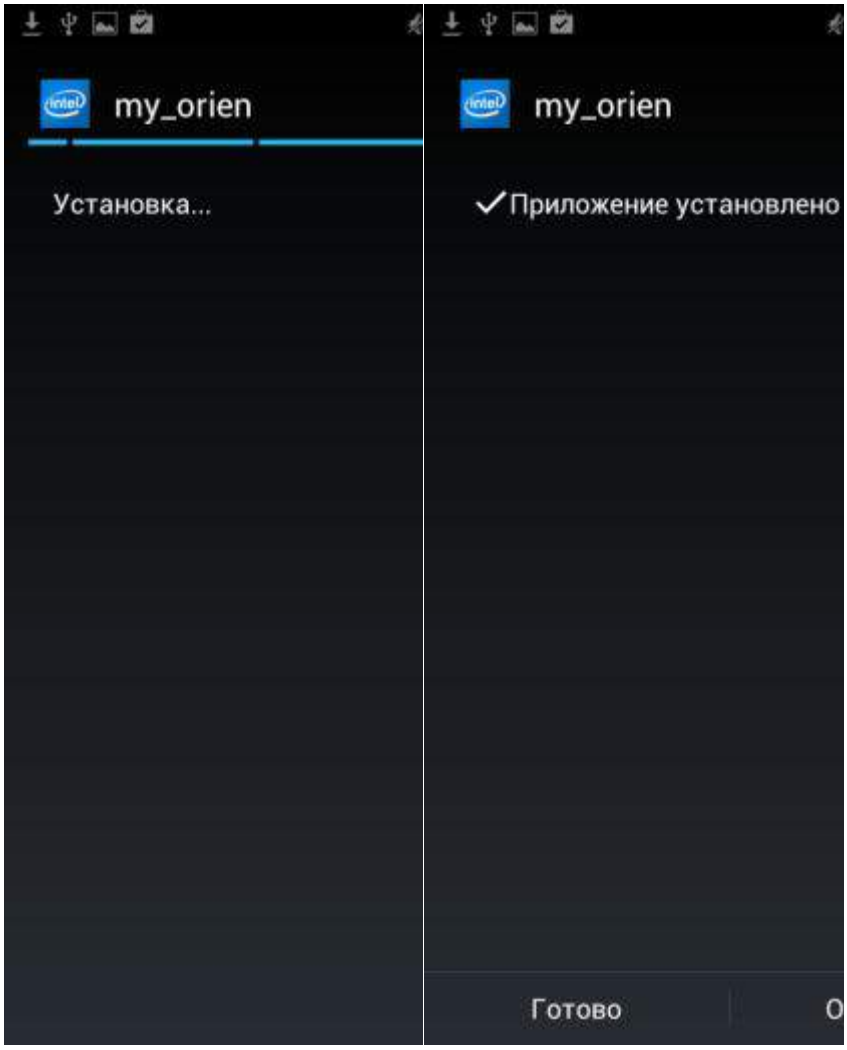
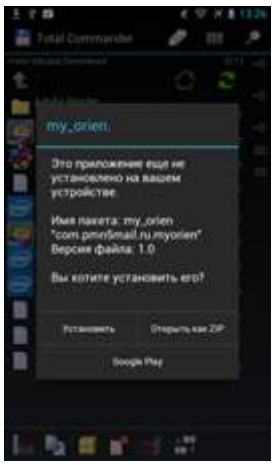
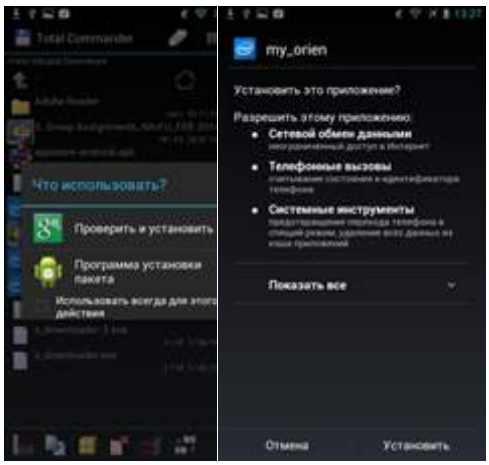
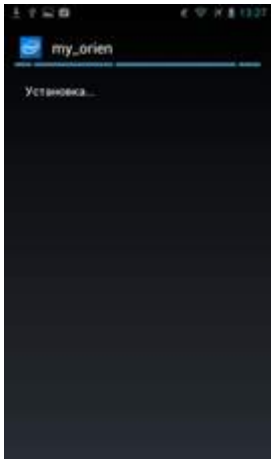


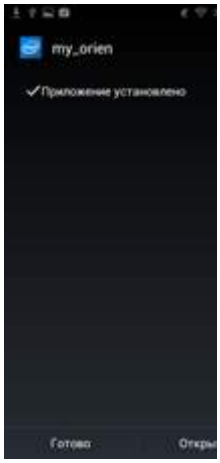
Рис. 19.22. Установка приложения





увеличить изображение увеличить изображение увеличить изображение





увеличить изображение

увеличить изображение

Рис. 8.22. Установка приложения

После установки приложения находим его иконку и запускаем (рис. 19.23).

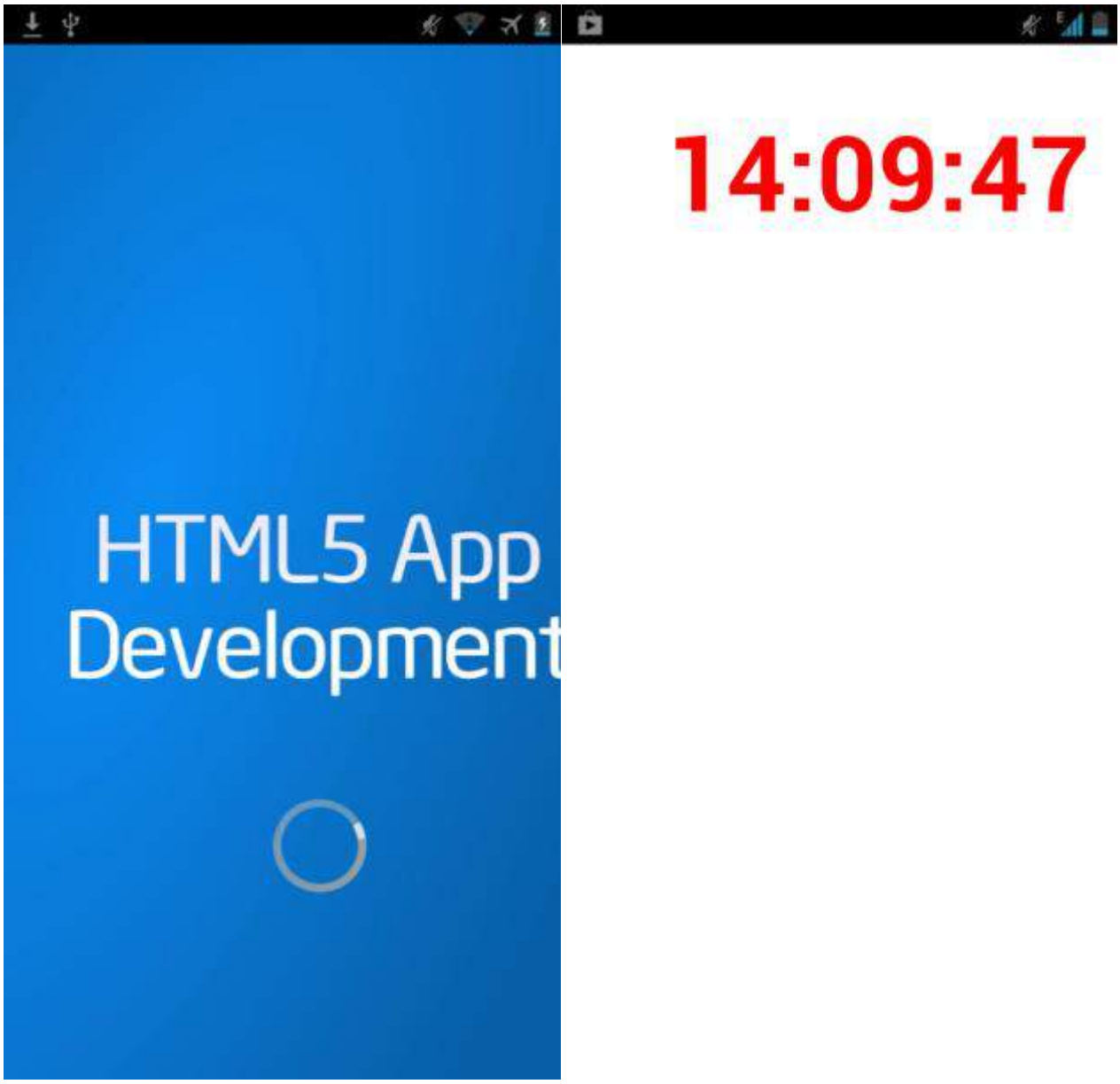


Рис. 19.23. Запуск приложения "Часы" на мобильном устройстве

увеличить изображение

увеличить

изображение

Рис. 19.23. Запуск приложения "Часы" на мобильном устройстве

Задания для самостоятельной работы

1. На основе примера "**Orientation**" разработать приложение с анимированным человечком: в режиме портрета он занимается спортом, в режиме альбома сладко спит.
2. Предлагаем вам проявить творчество и разработать небольшое приложение на основе пустого шаблона и на основе любого из примеров.

```
<!DOCTYPE html><!--HTML5 doctype-->
<html>
<head>
        <title>Your New Application</title>
        <meta http-equiv="Content-type" content="text/html; charset=utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0,
maximum-scale=1.0, minimum-scale=1.0, user-scalable=0" />
        <style type="text/css">
                /* Prevent copy paste for all elements except text fields */
                *  { -webkit-user-select:none; -webkit-tap-highlight-
color:rgba(255, 255, 255, 0); }
                input, textarea  { -webkit-user-select:text; }
                body { background-color:white; color:black }
         #times {color:red;font-size:50pt;text-align:center;}
        </style>
        <script src='intelxdk.js'></script>
        <script type="text/javascript">
                /* This code is used to run as soon as Intel activates */
                var onDeviceReady=function(){
                //hide splash screen
                intel.xdk.device.hideSplashScreen();
                };

        document.addEventListener("intel.xdk.device.ready",onDeviceReady,false
);


         function startTime(){
             var d=new Date();
             var h=d.getHours();
             var m=d.getMinutes();
             var s=d.getSeconds();
             m=FullTime(m);
             s=FullTime(s);
             document.getElementById('times').innerHTML=h+":"+m+":"+s;
             t=setTimeout('startTime()',500);
         }
         function FullTime(i){
             if (i<10)
             {
                 i="0" + i;
             }
             return i;
         }
        </script>
</head>
<body onload="startTime()">
        <!-- content goes here-->
<h1 id="times"></h1>
</body>
```

```
</html>
```

Листинг 19.1. Приложение "Часы"

Внимание! Если Вы увидите ошибку на нашем сайте, выделите её и нажмите Ctrl+Enter.
© Национальный Открытый Университет "ИНТУИТ", 2016 | www.intuit.ru