

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
по выполнению лабораторных работ
по дисциплине «Программирование мобильных устройств»
для студентов направления подготовки
09.03.01 Информатика и вычислительная техника
Направленность (профиль)
« Программное обеспечение средств вычислительной техники и автоматизированных систем»

Ставрополь 2026

Содержание

Лабораторная работа 1. Установка инструментария и настройка среды для разработки мобильных приложений.....	5
Лабораторная работа 2. Установка среды разработки мобильных приложений с применением виртуальной машины.....	10
Лабораторная работа 3. Создание эмуляторов и подключение устройств. Настройка режима терминала.....	19
Лабораторная работа 4. Создание нового проекта. Изучение и комментирование кода.....	33
Лабораторная работа 5. Изменение элементов дизайна. Обработка событий: подсказки. Обработка событий: цветовая индикация.....	85
Лабораторная работа 6. Подготовка стандартных модулей. Обработка событий: переключение между экранами. Передача данных между модулями.....	103
Лабораторная работа 7. Тестирование и оптимизация мобильного приложения.....	121

Введение

Настоящие методические указания предназначено для проведения лабораторных работ по дисциплине Программирование мобильных устройств для студентов направления подготовки 09.03.01 Информатика и вычислительная техника, а также для студентов других направлений, направленных на подготовку программистов со знаниями основ разработки мобильных приложений.

Пособие посвящено проектированию интерфейсов и программированию мобильных приложений на платформе Android: освещаются архитектура и основные компоненты системы Android (деятельность, служба, приемник широковещательных намерений, контент-провайдер); элементы экрана Android-приложения и их свойства, обработка событий в Activity; принципы работы с несколькими Activity в Android-приложениях; работа с всплывающими сообщениями, списками и фрагментами; механизмы обратного вызова для отслеживания событий в многофункциональных Android-приложениях. Подробно раскрываются вопросы хранения информации с использованием базы данных SQLite.

Лабораторная работа 1. Установка инструментария и настройка среды для разработки мобильных приложений

Цель работы:

Установка и настройка среды программирования ADT Bundle

Задачи работы:

- Установить и настроить среду программирования ADT
- Создать первое приложение **Hello, world!**
- Научиться запускать приложение на эмуляторе мобильного устройства

Введение

Лабораторная работа посвящена описанию работы в среде ADT Bundle (Android IDE). 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 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 Android SDK. 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>).

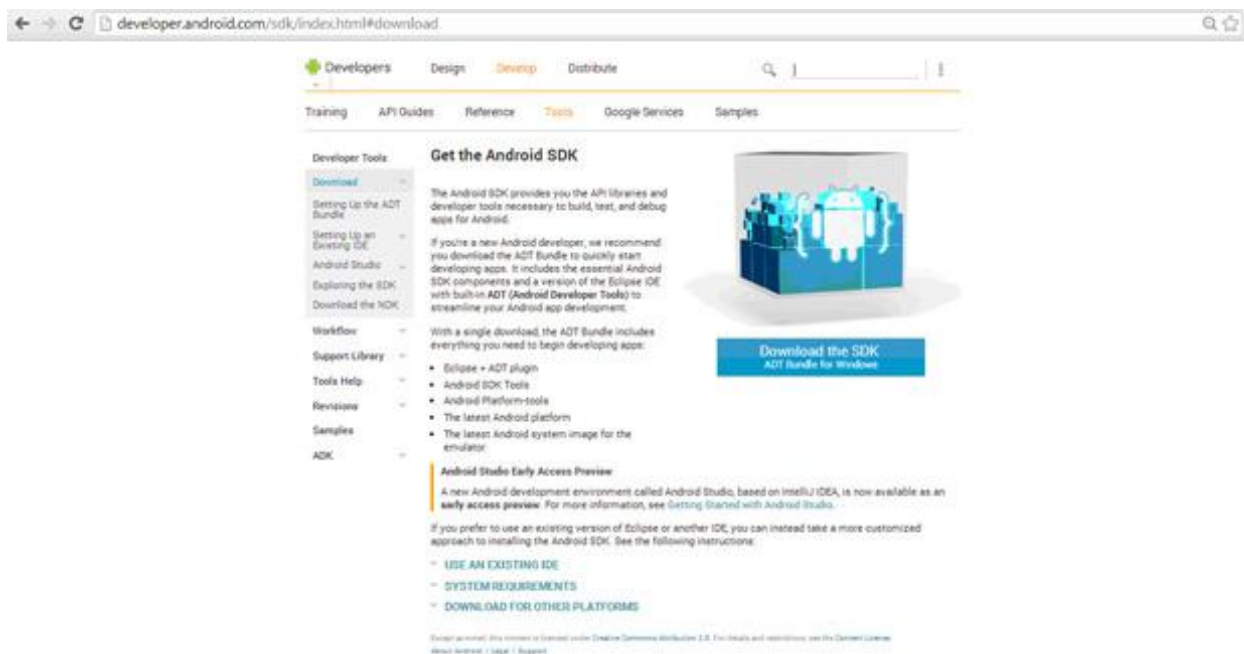


Рис. 1. Сайт разработчика

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

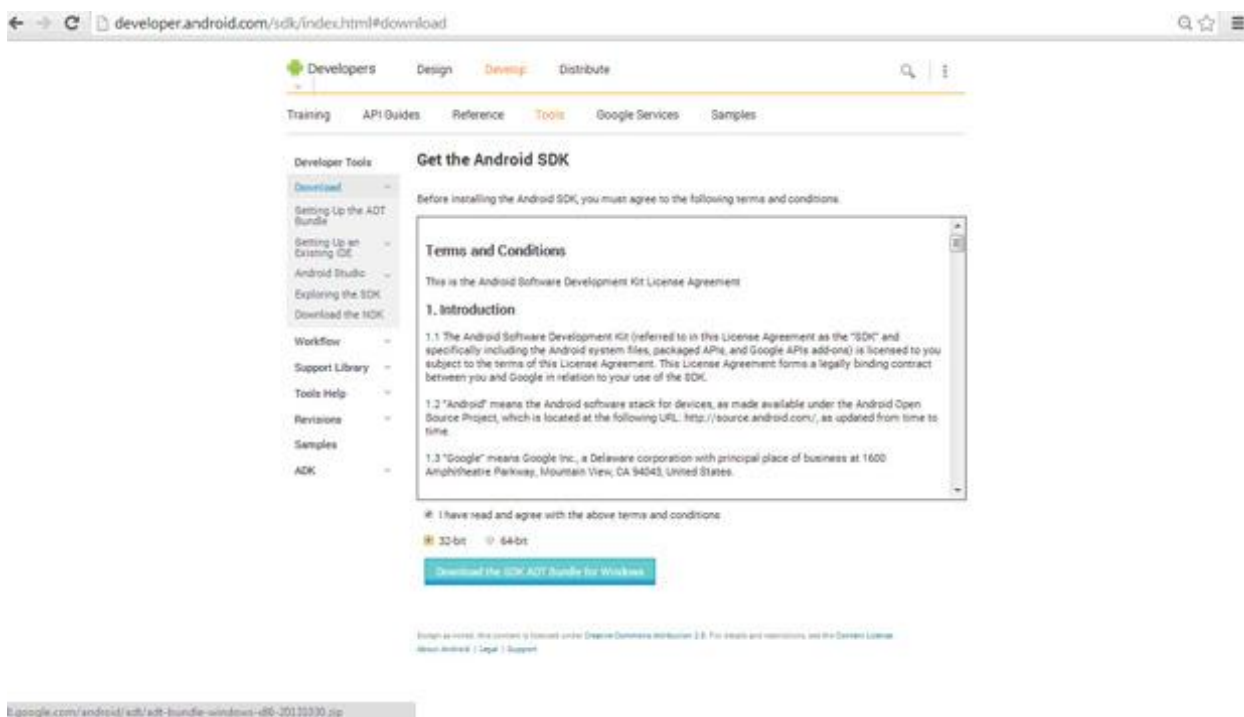


Рис. 2. Скачивание среды

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>)

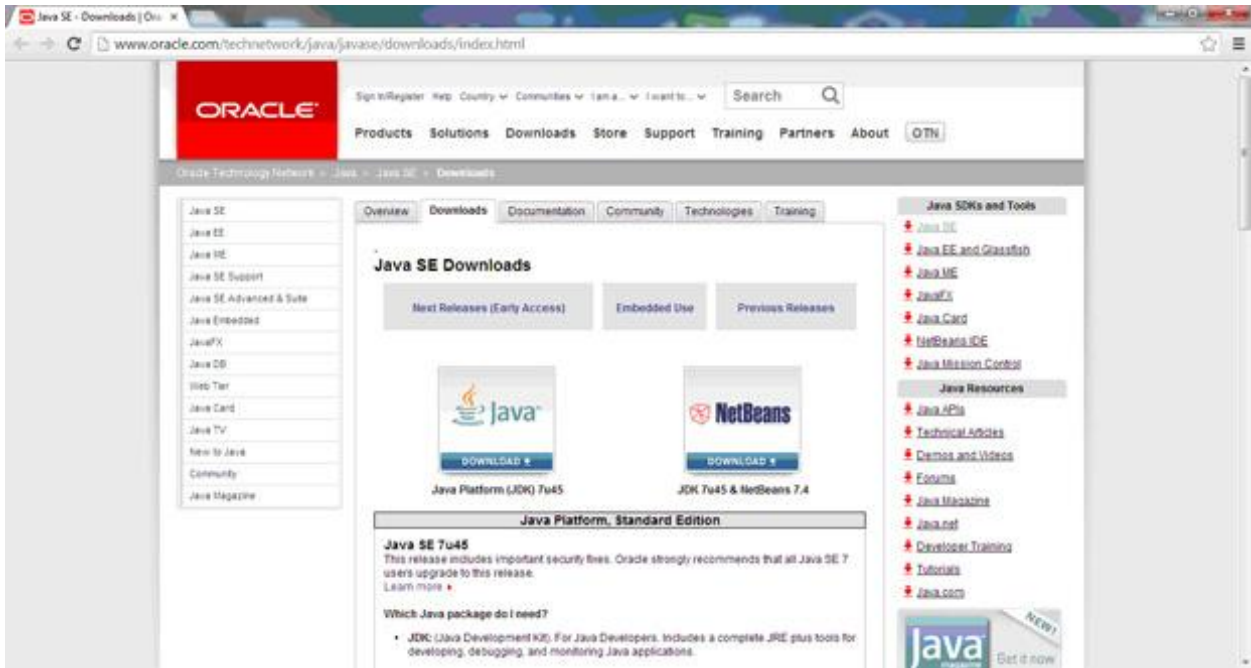


Рис. 3. Сайт компании Oracle

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

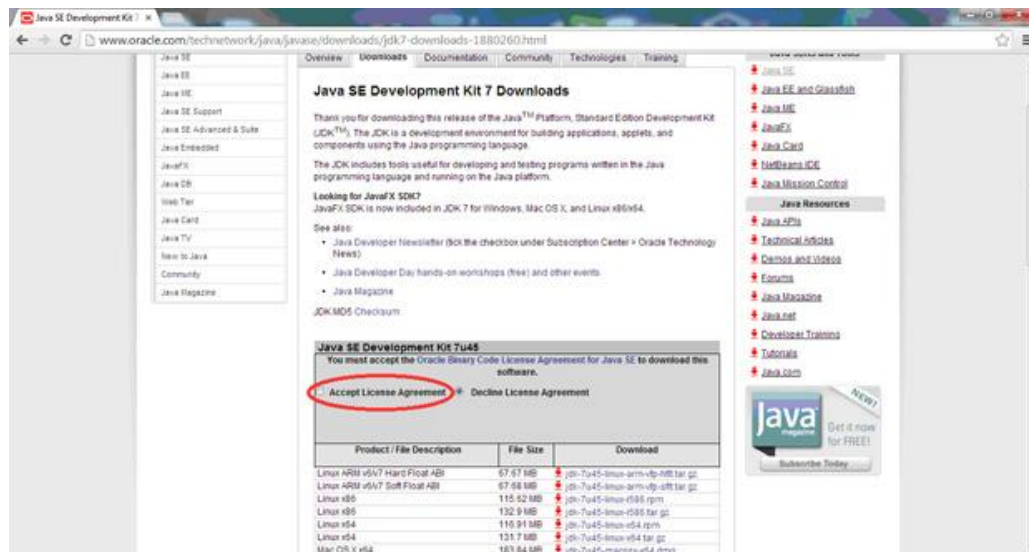


Рис. 4. Скачивание JDK

После скачивания запустите setup-файл и установите JDK.



Рис. 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.

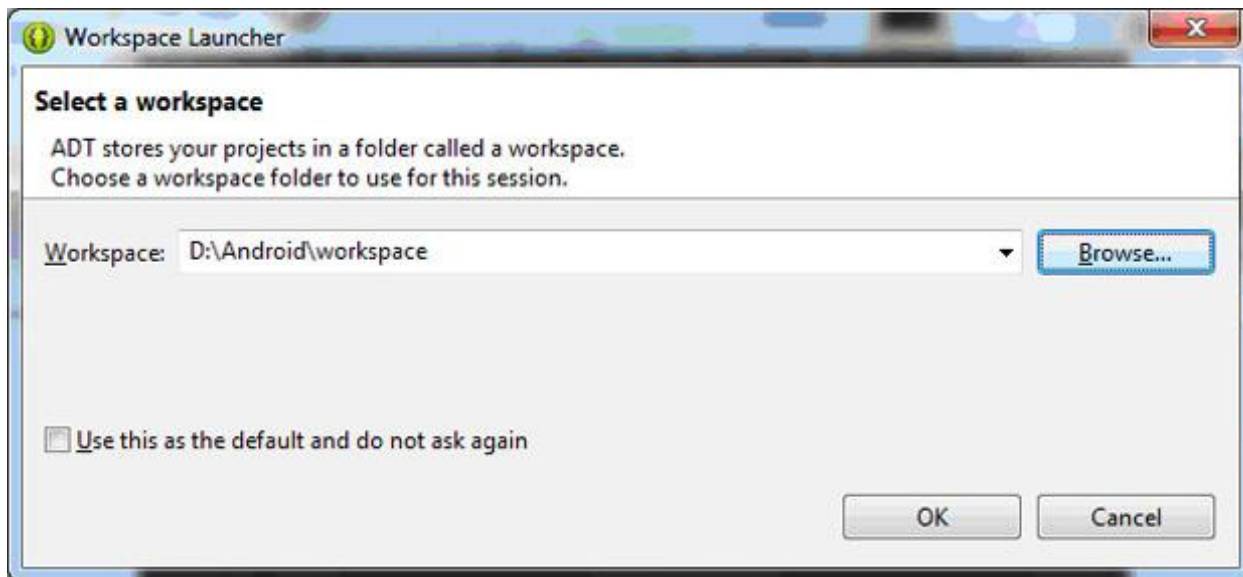


Рис. 6. Выбор рабочего пространства

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

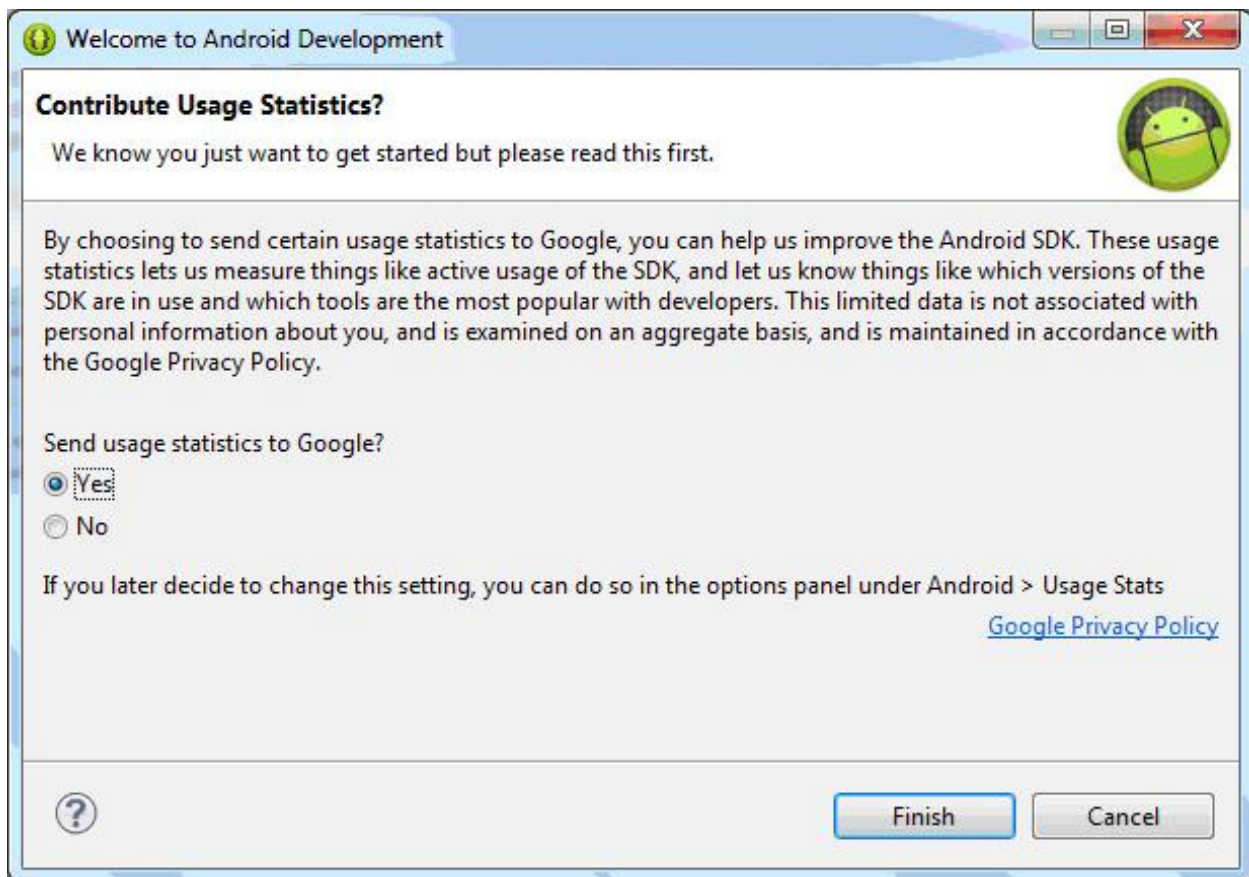


Рис. 7. Отправка статистики

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

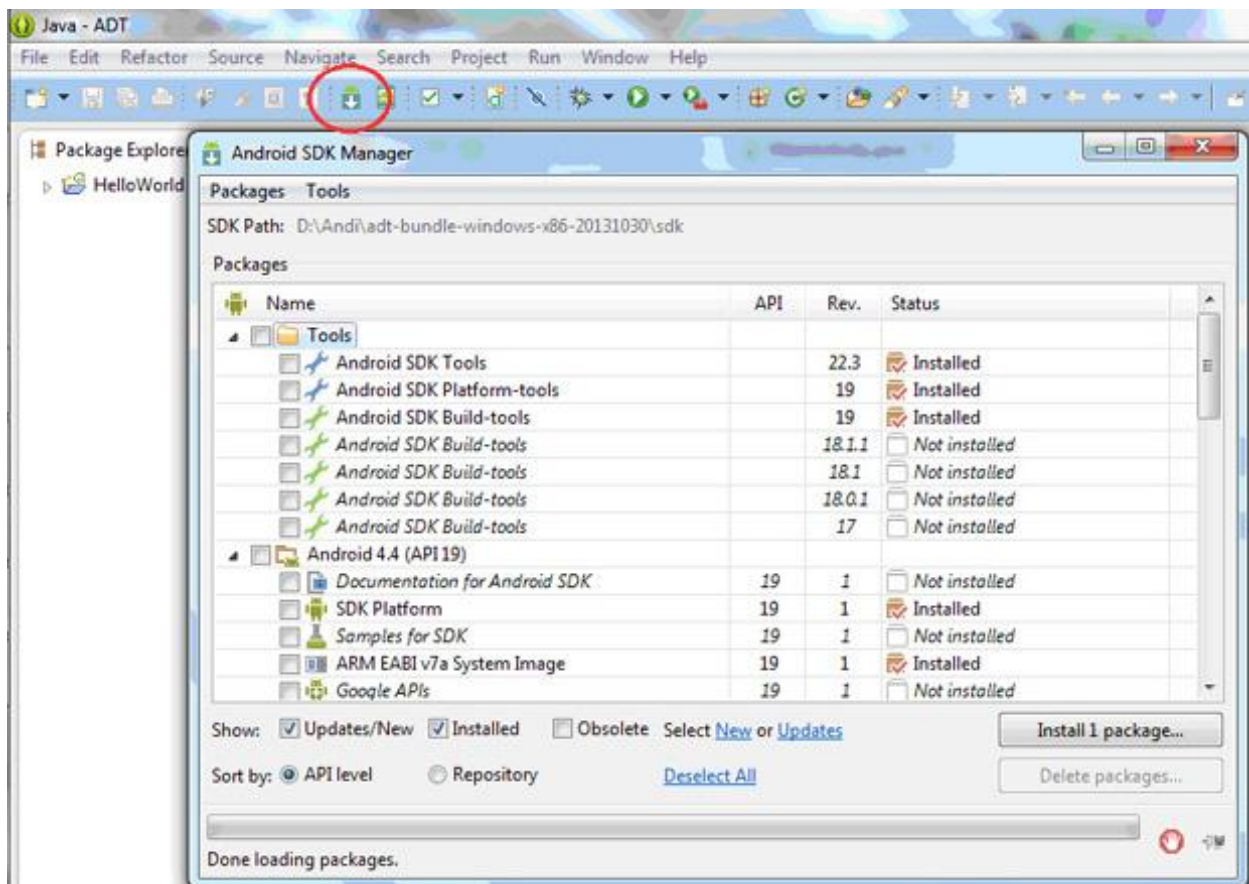


Рис. 8. Android SDK Manager

Создание проекта

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

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

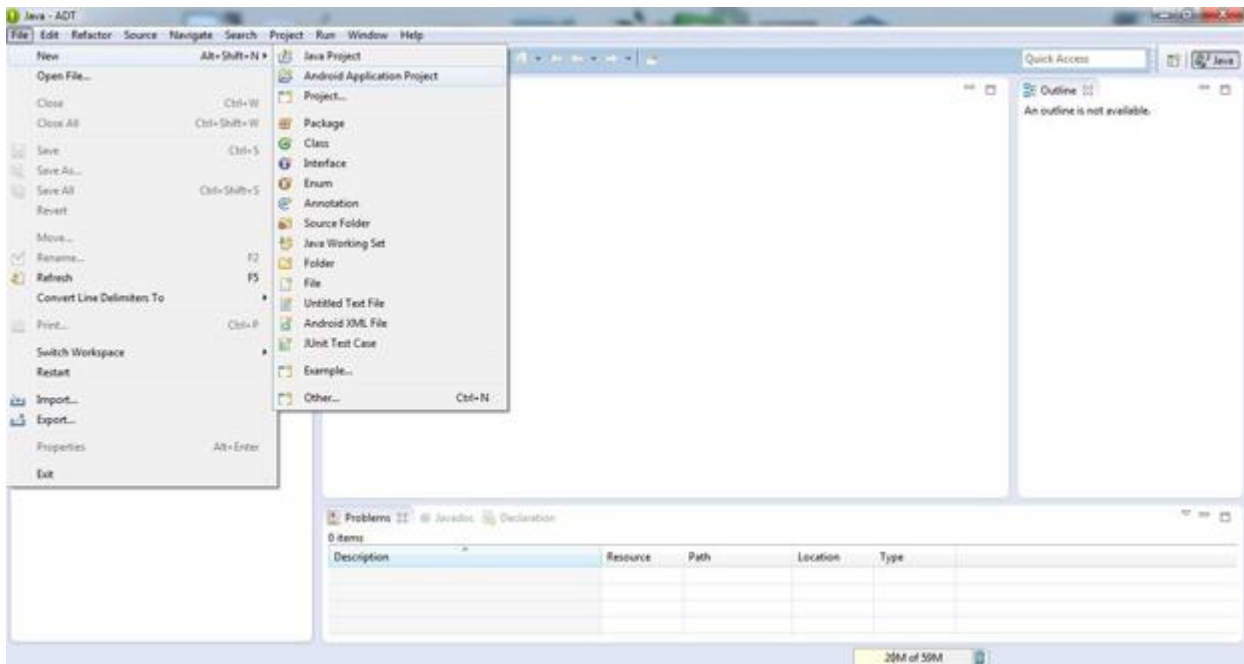


Рис. 9. Создание проекта

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 Google Play. Конечно, учебные приложения туда не загружают, однако, следует иметь это в виду на будущее.

Minimum Required 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.

Compile With определяет, возможности какой версии Android будет использовать приложение.

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

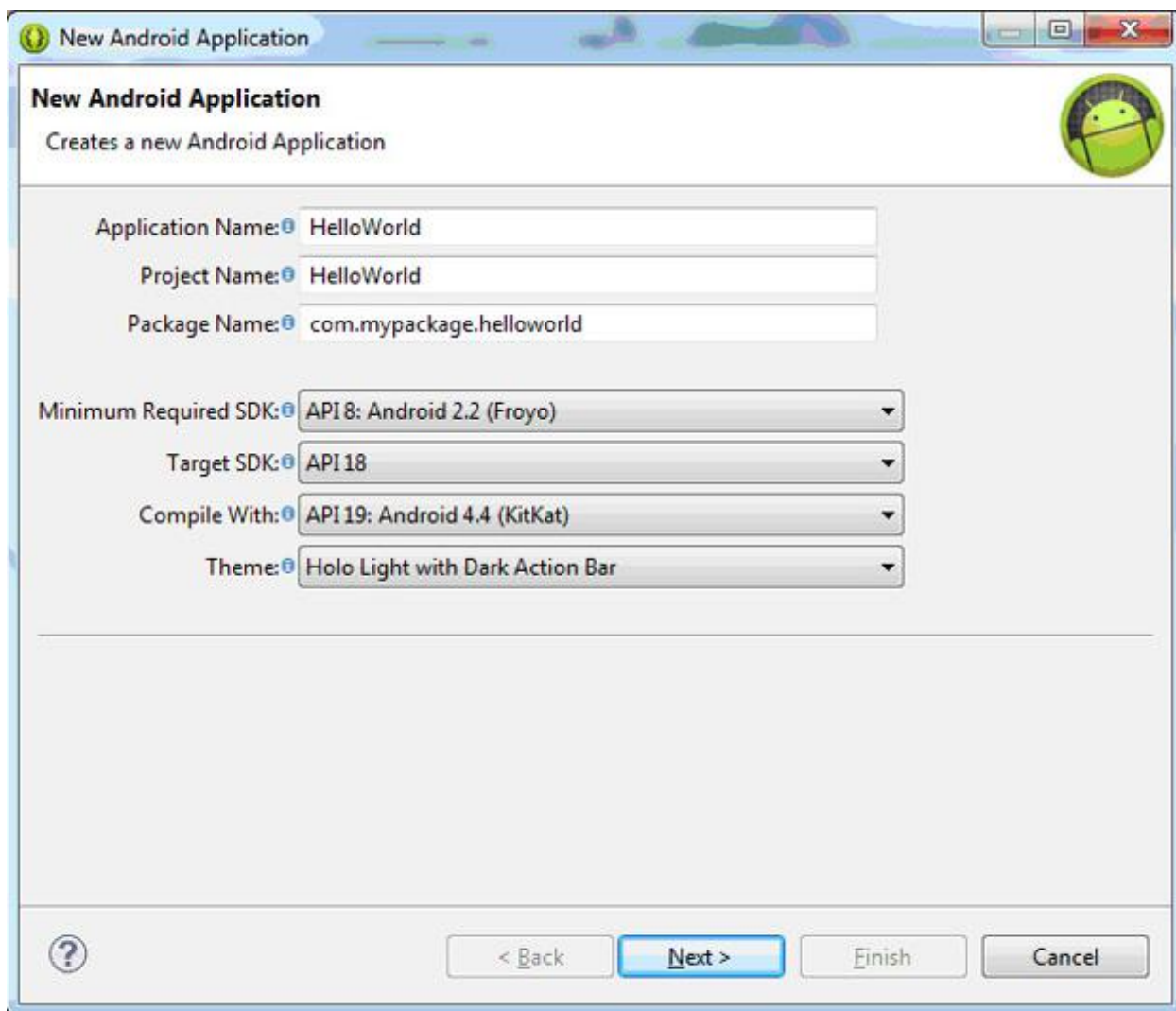


Рис. 10. Наименование проекта

Следующее окно можно пропустить без изменений.

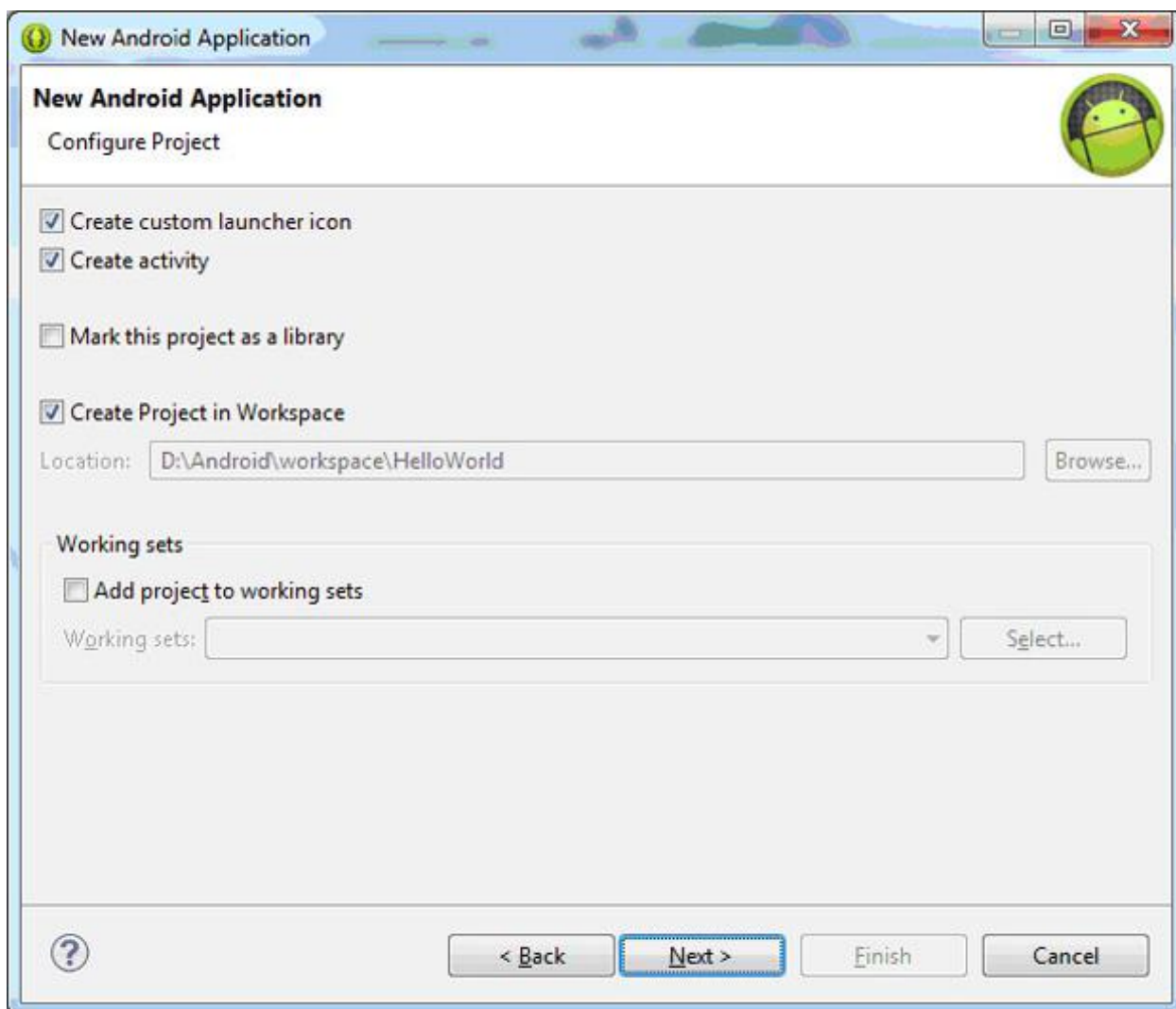


Рис. 11. Конфигурация проекта

Create custom launcher icon - создать значок приложения.

Create activity - создать Activity (активность, деятельность).

Mark this project as library - создать проект, как библиотеку. Сейчас в этом нет необходимости, наше приложение в других проектах использоваться не будет.

Create Project in Workspace - создать проект в папке 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.

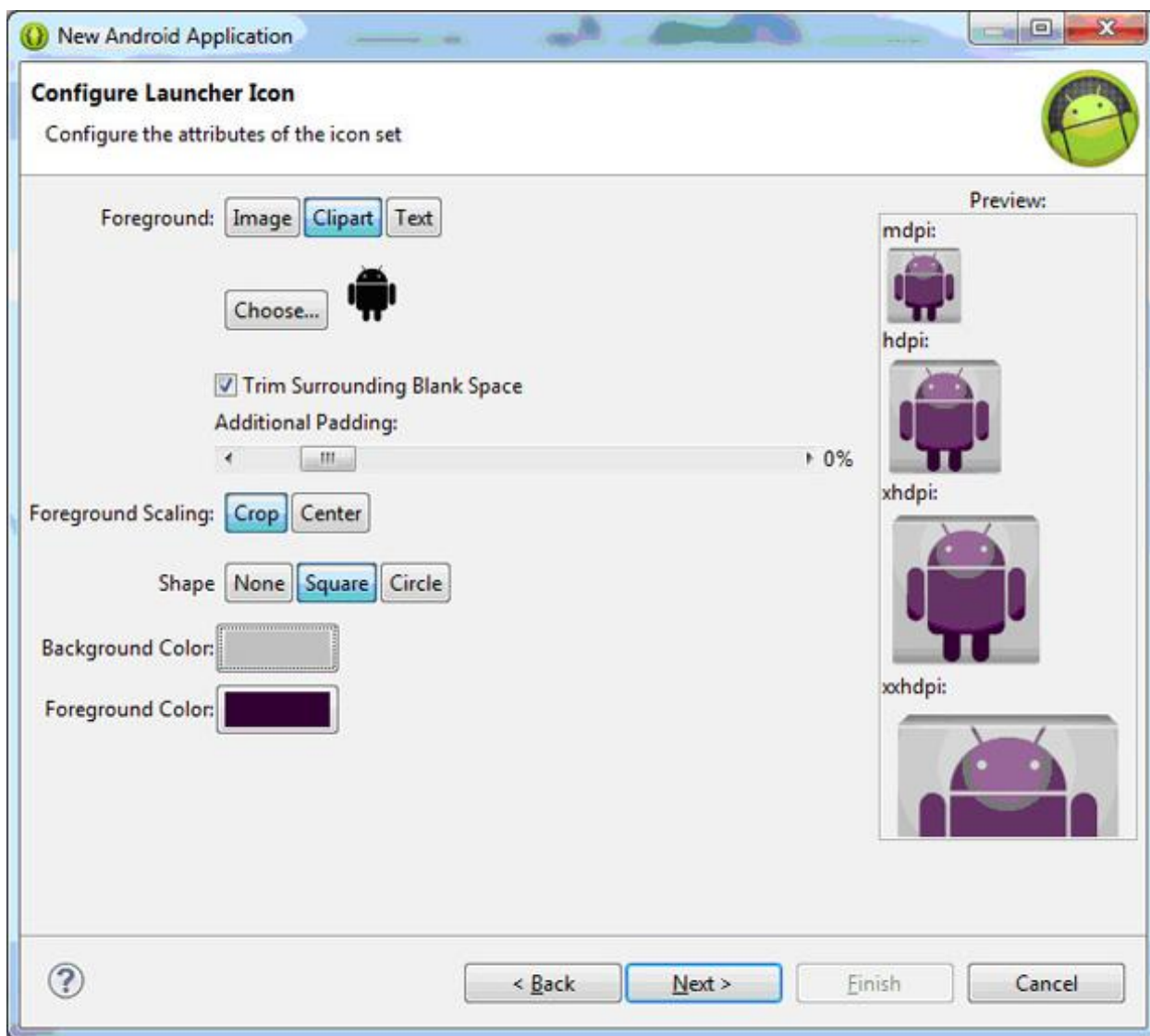


Рис. 12. Создание иконки приложения

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).

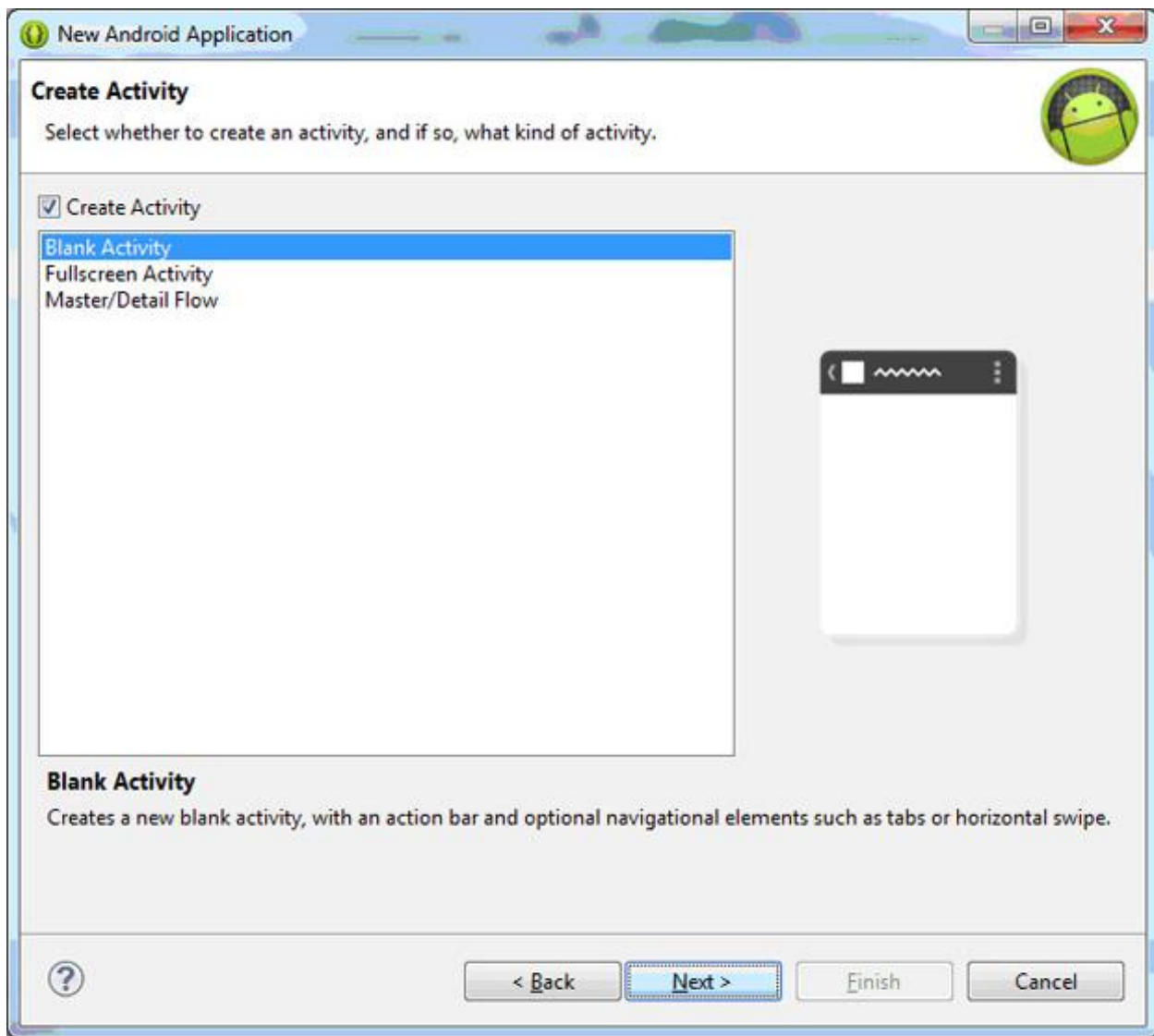


Рис. 13. Создание активности

Blank Activity - шаблон, предназначенный для мобильных телефонов.

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

Master/Detail Flow - шаблон, предназначенный для планшетных компьютеров.

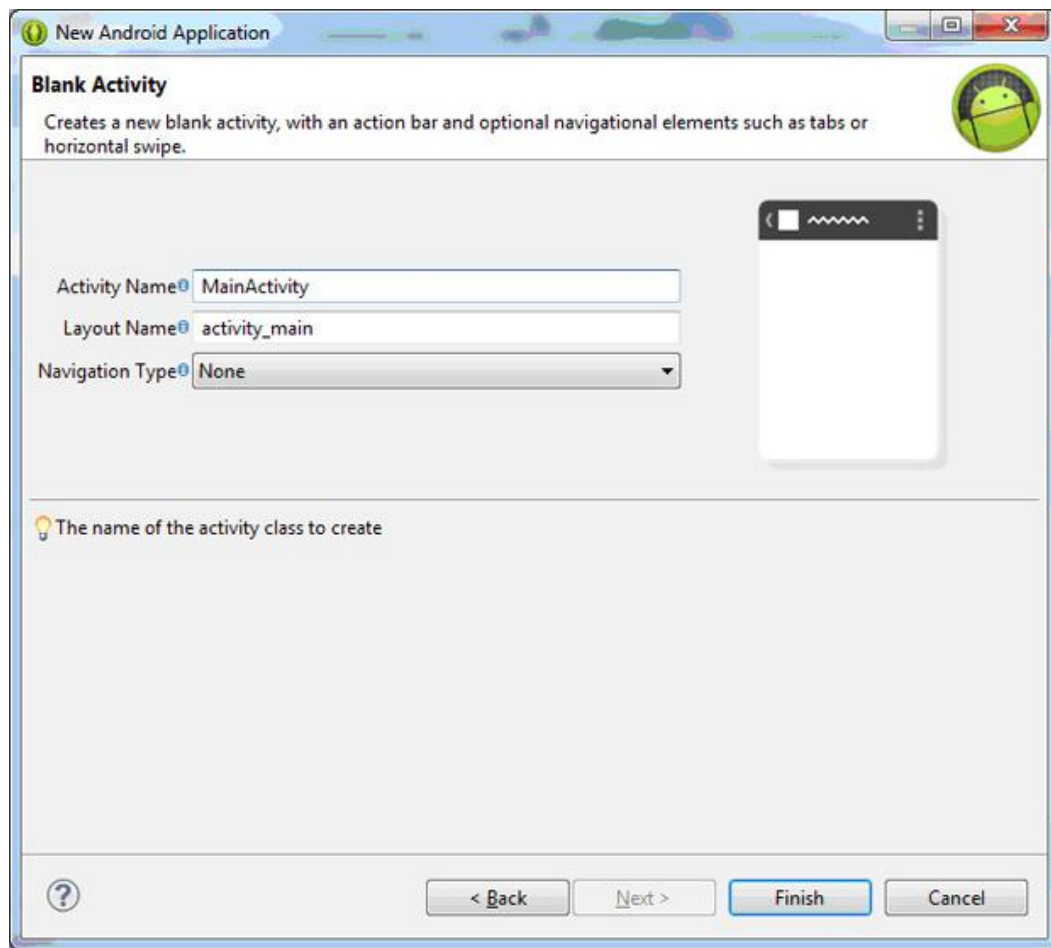


Рис. 14. Переименование активности

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**.

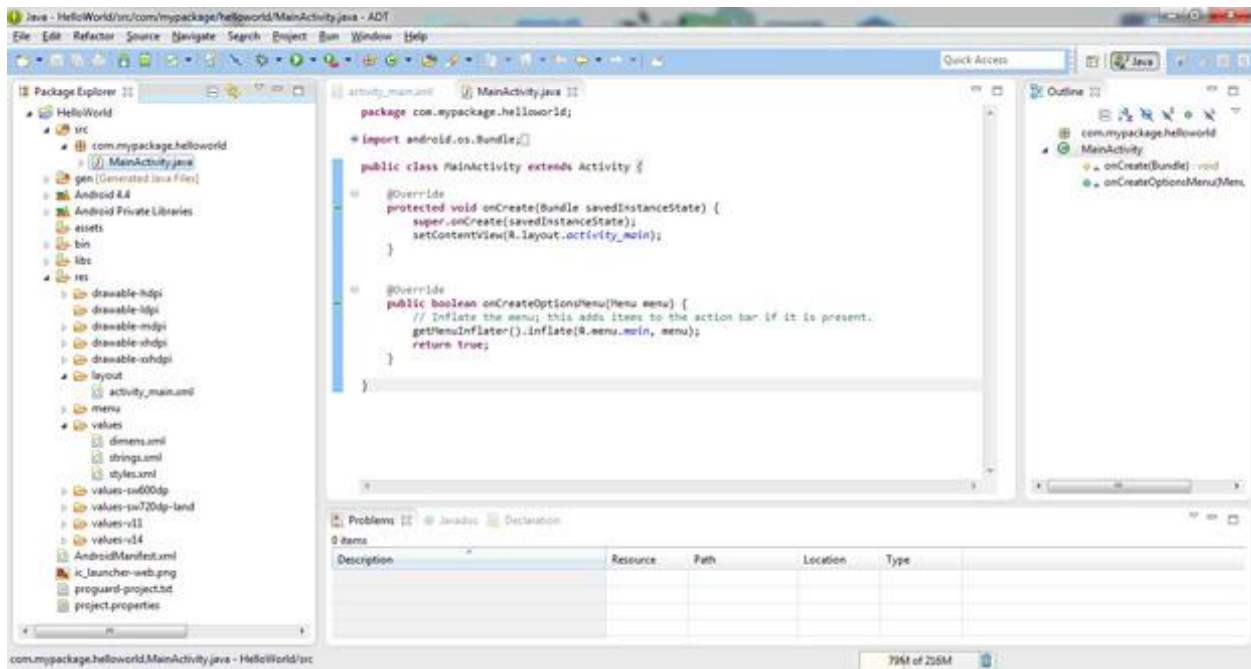


Рис. 15. Активность

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.

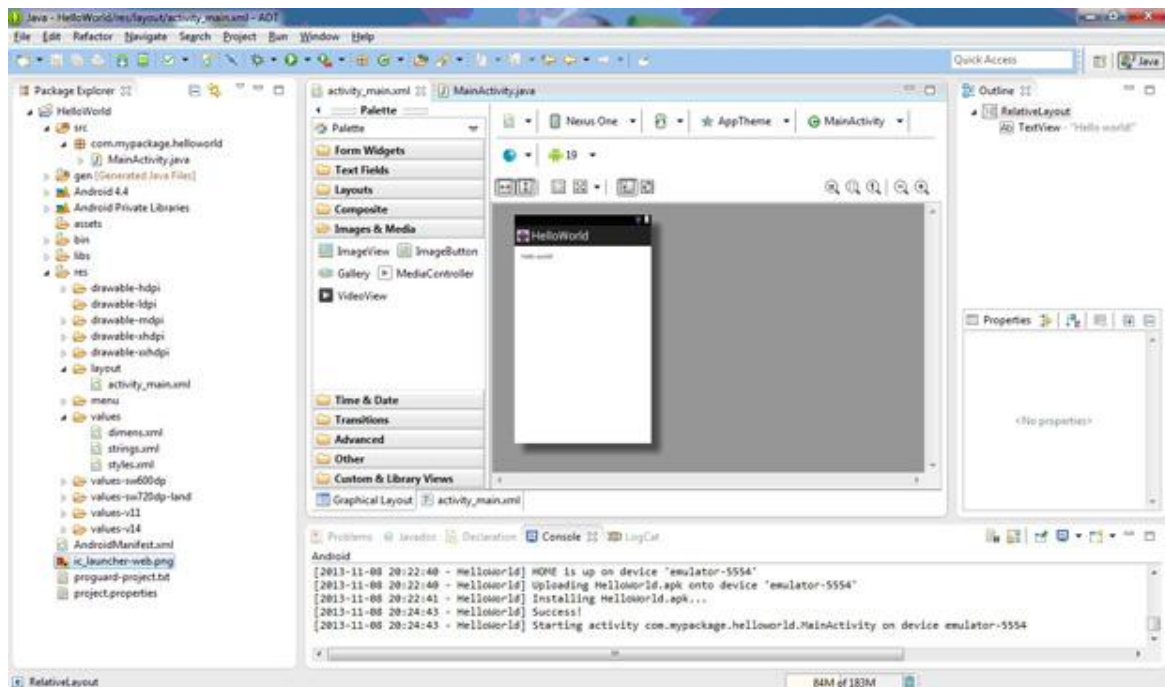


Рис. 16. Xml-файл. Графический редактор

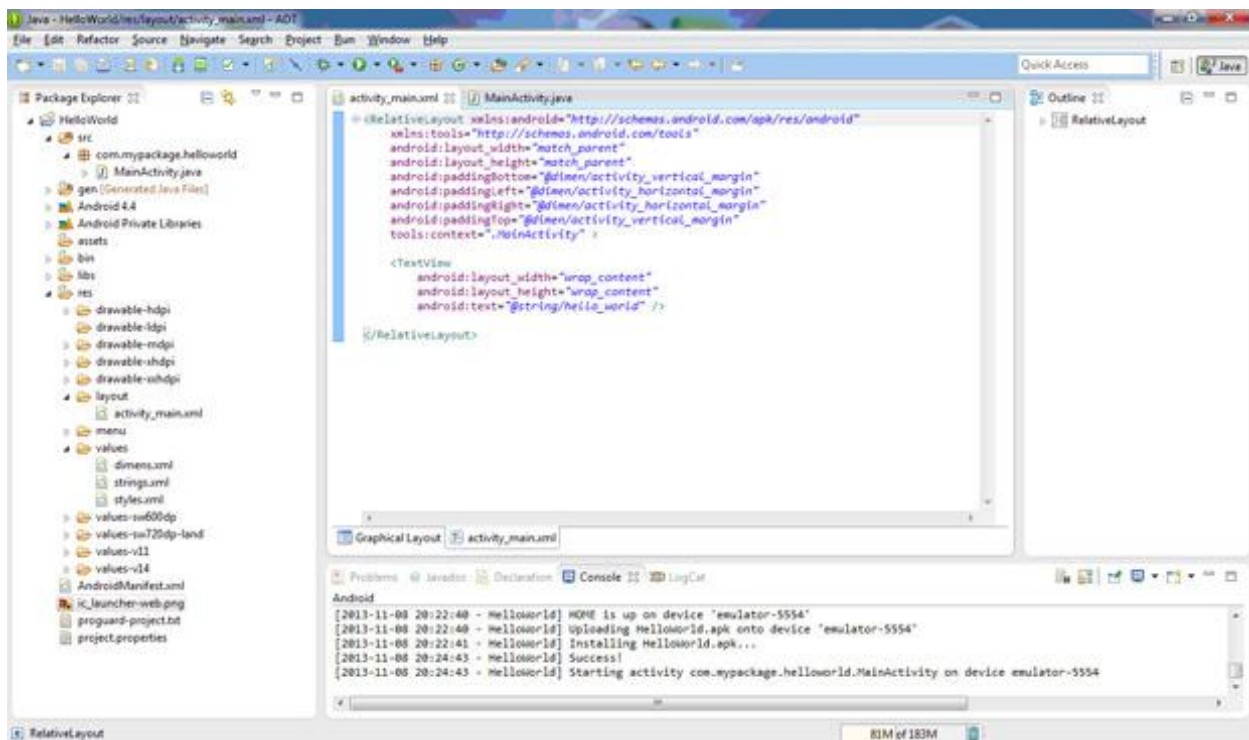


Рис. 17. Xml-файл

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!*

Лабораторная работа 3. Создание эмуляторов и подключение устройств. Настройка режима терминала

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**.

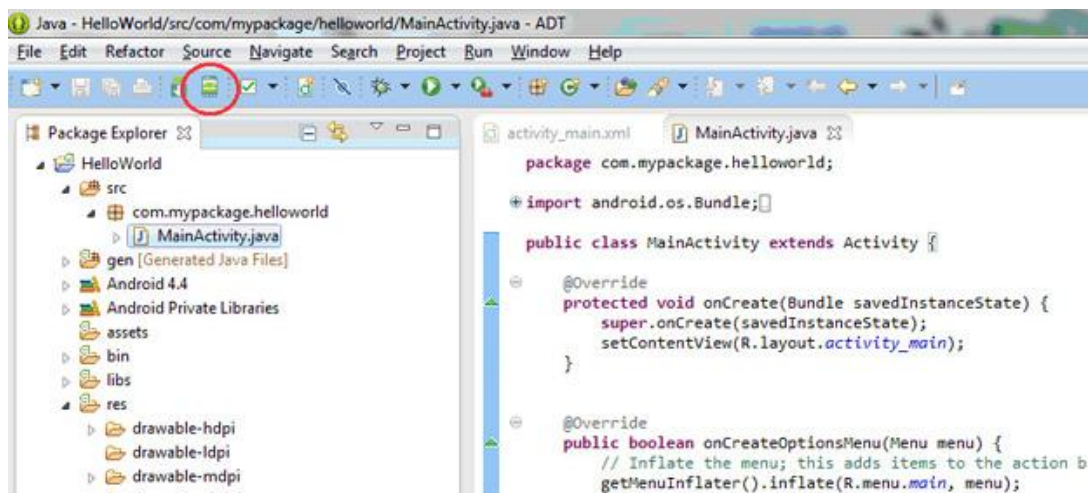


Рис. 18. Запуск проекта

Откроется Android Virtual Device Manager. Пока в нем нет ни одного виртуального устройства.

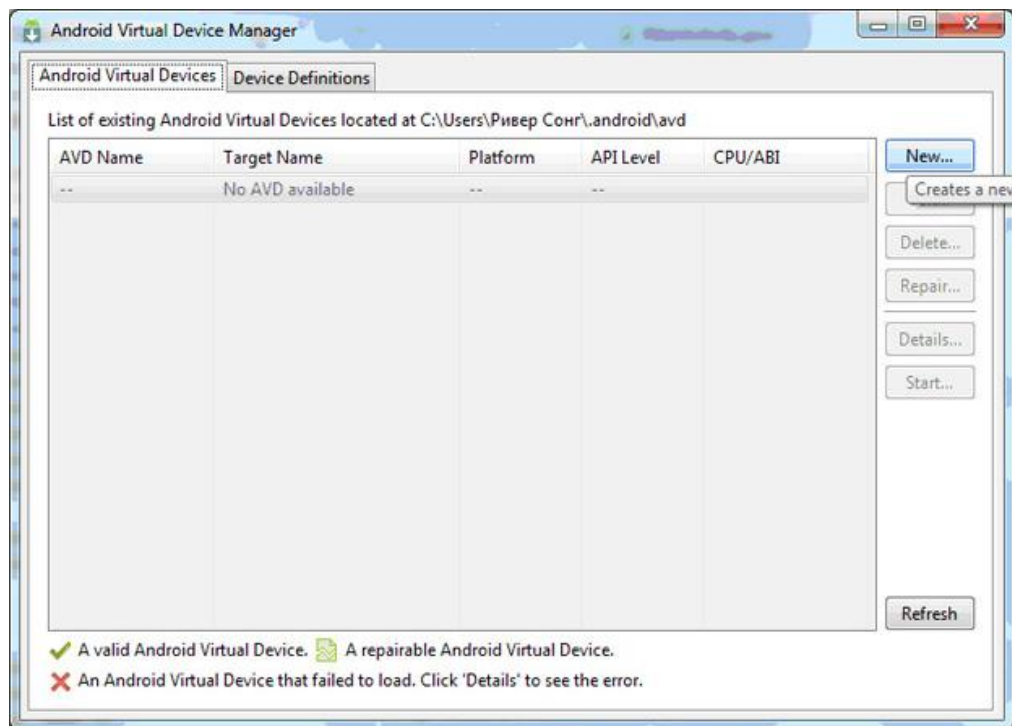


Рис. 19. Android Virtual Device Manager

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.

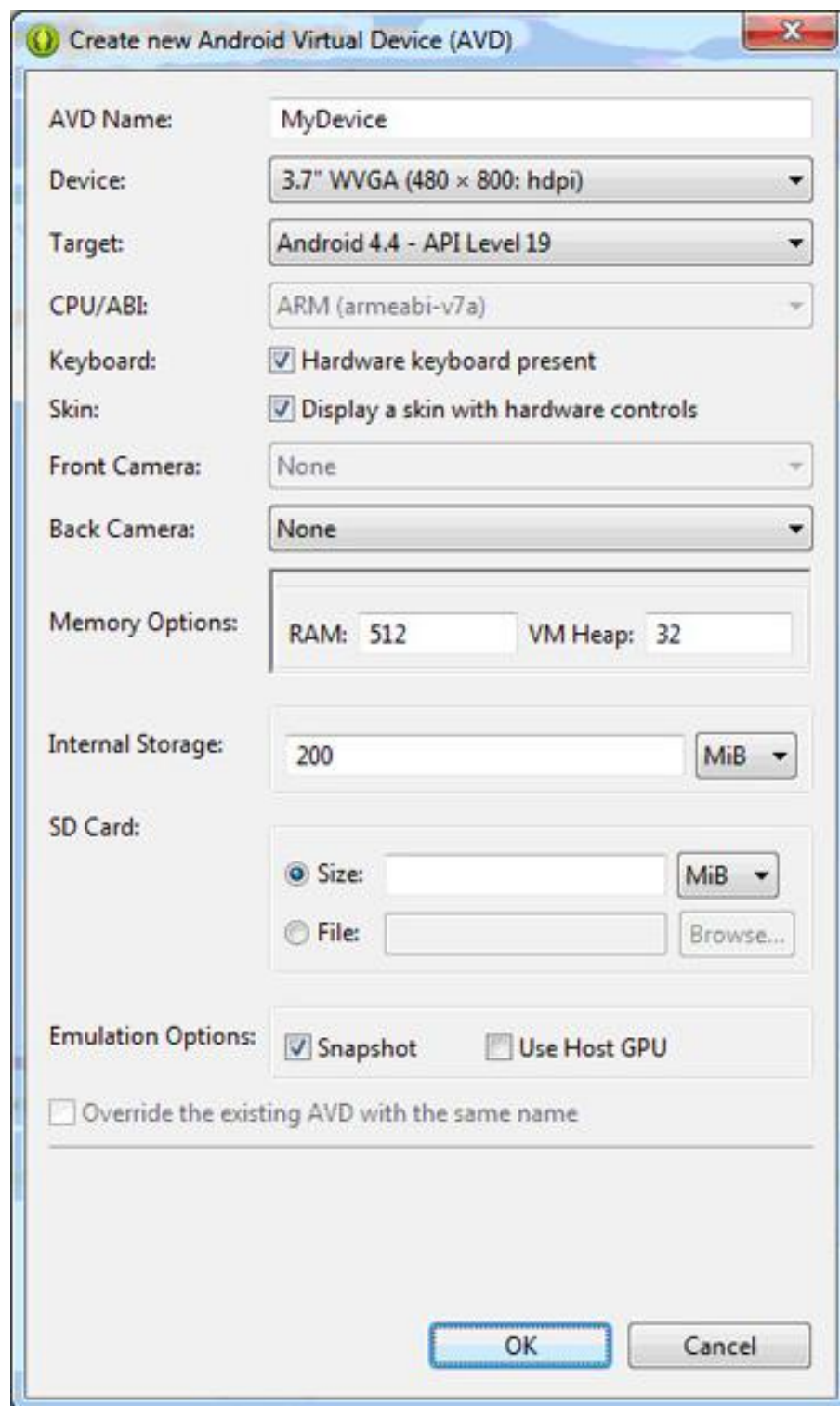


Рис. 20. Создание AVD

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 **Run Configurations**, затем во вкладке **Target** выбрать созданное устройство и запустить проект снова.

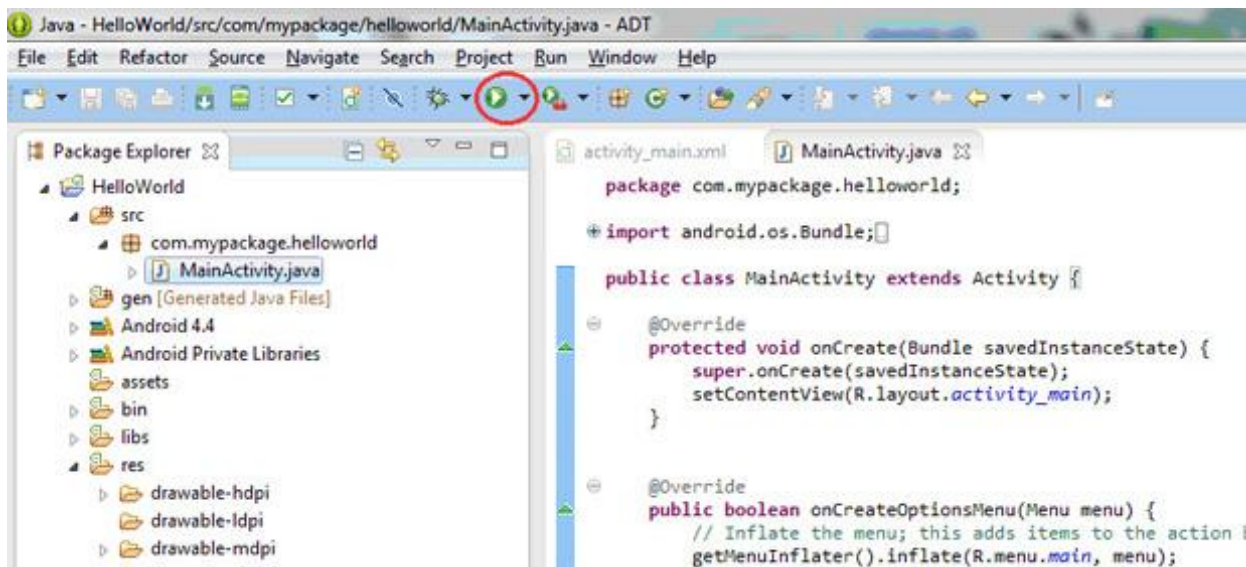


Рис. 21. Запуск приложения

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.



Рис. 22. Запуск эмулятора



Рис. 23. Запущенный эмулятор

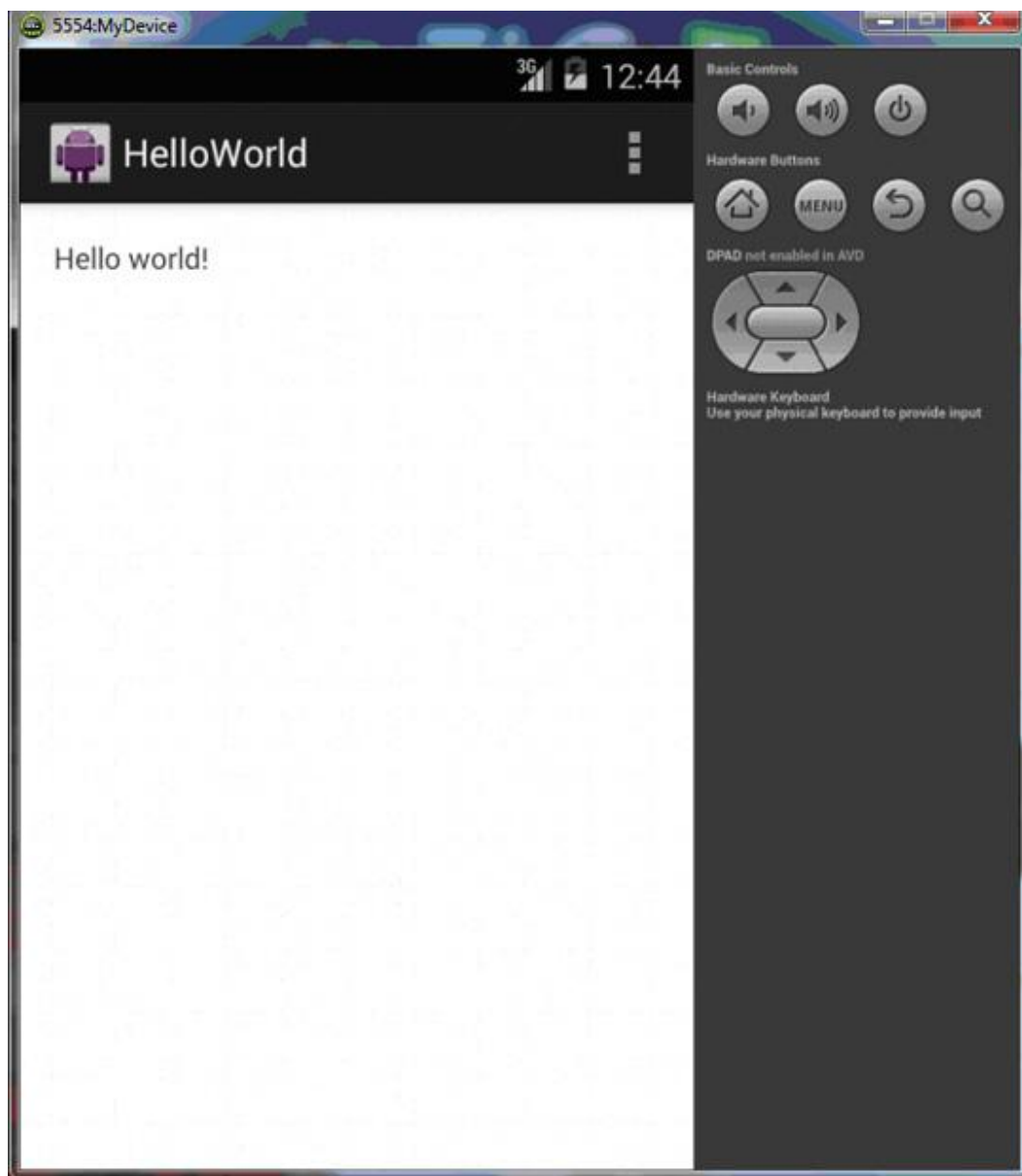


Рис. 24. Приложение Hello, world!

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

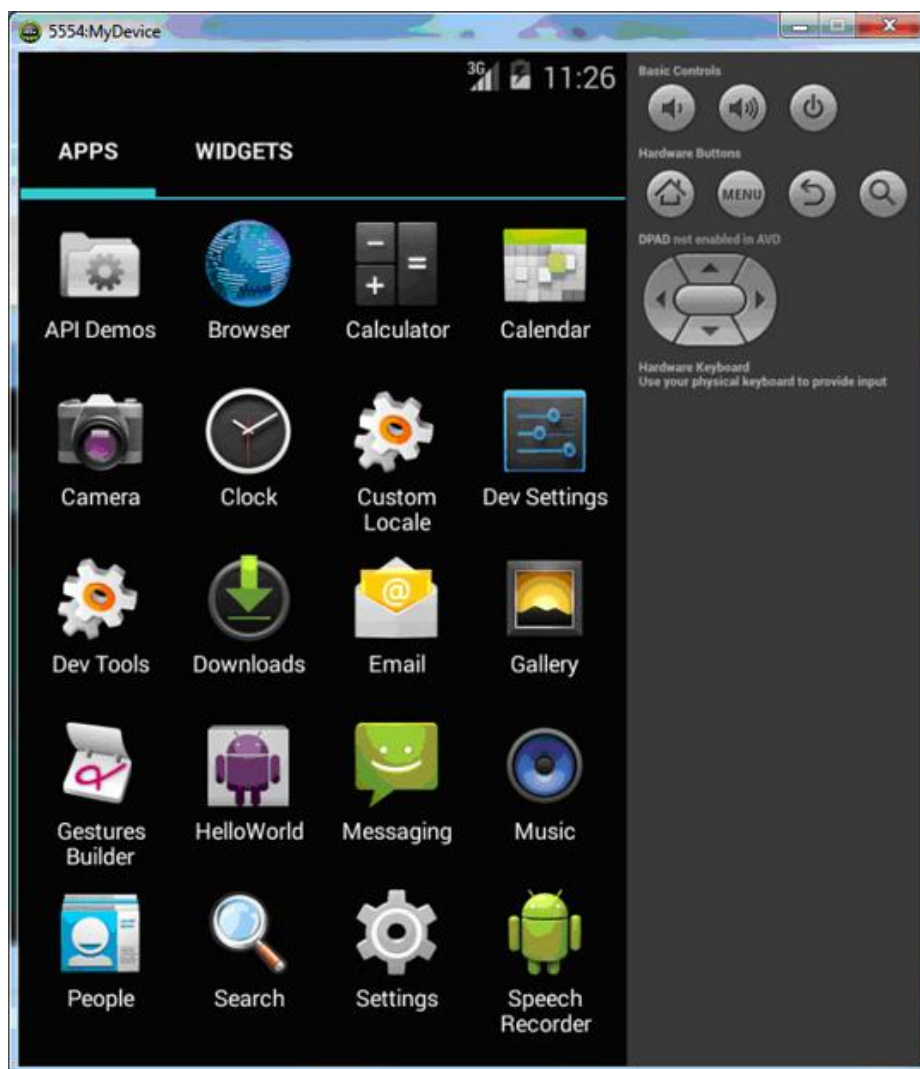


Рис. 25. Меню приложений

Запуск проекта на устройстве

Прежде чем запустить проект на реальном устройстве, необходимо:

- Настроить устройство
- Настроить компьютер
- Настроить среду

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 Google Play, необходимо разрешить установку приложений из альтернативных источников.

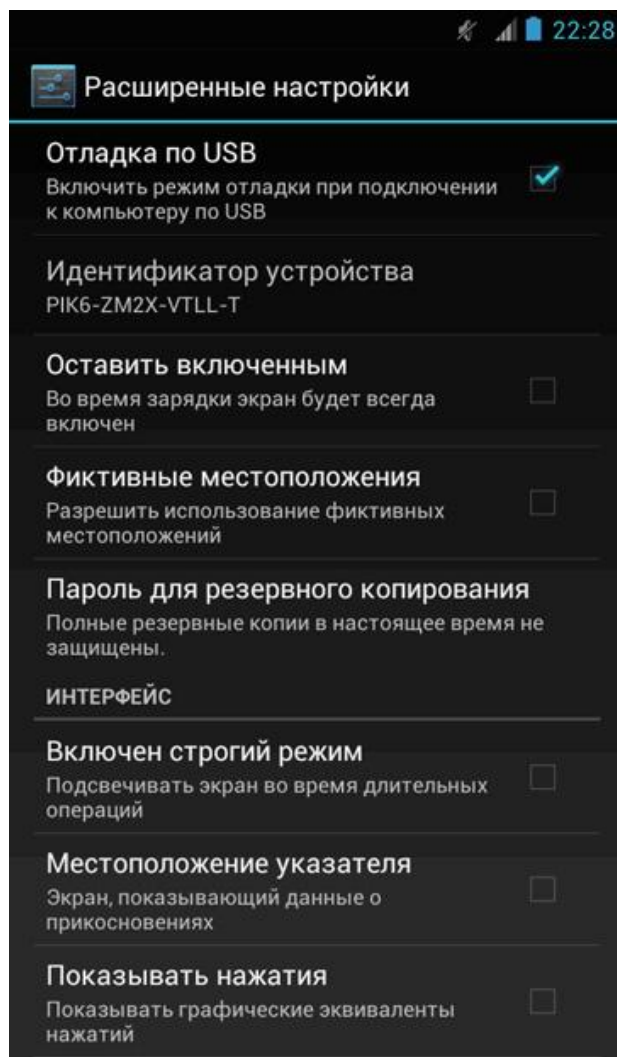


Рис. 26. Настройка устройства

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

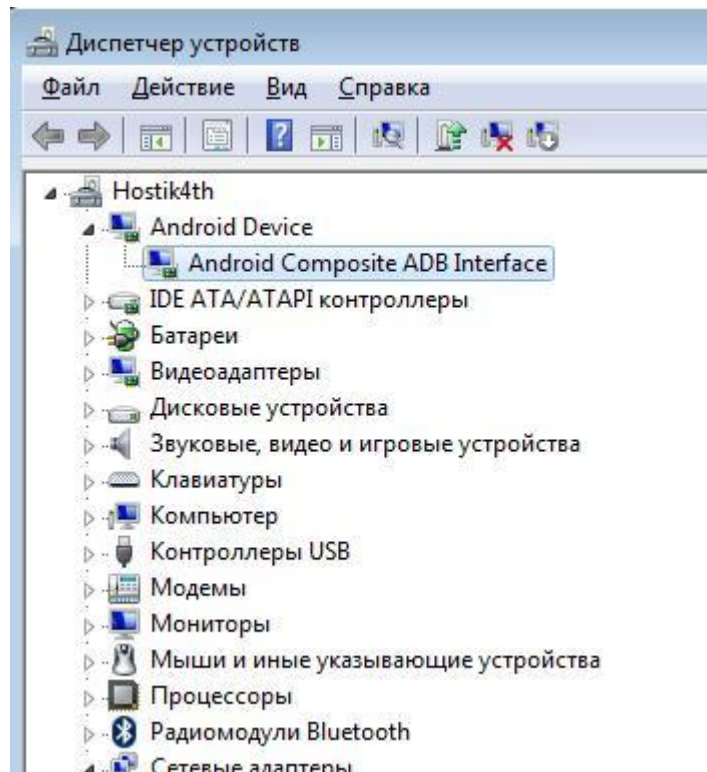


Рис. 27. Диспетчер устройств

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

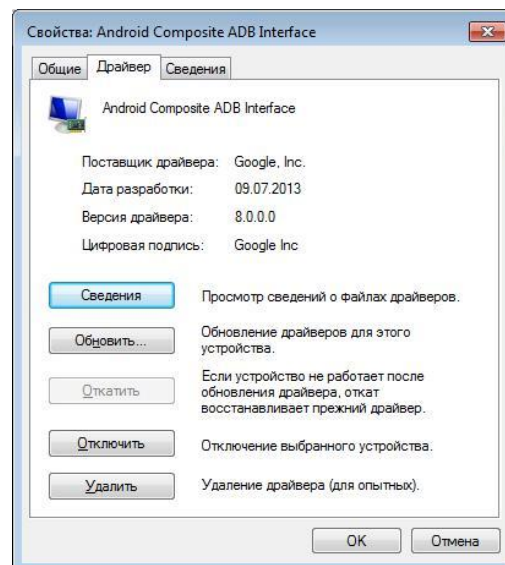


Рис. 2.28. Меню Свойства

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

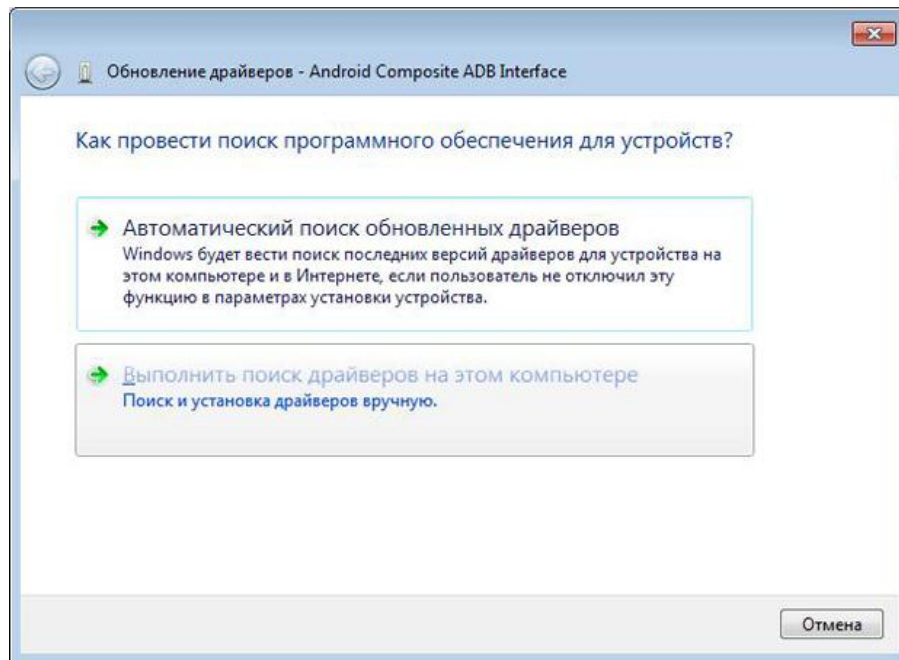


Рис. 29. Обновление драйвера

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

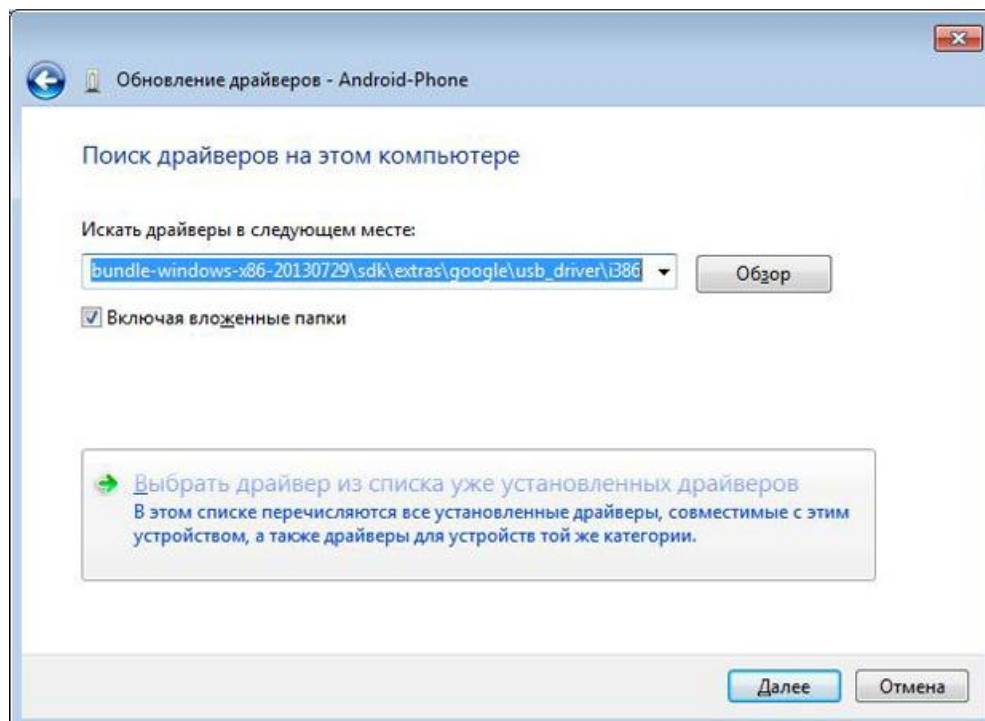


Рис. 30. Поиск драйверов на компьютере

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

В нашем случае путь следующий: **adt-bundle-windows-x86-20130729\sdk\extras\google\usb_driver\i386**.

Или можно отыскать драйвер следующим образом:

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

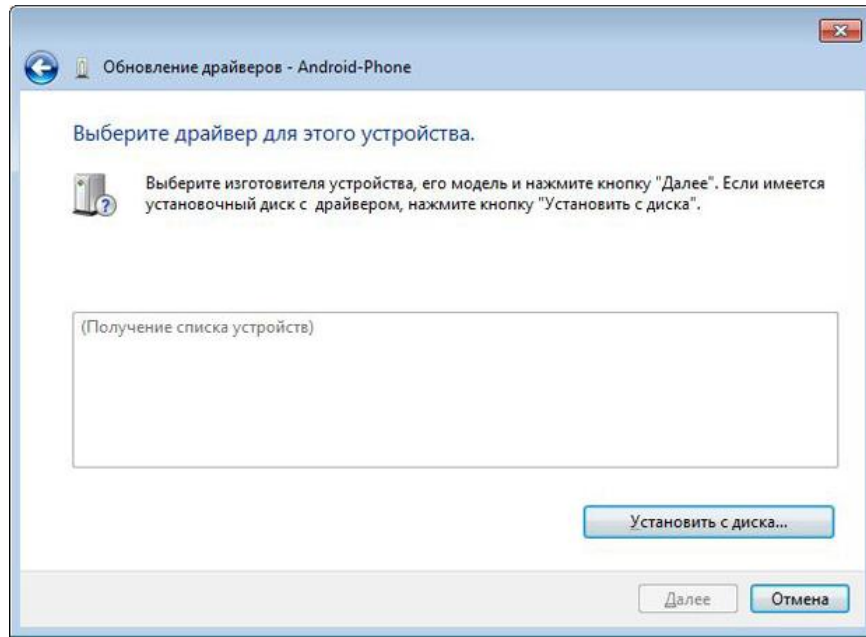


Рис. 31. Выбор драйвера устройства

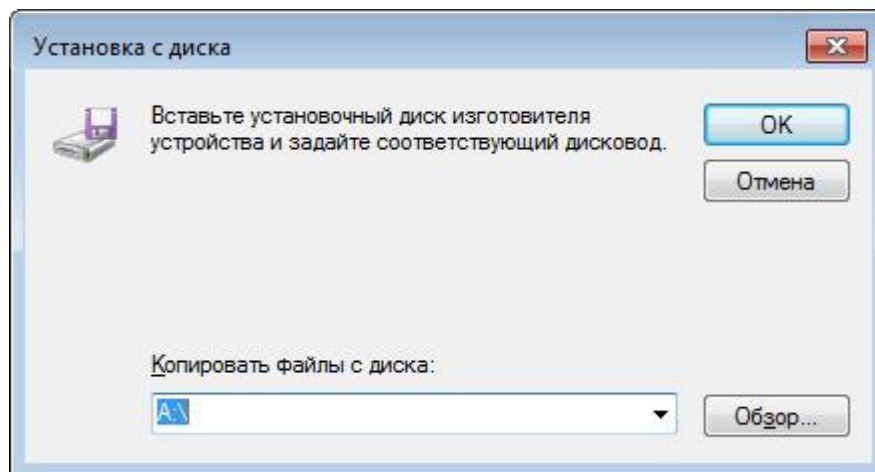


Рис. 32. Поиск драйвера

- Далее выберите файл **android_winusb.inf**

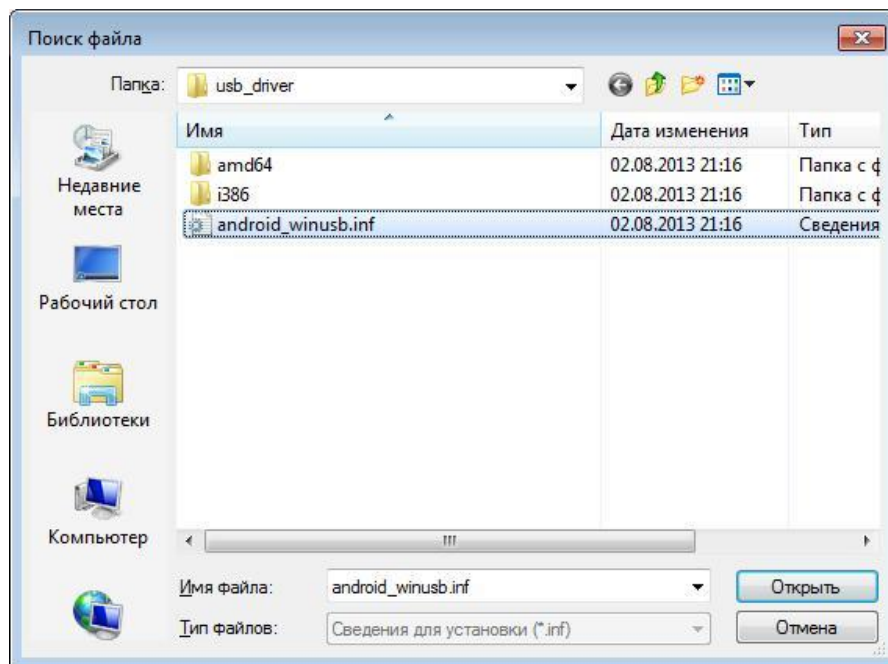


Рис. 33. Сведения для установки

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

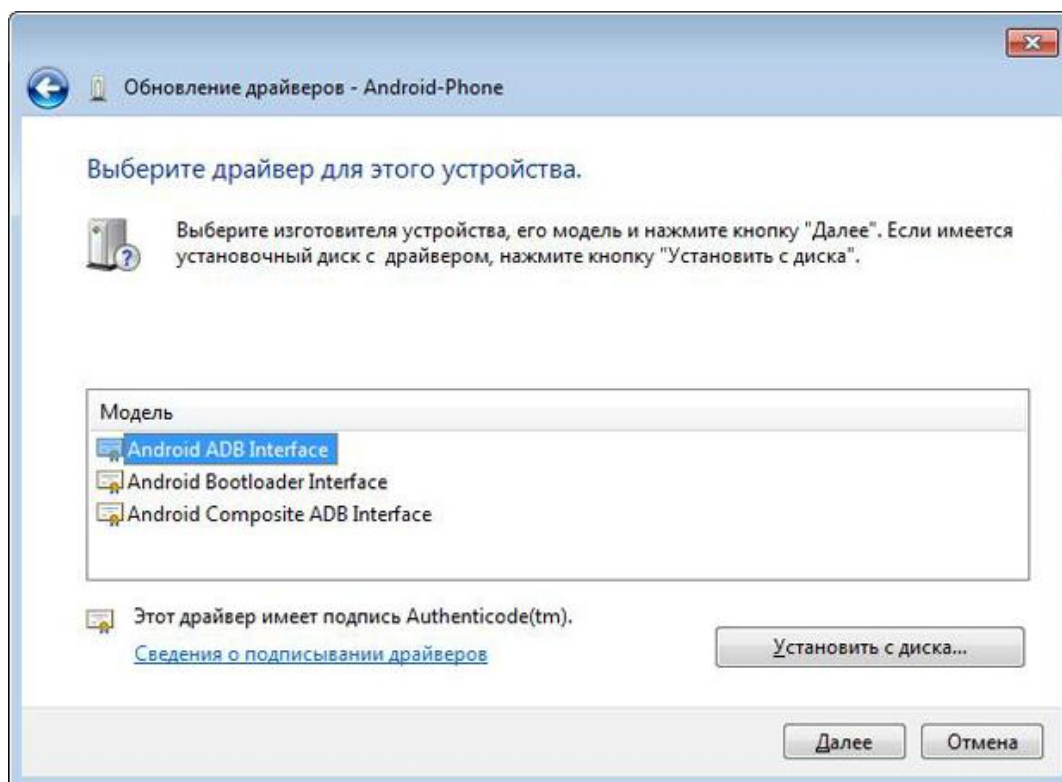


Рис. 34. Завершение установки драйвера

Настройка среды

1. В ADT зайдите в меню **Run\Debug Configurations** и перейдите на вкладку **Target**. Поставьте флажок напротив **Always prompt to pick device**.

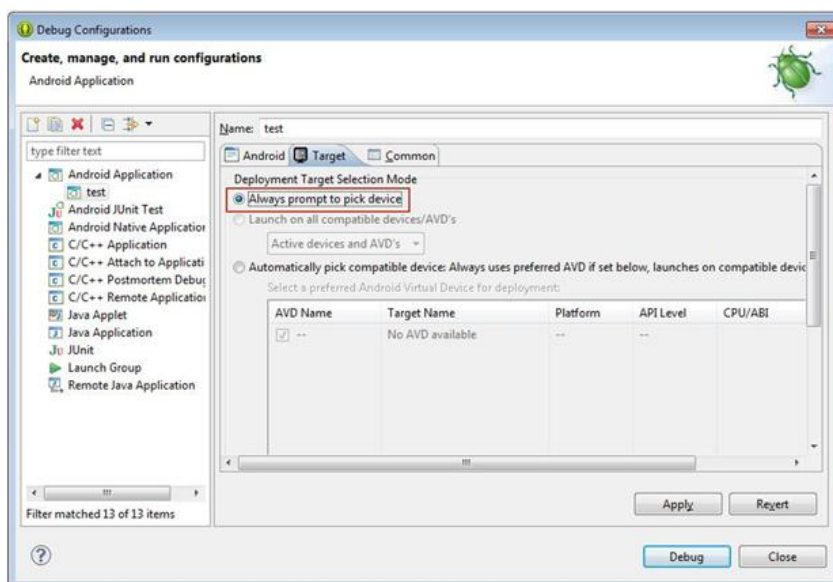


Рис. 35. Настройка среды

2. В открывшемся окне выберите подключенное устройство, поставив флажок напротив **Choose a running Android Device**

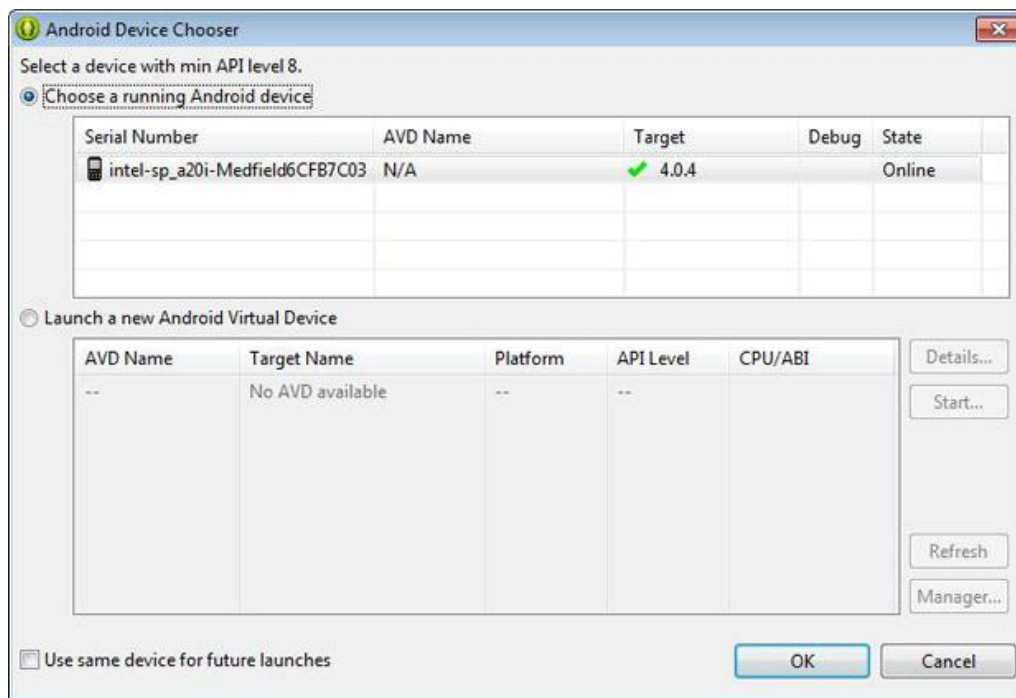


Рис. 36. Выбор устройства

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!*

Лабораторная работа 4. Создание нового проекта. Изучение и комментирование кода

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:

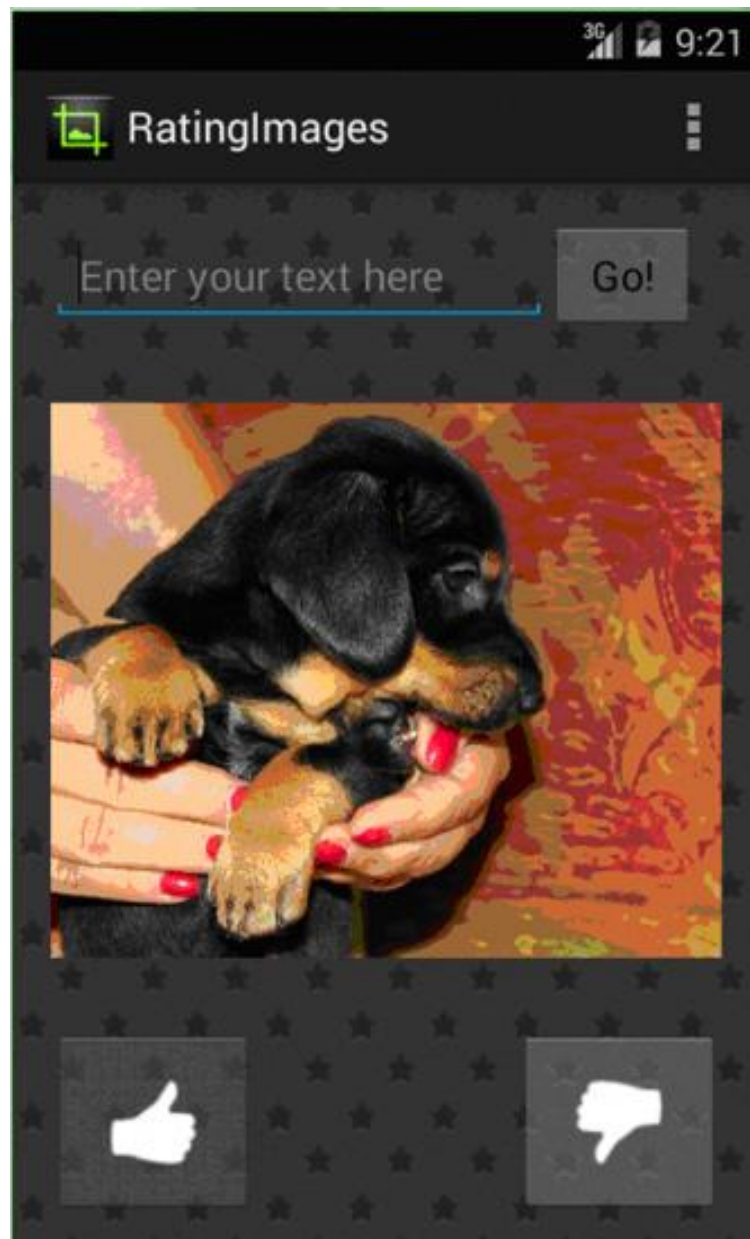


Рис. 6.1. Интерфейс главной активности

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.

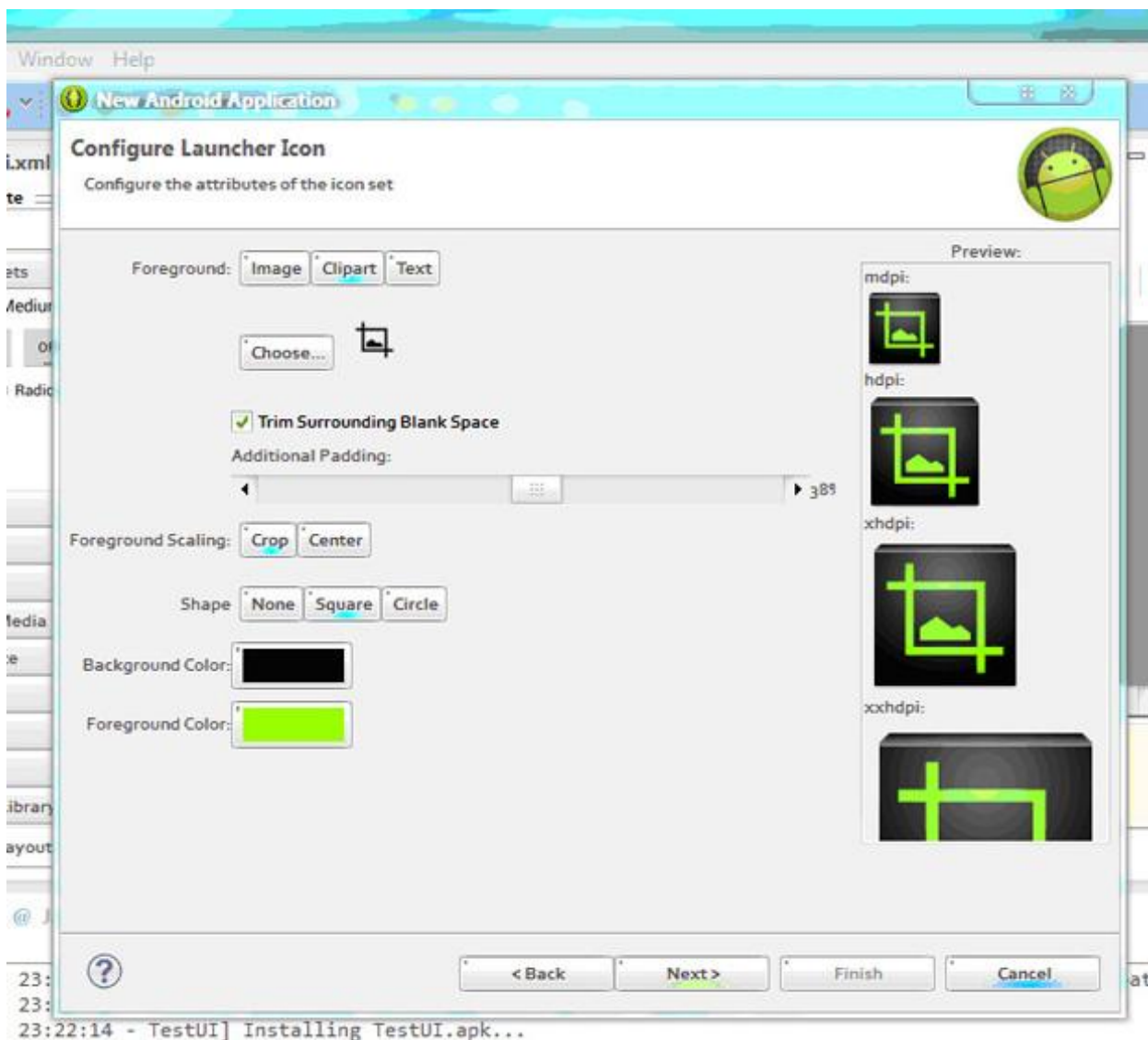


Рис. 6.2. Создание иконки

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:

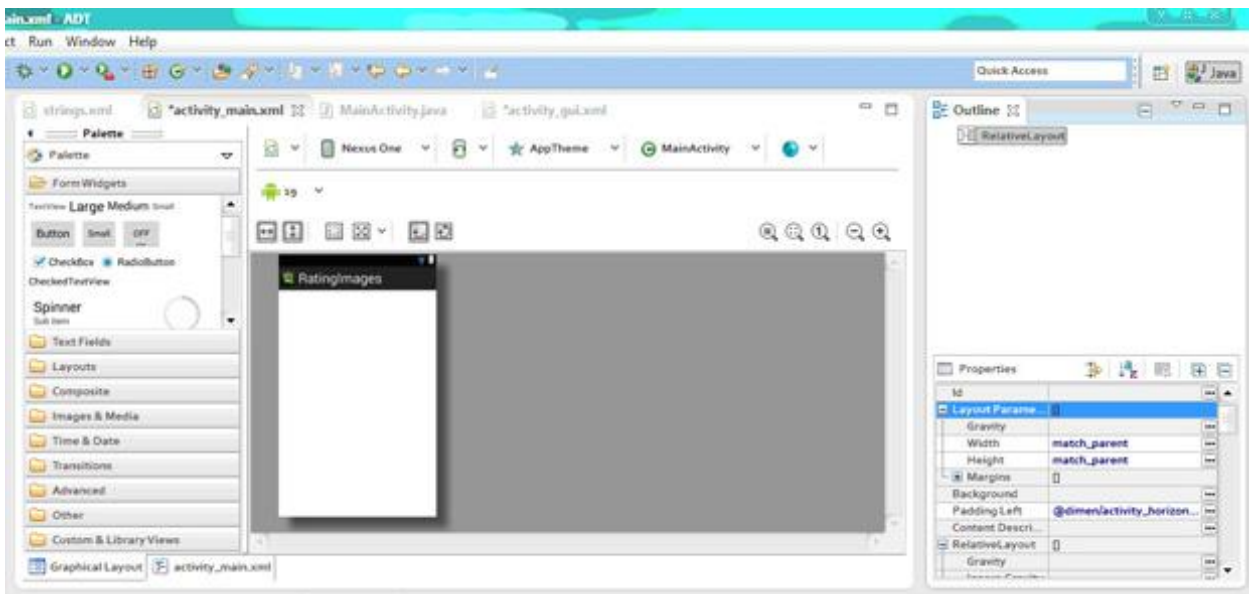


Рис. 6.3. Проект, готовый к началу разработки

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.

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.

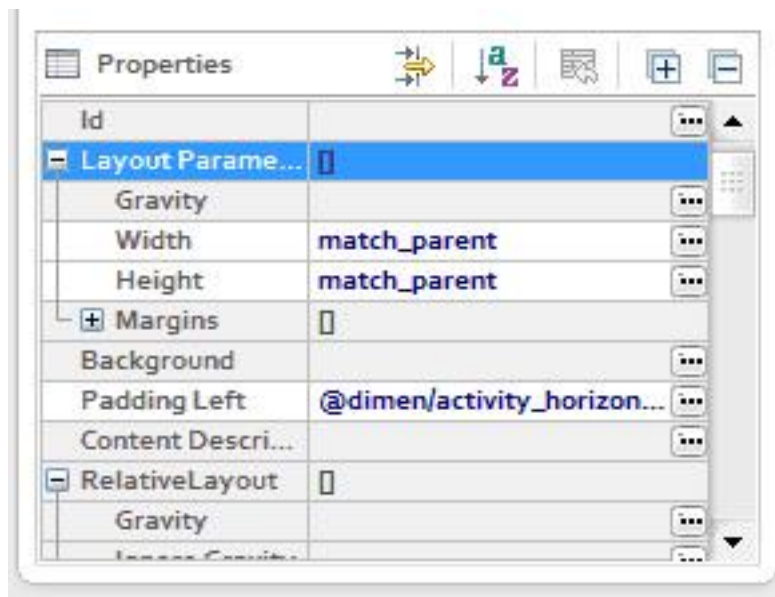


Рис. 6.4. Свойства <RelativeLayout> элемента

Как установить это значение? Будем разбираться.

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

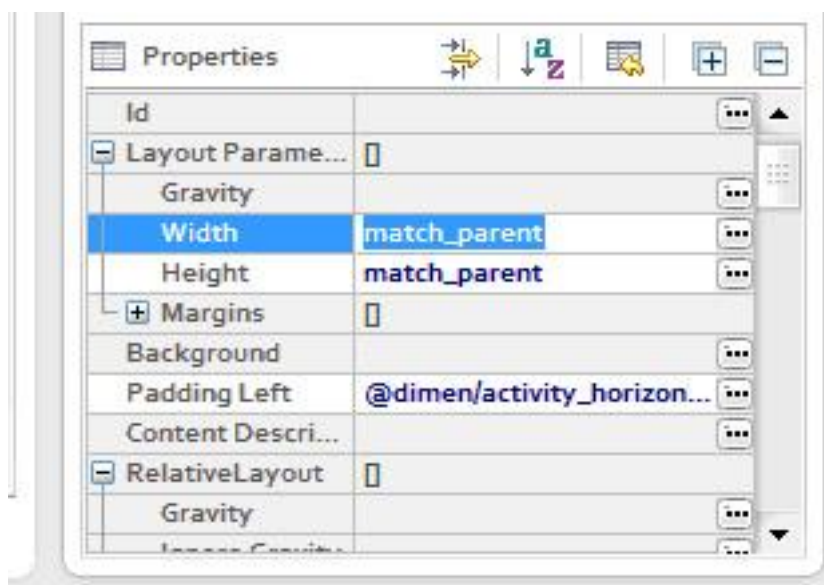


Рис. 6.5. Свойства <RelativeLayout> элемента, выбран атрибут android:layout_width

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

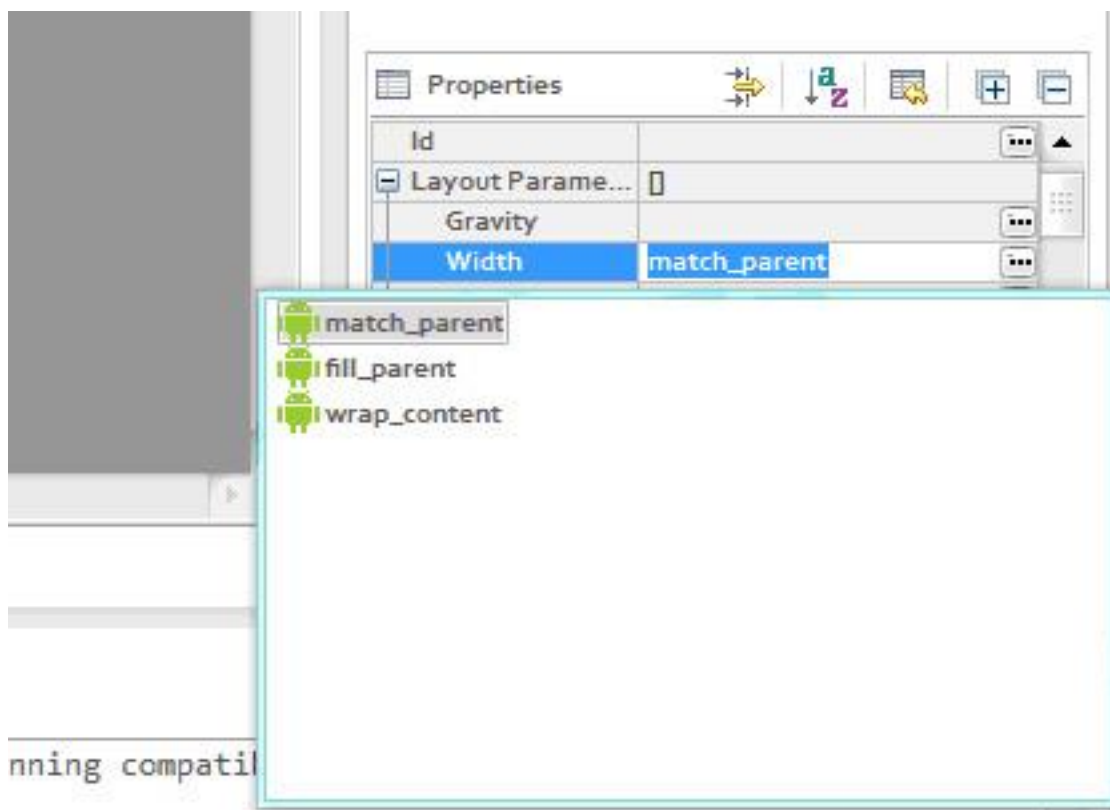


Рис. 6.6. Свойства `<RelativeLayout>` элемента, выбран параметр "match_parent"

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

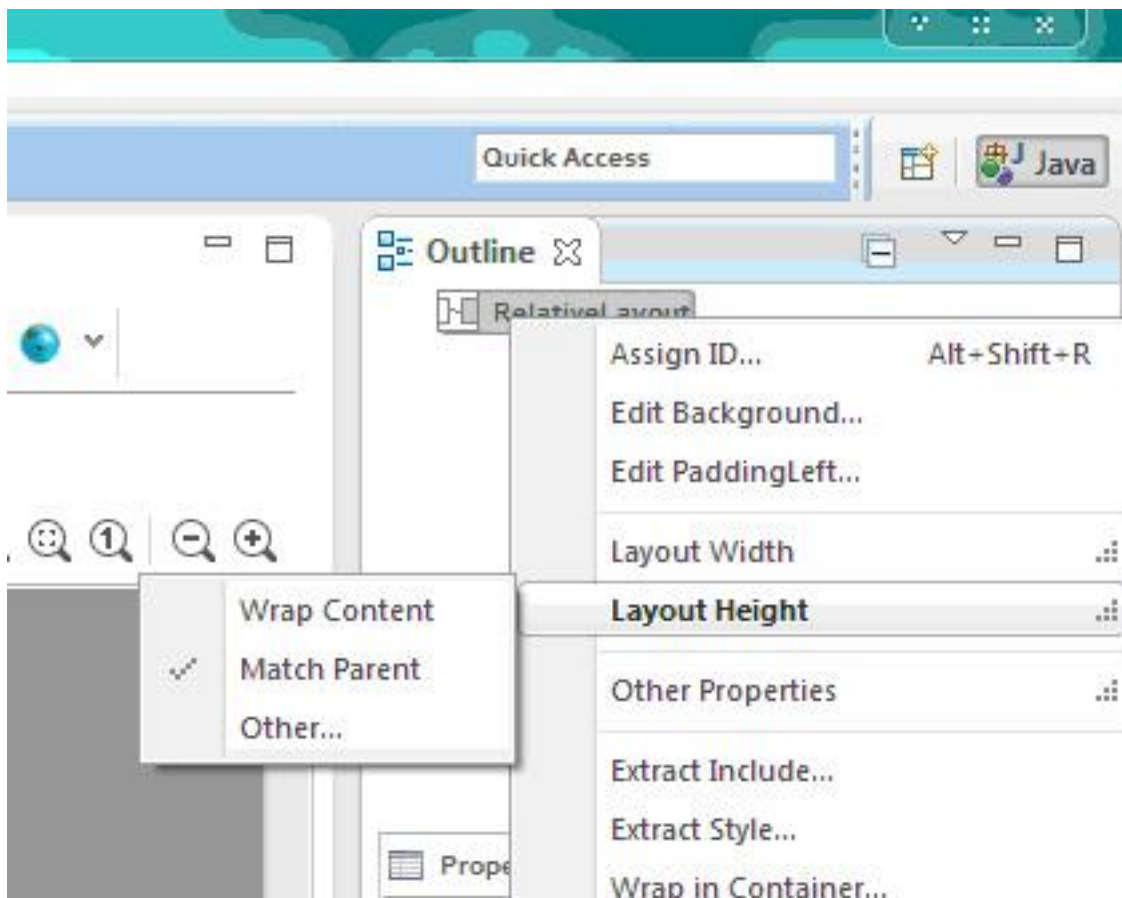


Рис. 6.7. Контекстное меню `<RelativeLayout>` элемента

При выполнении первого способа вы увидите 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:

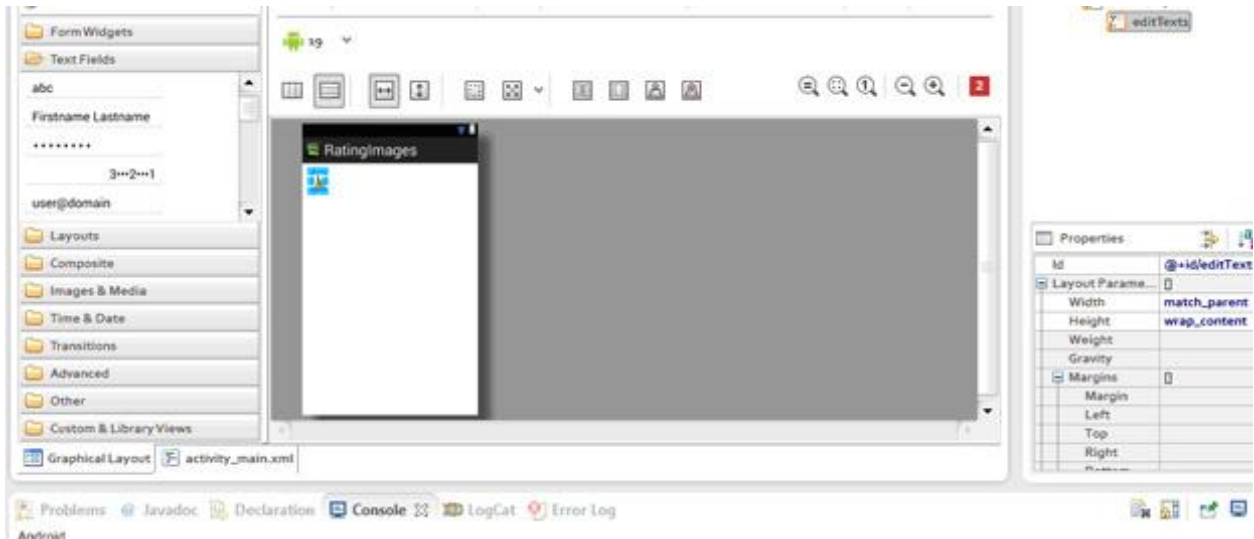


Рис. 6.8. Добавление текстового поля

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:

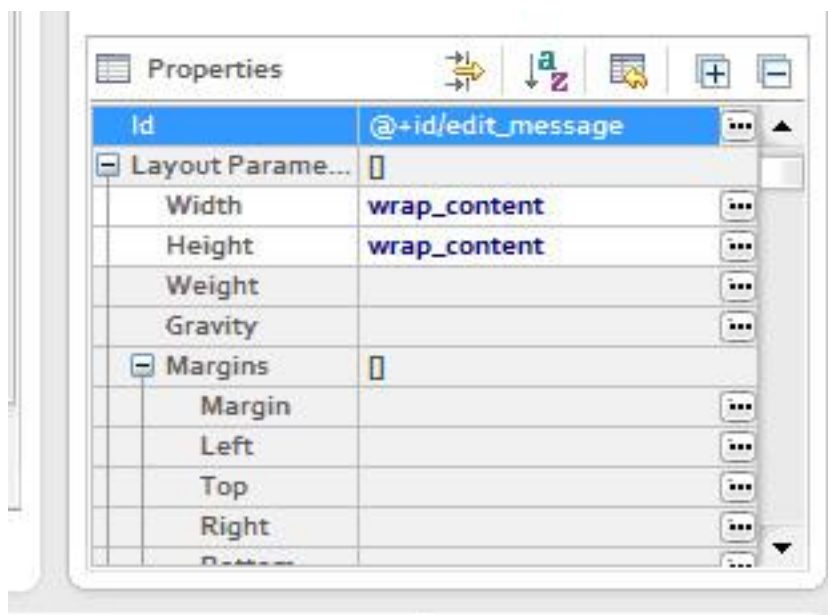


Рис. 6.9. Идентификатор текстового поля

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.

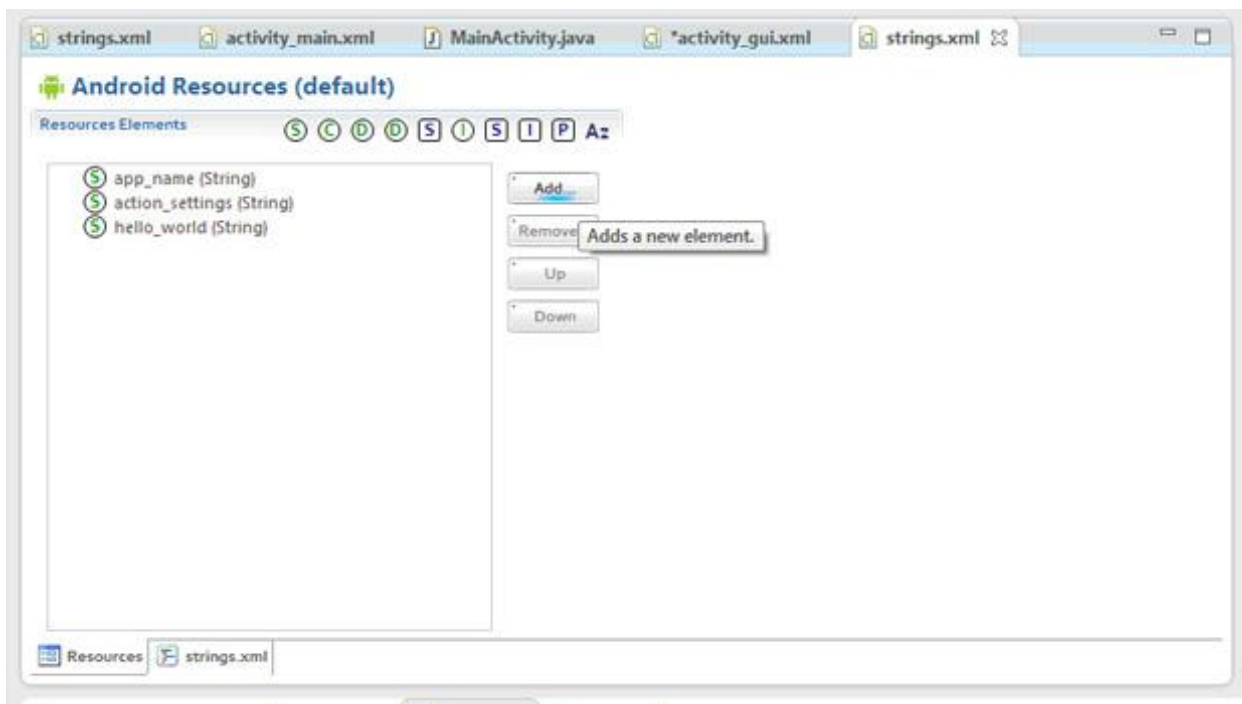


Рис. 6.11. Редактирование файла ресурсов графическим способом; добавление нового ресурса

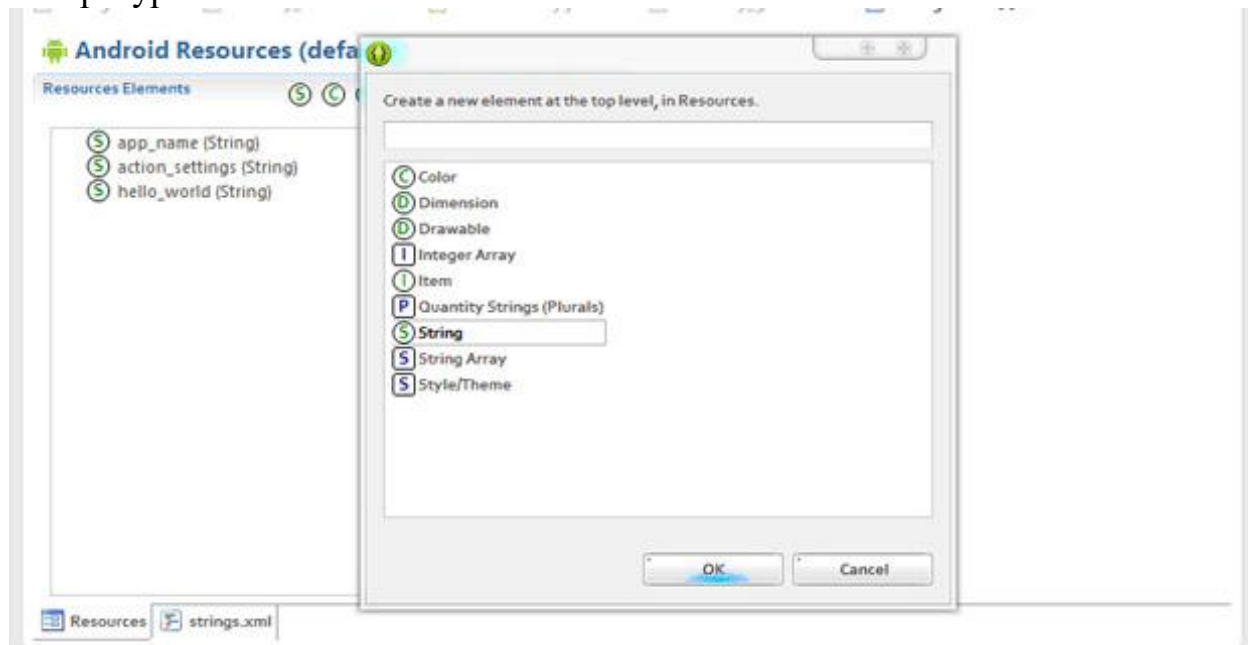


Рис. 6.12. Редактирование файла ресурсов графическим способом; выбор типа ресурса, выбран тип String

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

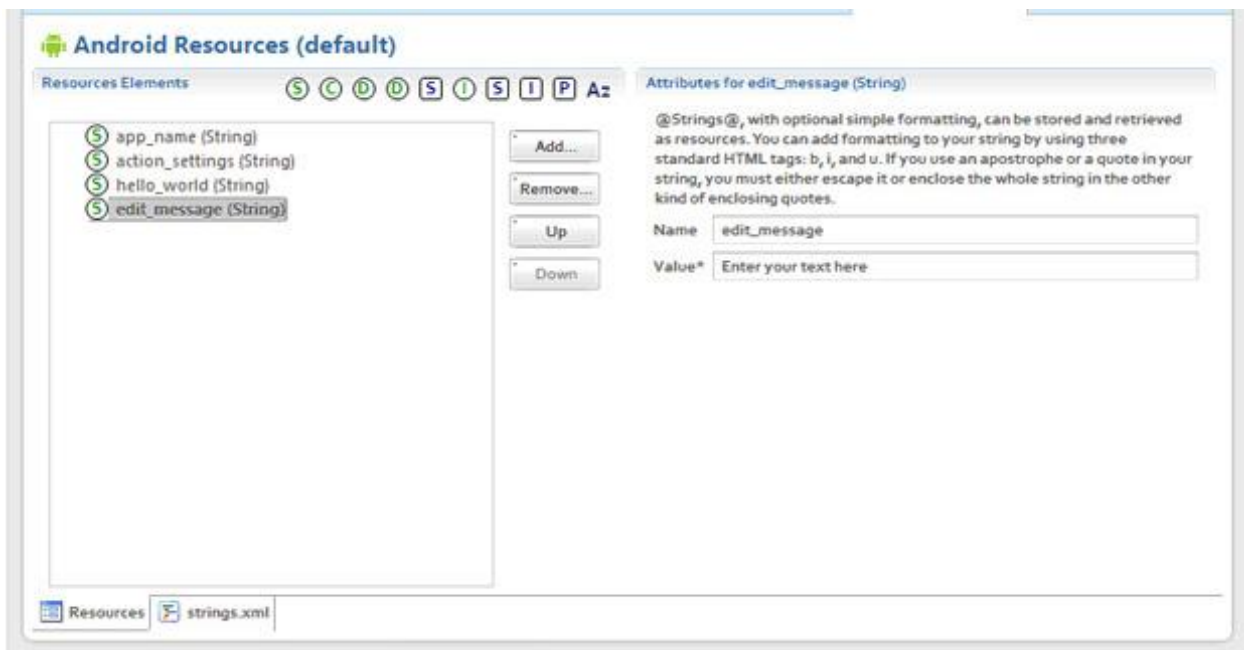


Рис. 6.13. Редактирование файла ресурсов графическим способом

Сохраняем и любуемся результатом:



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

Аналогично создадим id для <LinearLayout>

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

Сохраните изменения.

6.2.3 Добавление кнопки

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

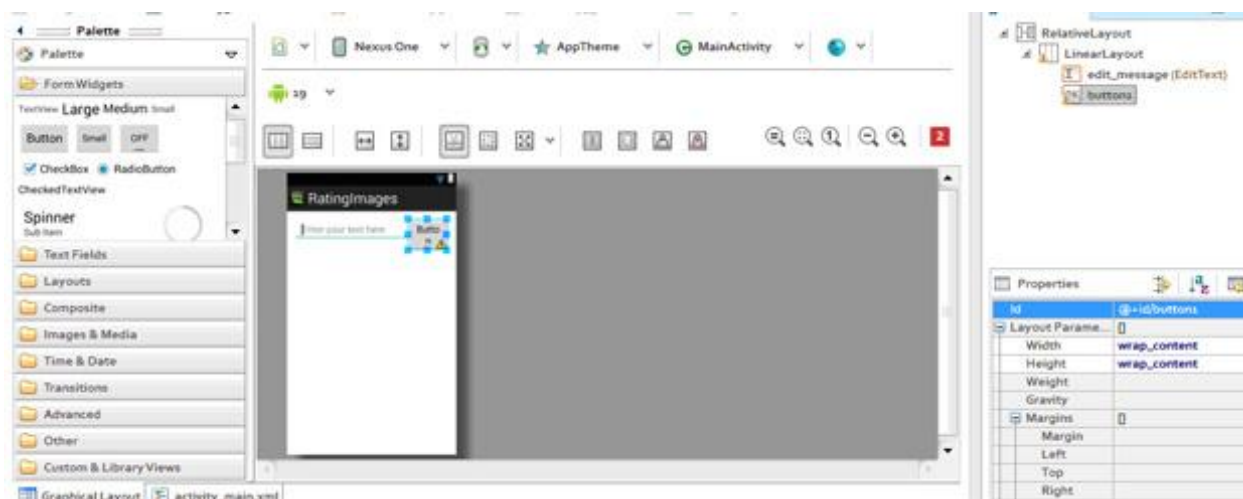


Рис. 6.15. Новая кнопка

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!

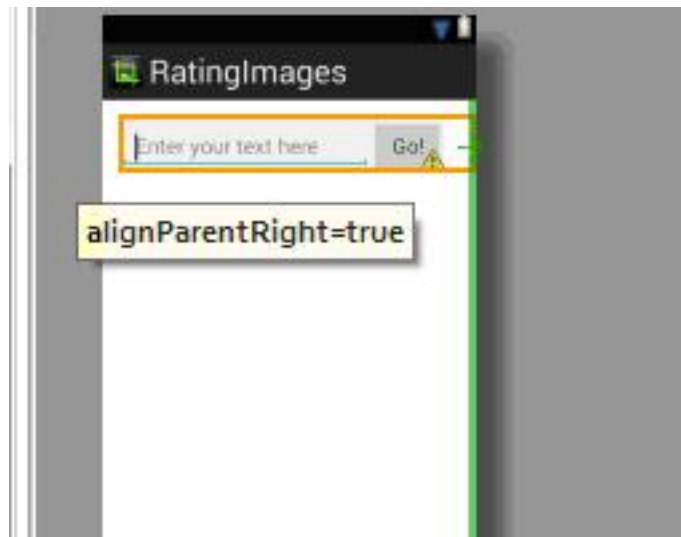


Рис. 6.18. "Приращение" правого края

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

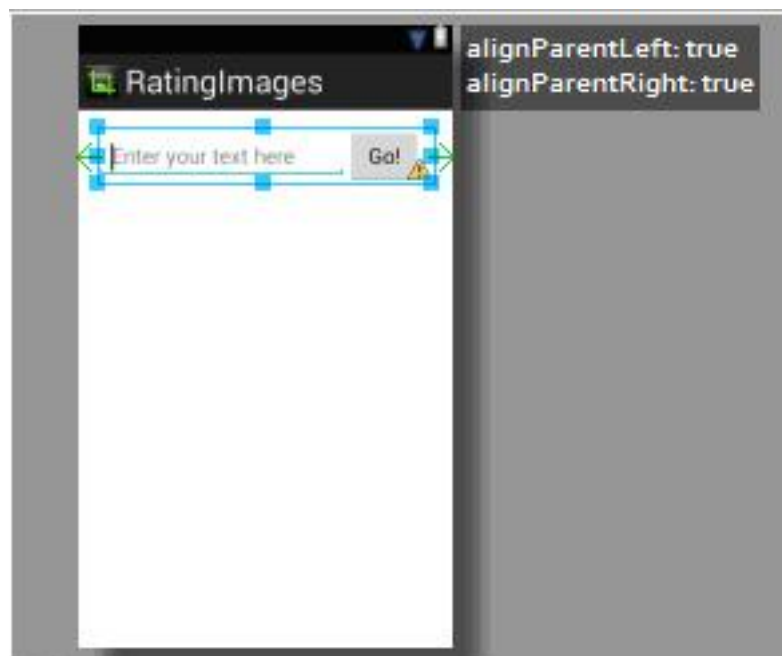


Рис. 6.19. <LinearLayout> после выравнивания

И, конечно, это отобразится в свойствах:

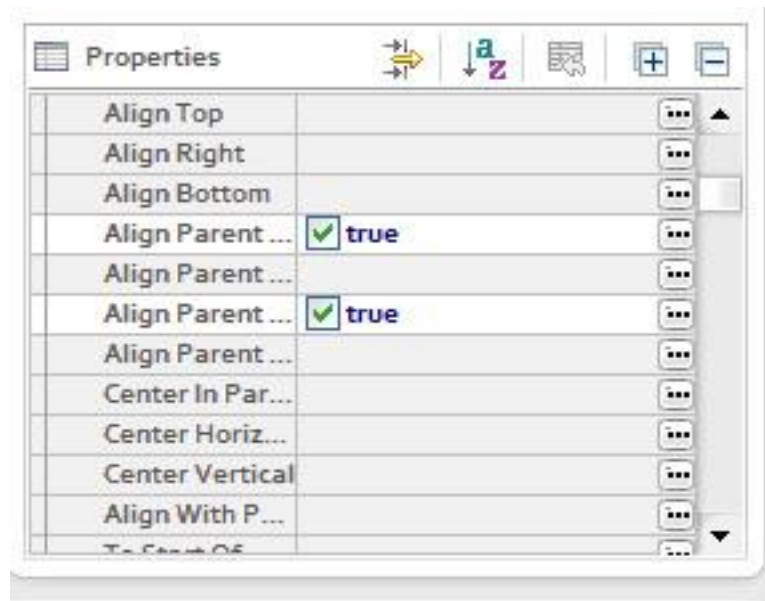


Рис. 6.20. Свойства <LinearLayout>

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.

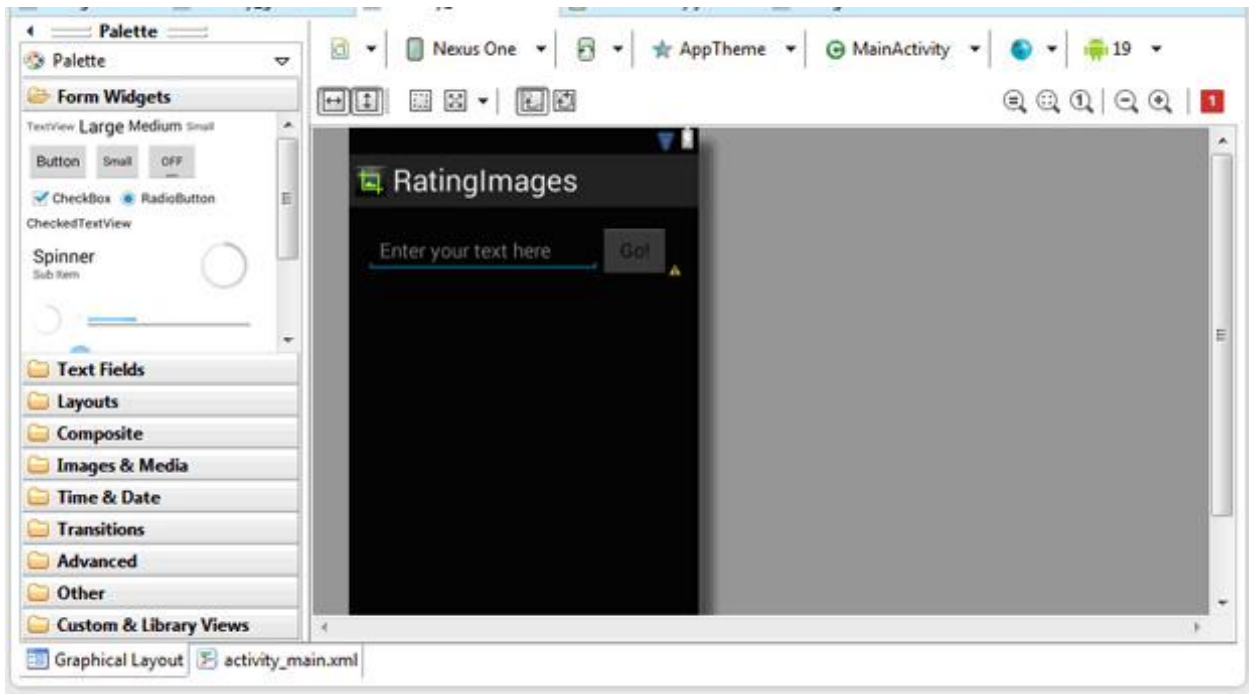


Рис. 6.21. Фон стал чёрным

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

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

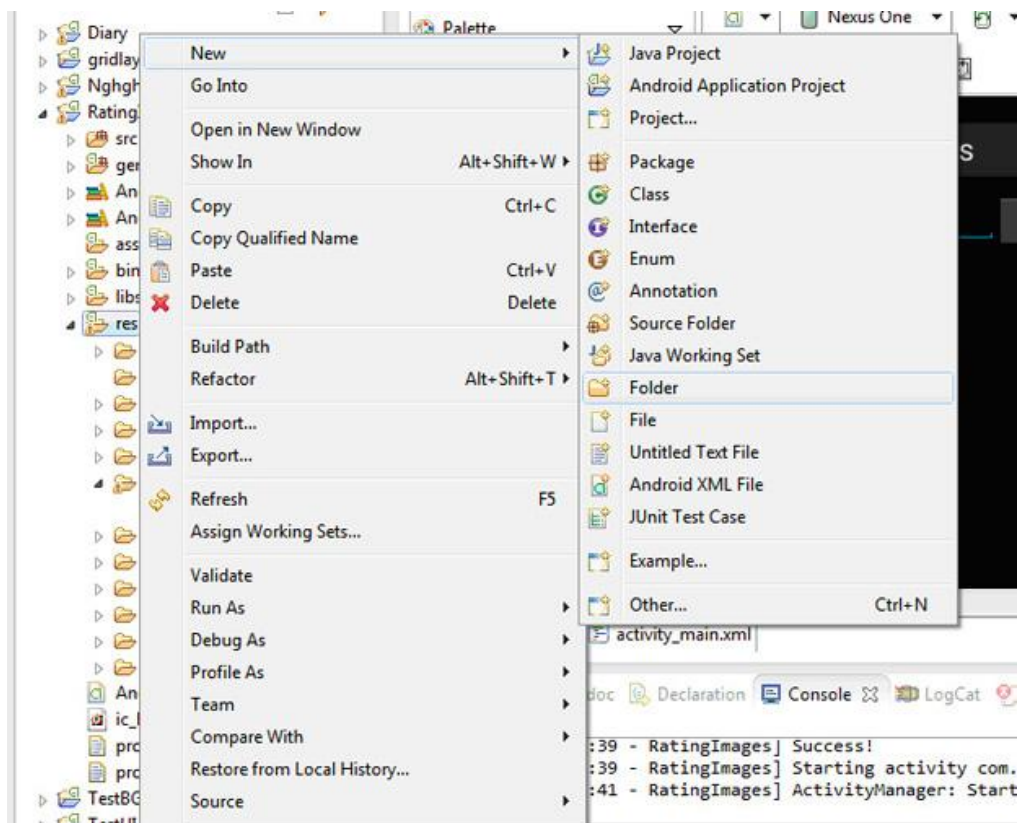


Рис. 6.22. Создание папки

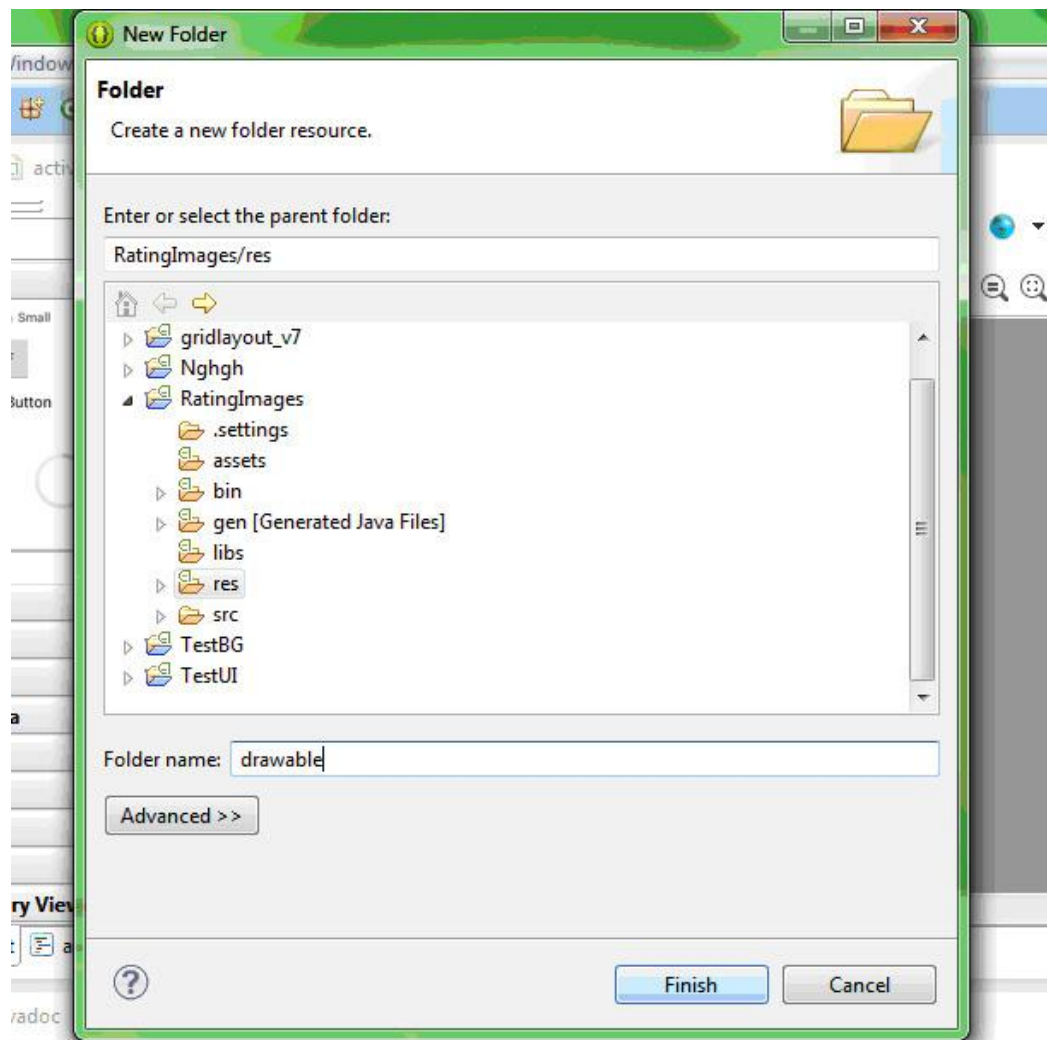


Рис. 6.23. Создание папки: имя папки

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

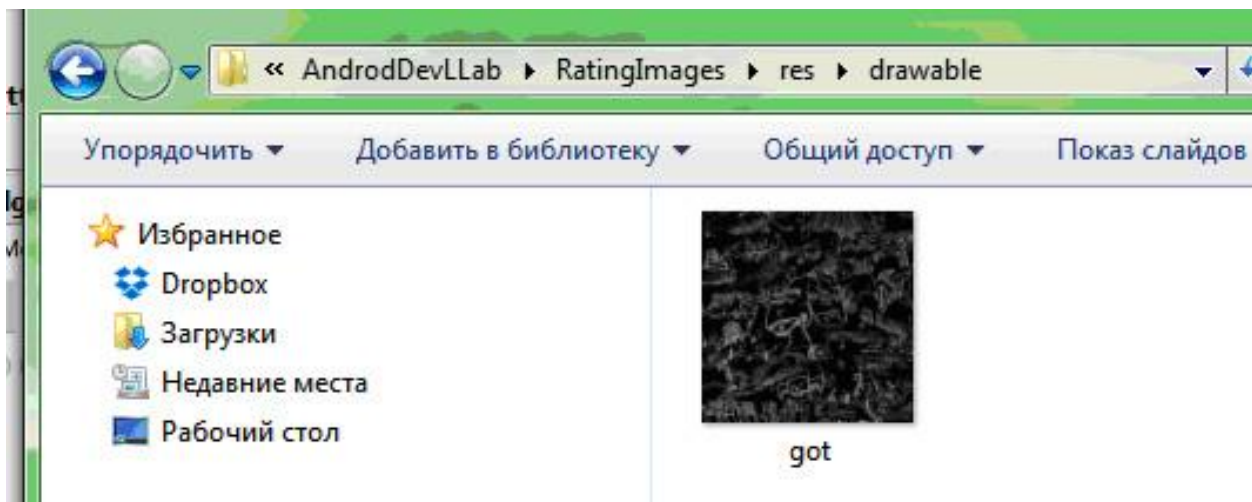


Рис. 6.24. Изображение в папке drawable/

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

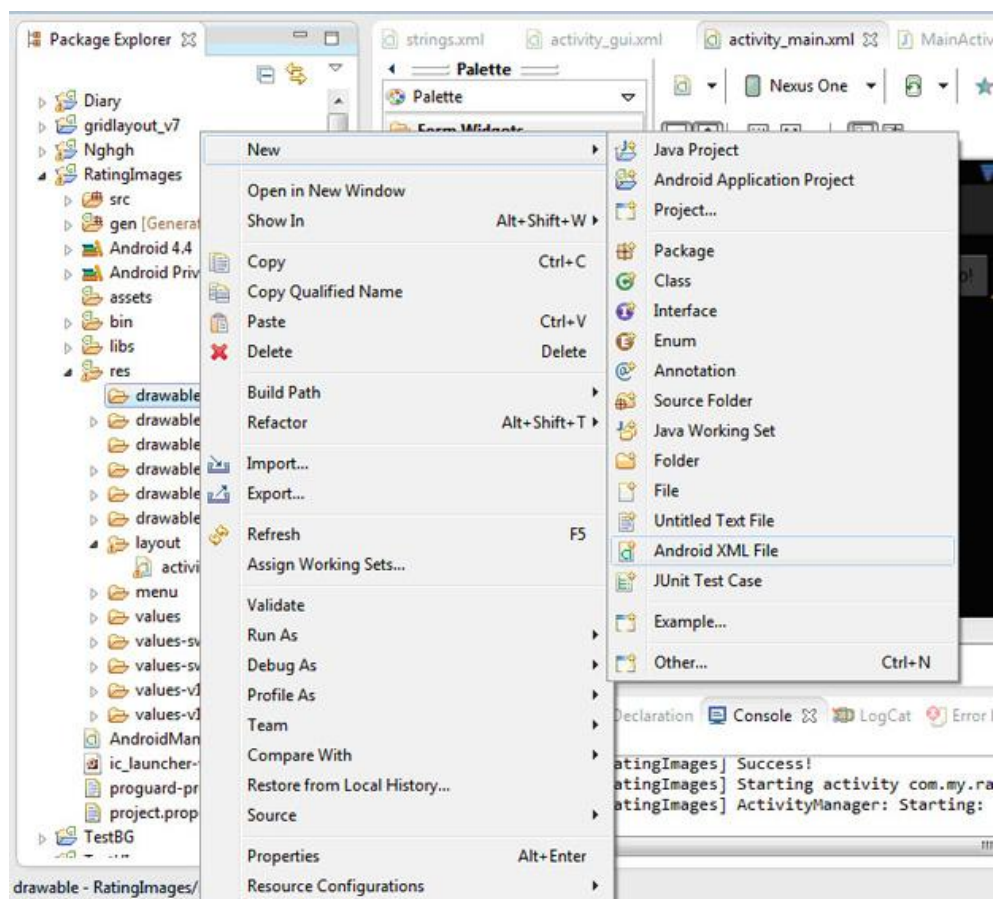


Рис. 6.25. Создание нового XML-файла

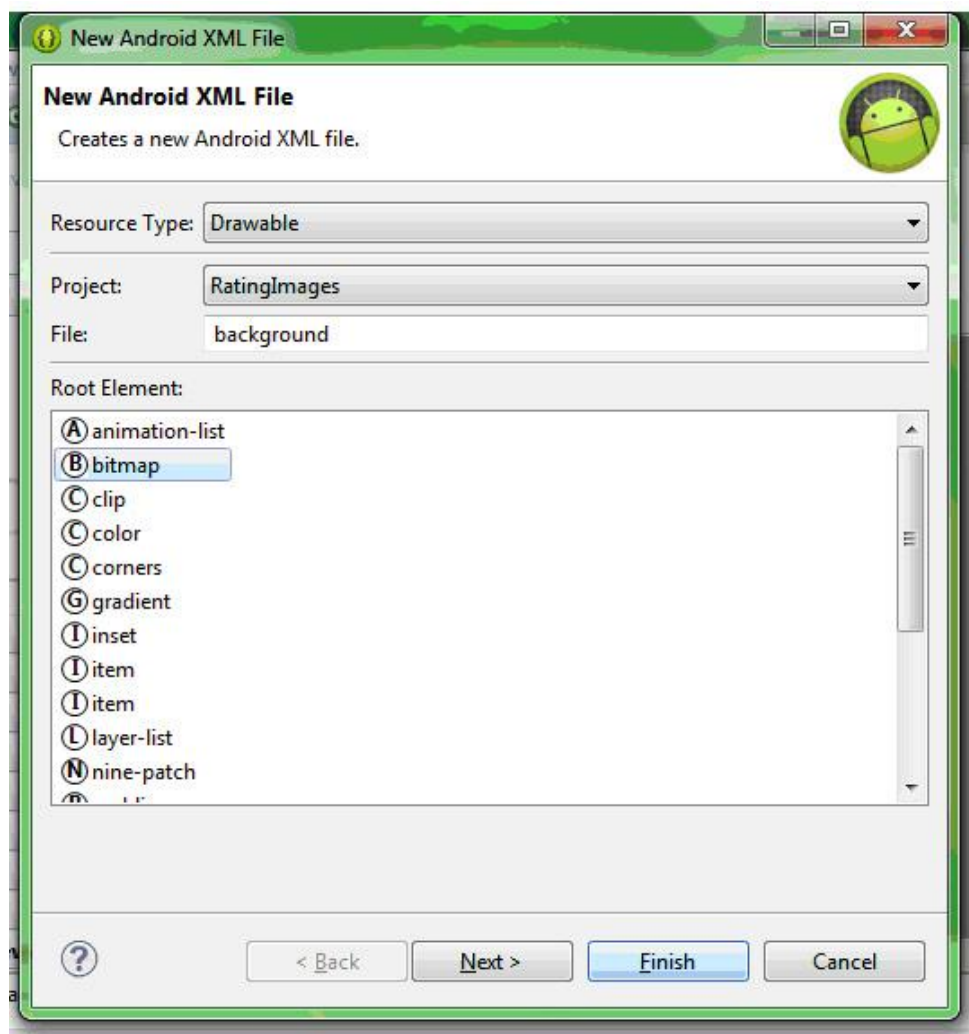


Рис. 6.26. Создание нового XML-файла

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:

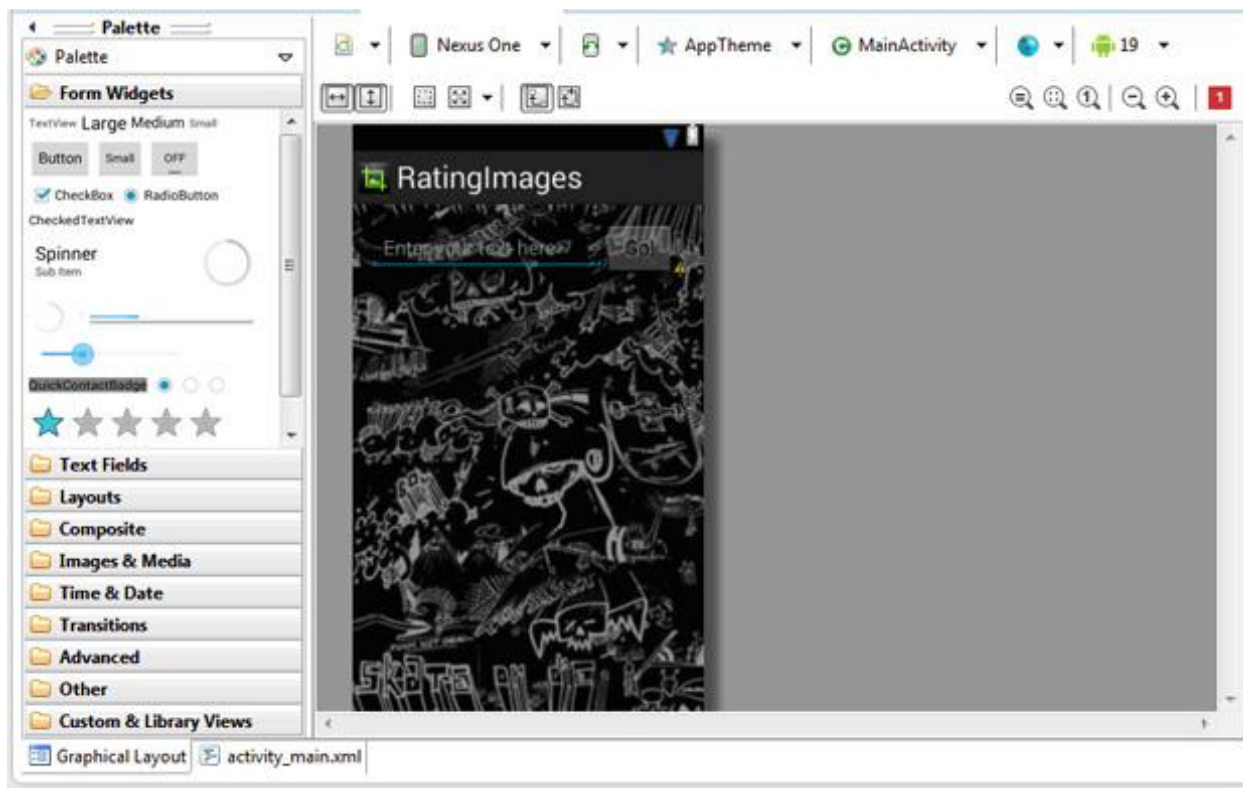


Рис. 6.27. Новый фон

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/**.

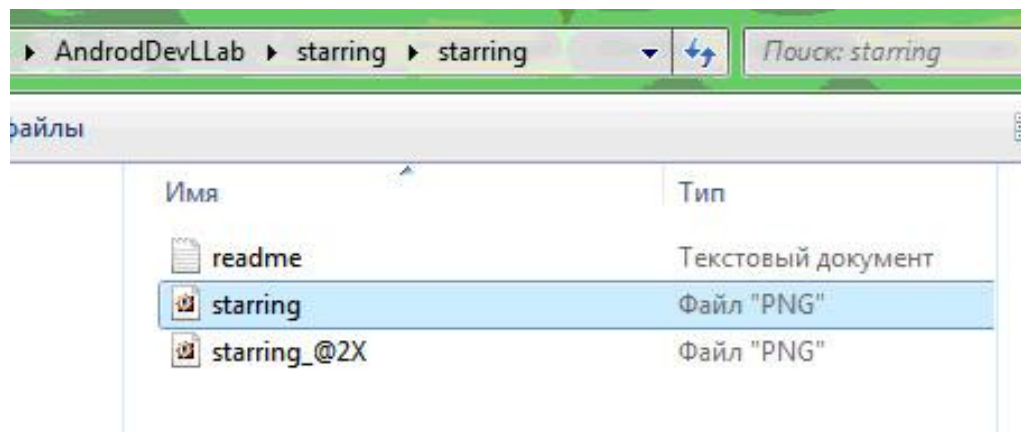


Рис. 6.28. Копирование изображения

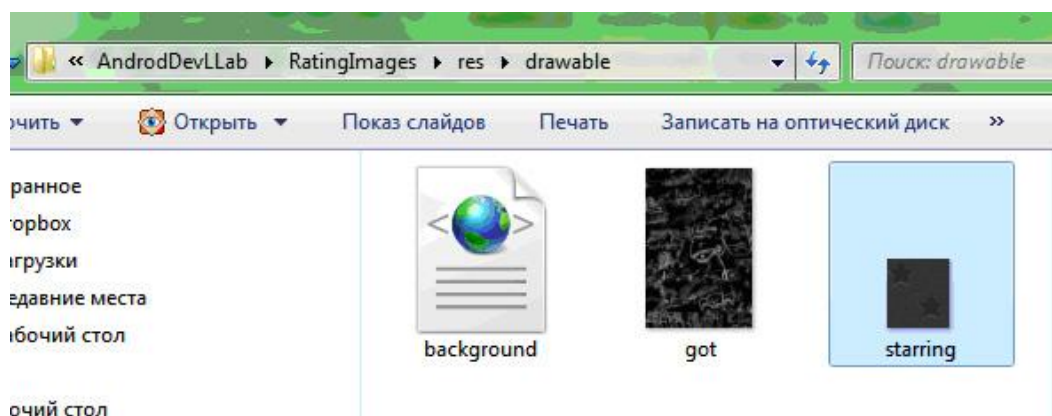


Рис. 6.29. Скопированное изображение

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

Во-первых, нужно изменить имя изображения 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.

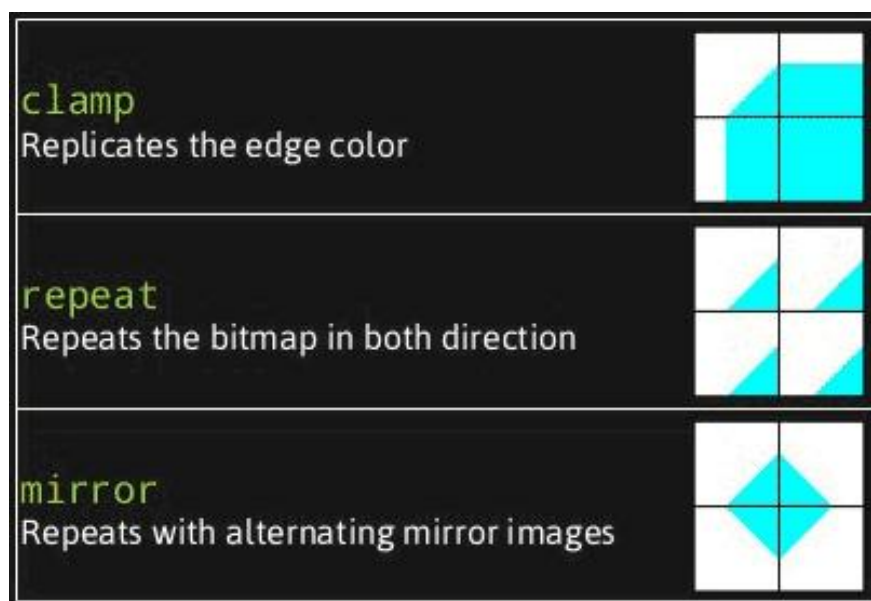


Рис. 6.30. Варианты заполнения

Настало время посмотреть, что из этого получилось:

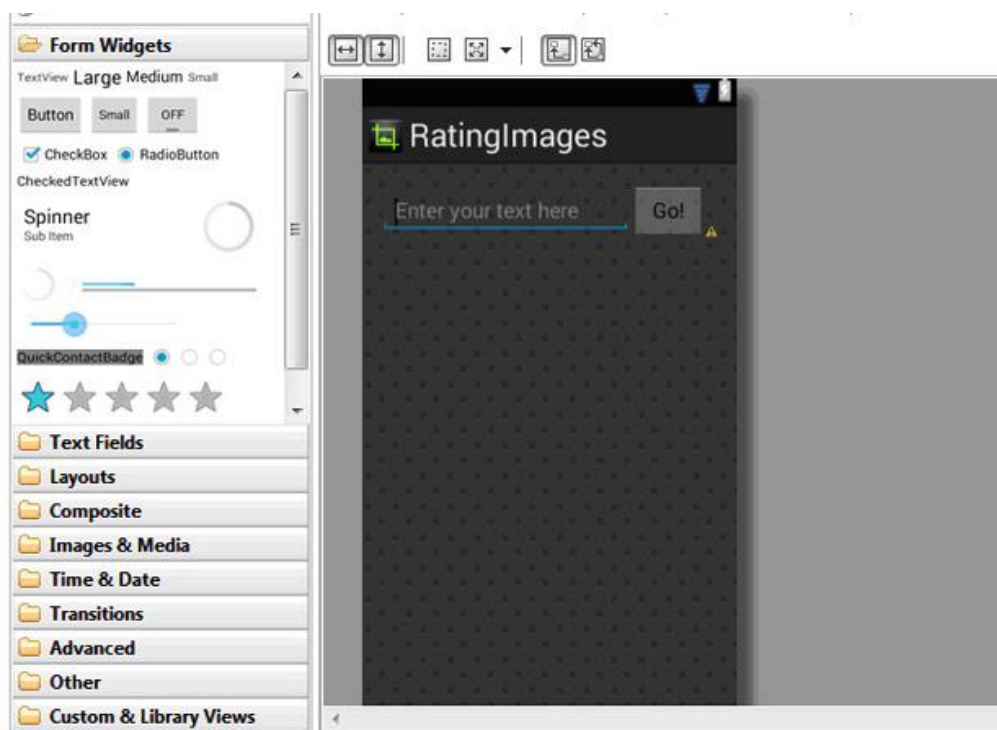


Рис. 6.31. Фон из звездочек

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>`:

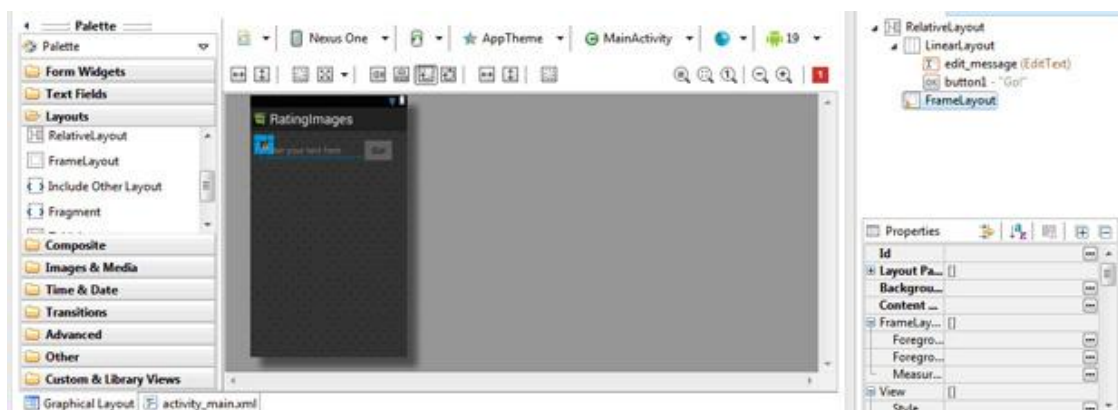


Рис. 6.32. Добавление "рамки"

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.

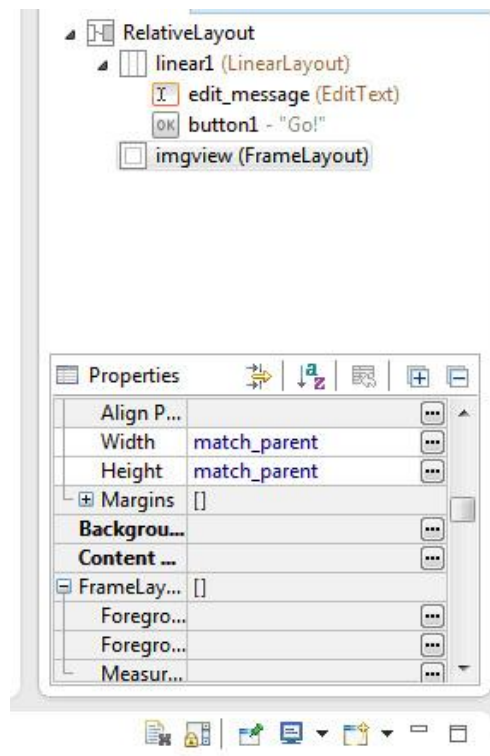


Рис. 6.33. Свойства "рамки"

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...**

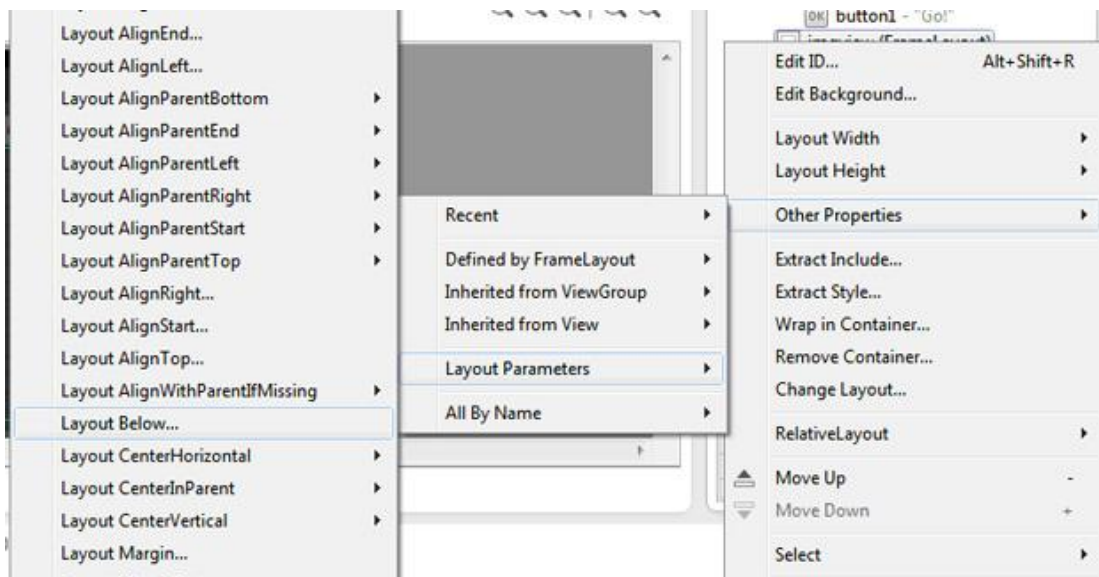


Рис. 6.34. Контекстное меню "рамки"

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

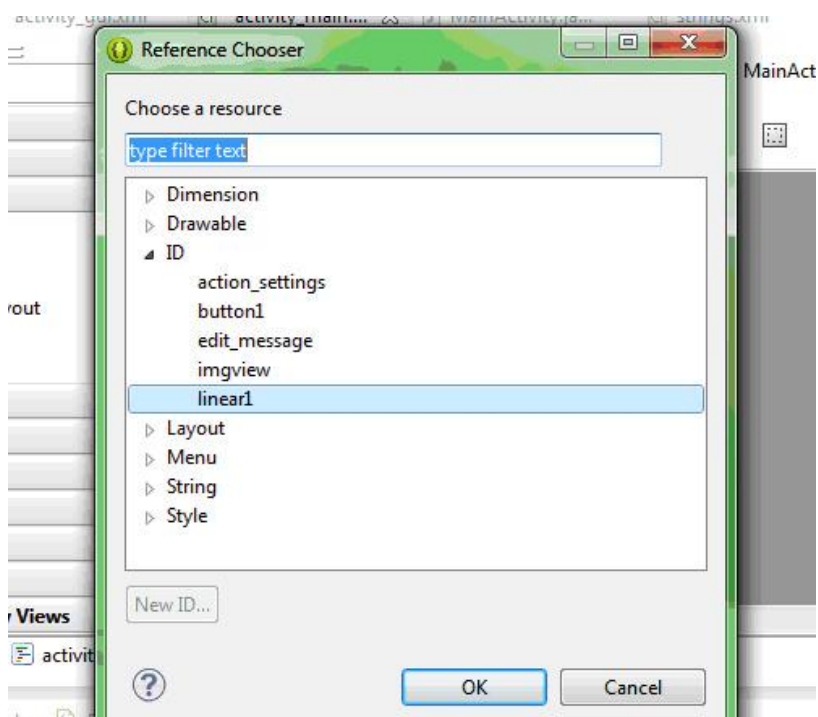


Рис. 6.35. Выбор элемента, ниже которого должна быть "рамка"

Жмём **ОК** и "рамка" примет вид:

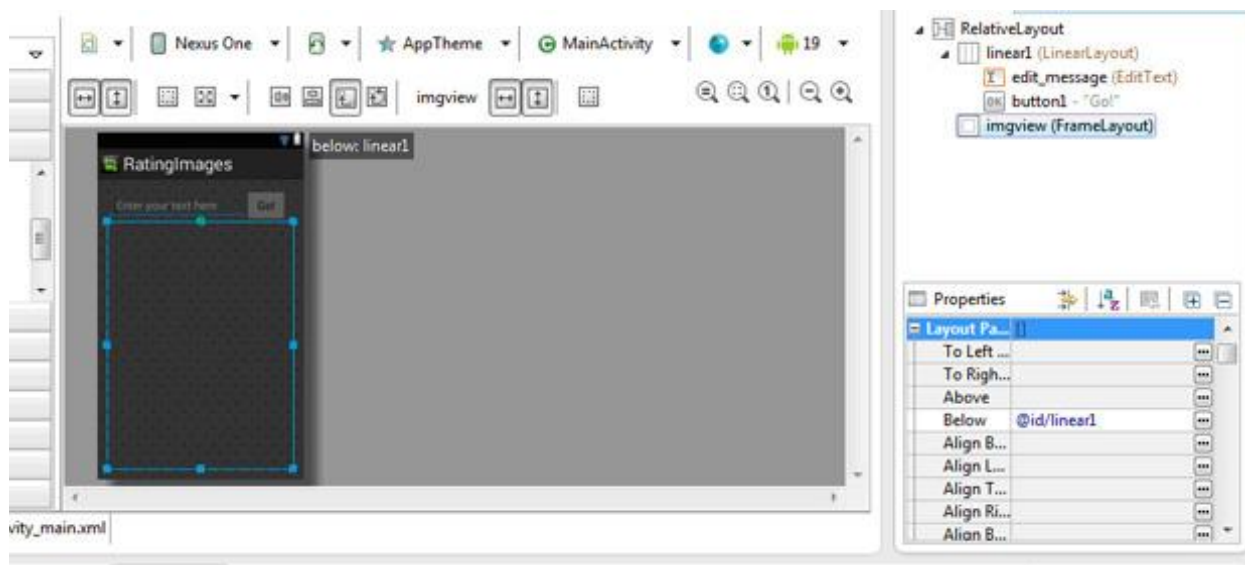


Рис. 6.36. Новое расположение "рамки"

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.

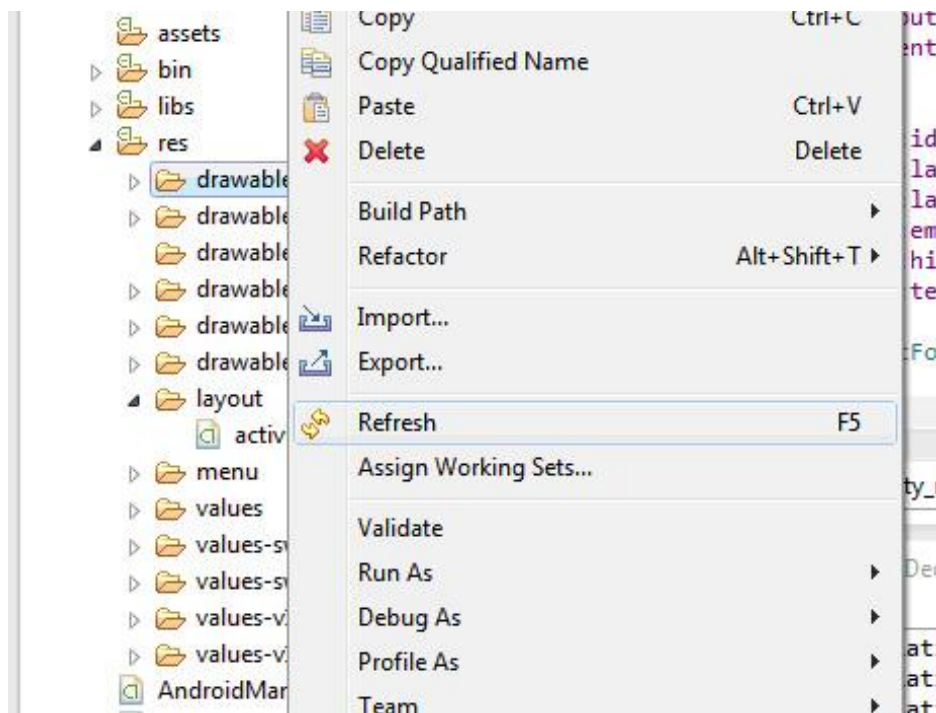


Рис. 6.37. Обновление папки

Теперь поместим внутрь "рамки" <ImageView>

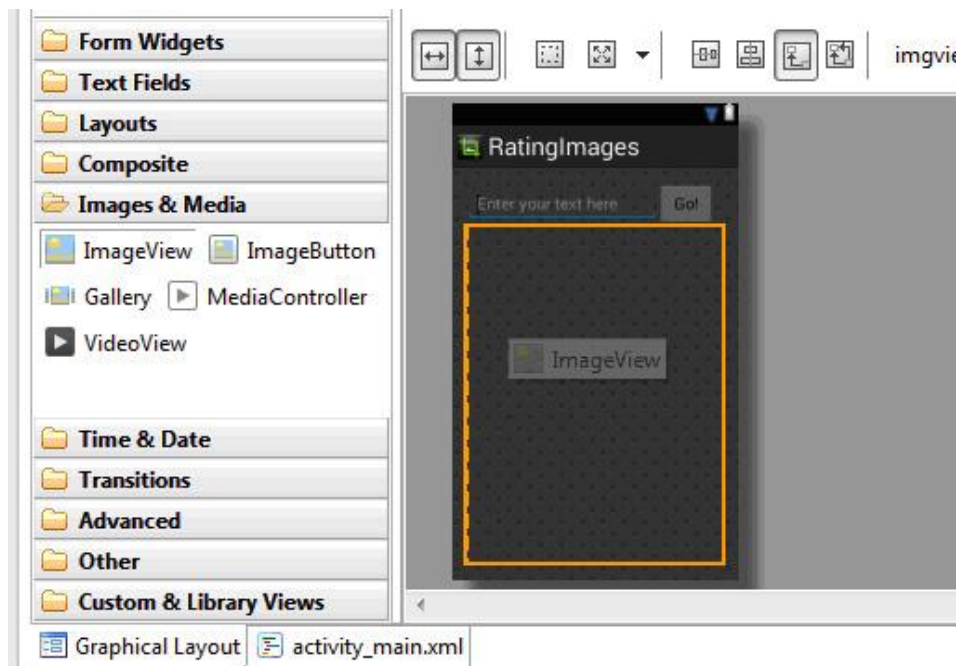


Рис. 6.38. Перемещение <ImageView>

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

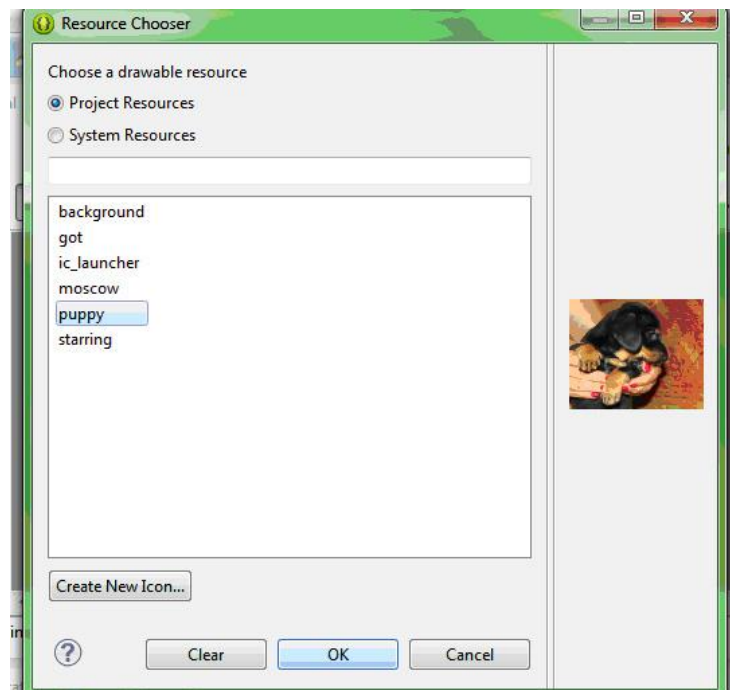


Рис. 6.39. Выбор файла

Изображение должно появиться на активности:

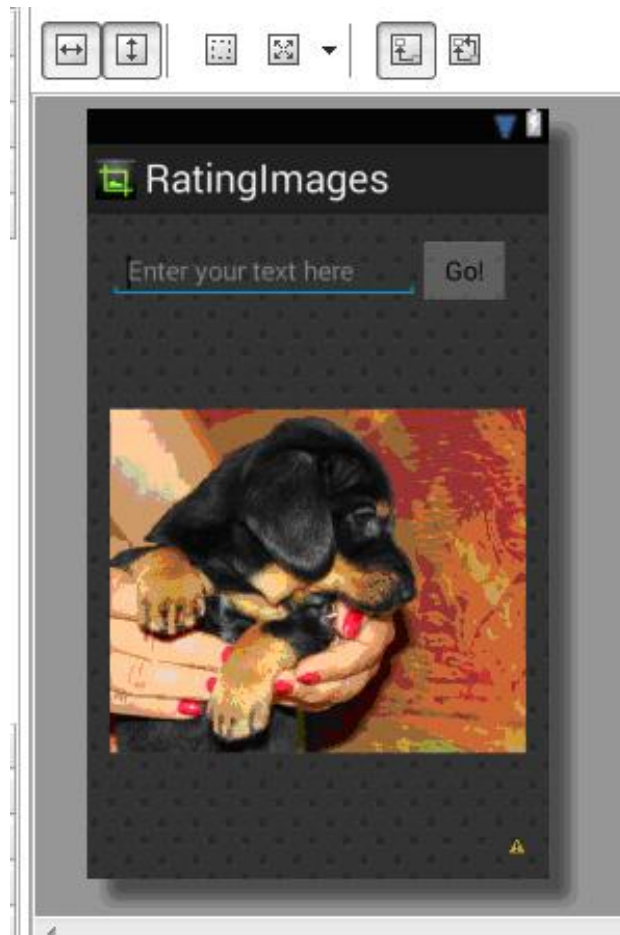


Рис. 6.40. Визуализация изображения

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:

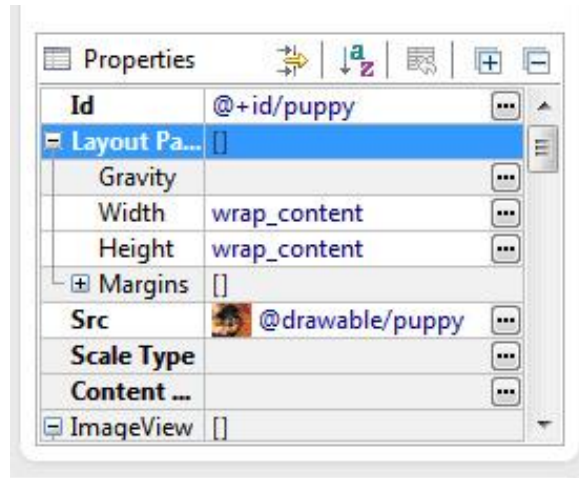


Рис. 6.41. Свойства изображения

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:

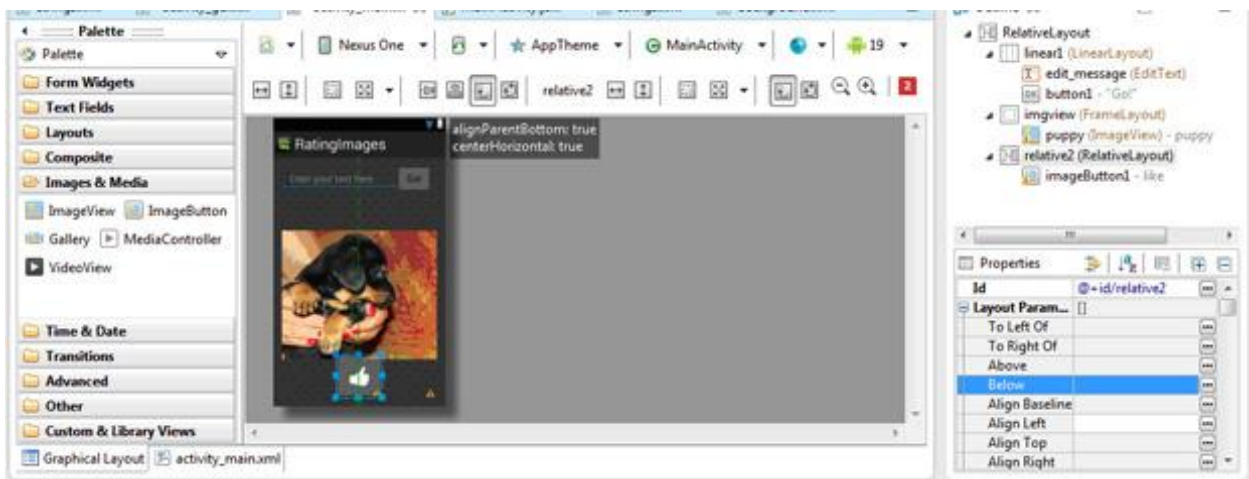


Рис. 6.42. Новая кнопка

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

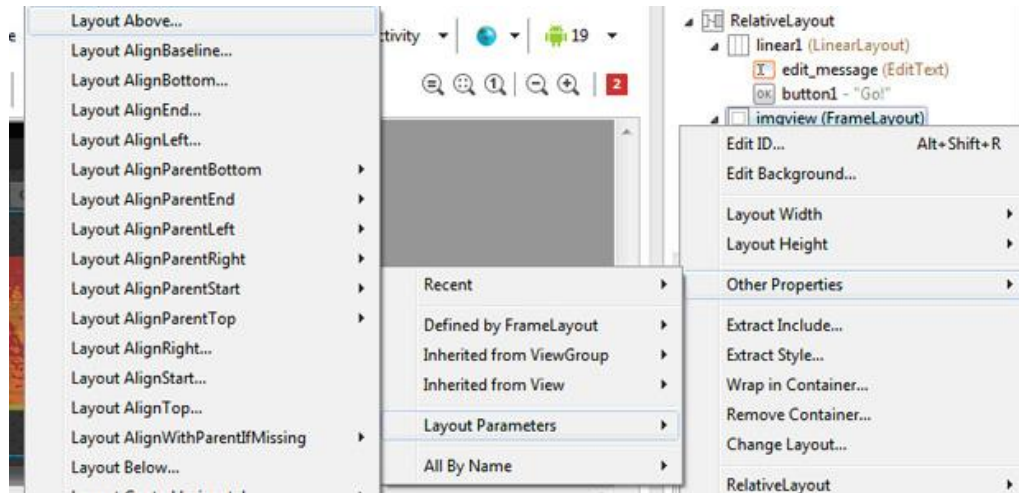


Рис. 6.43. Контекстное меню "рамки"

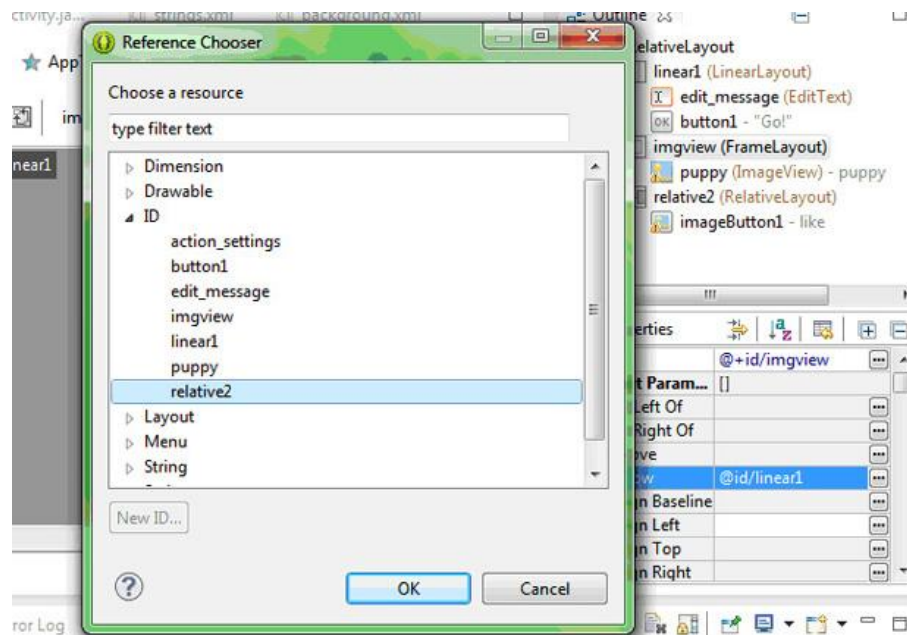


Рис. 6.44. Выбор элемента, выше которого должна быть "рамка"

В результате должно получиться следующее:

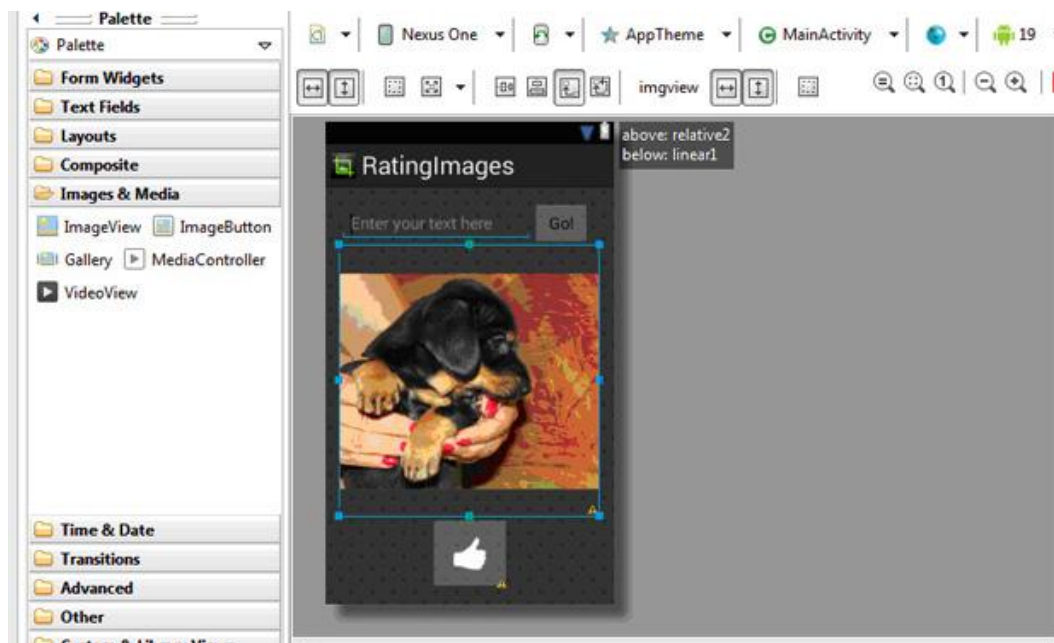


Рис. 6.45. Новое расположение элементов

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



Рис. 6.46. Выравнивание области с кнопками

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

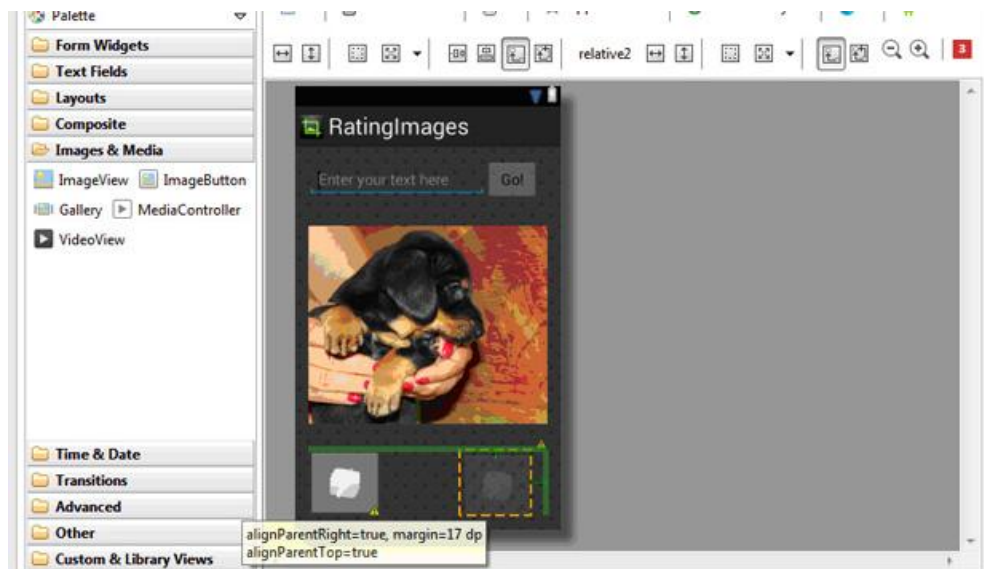


Рис. 6.47. Выравнивание кнопок

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

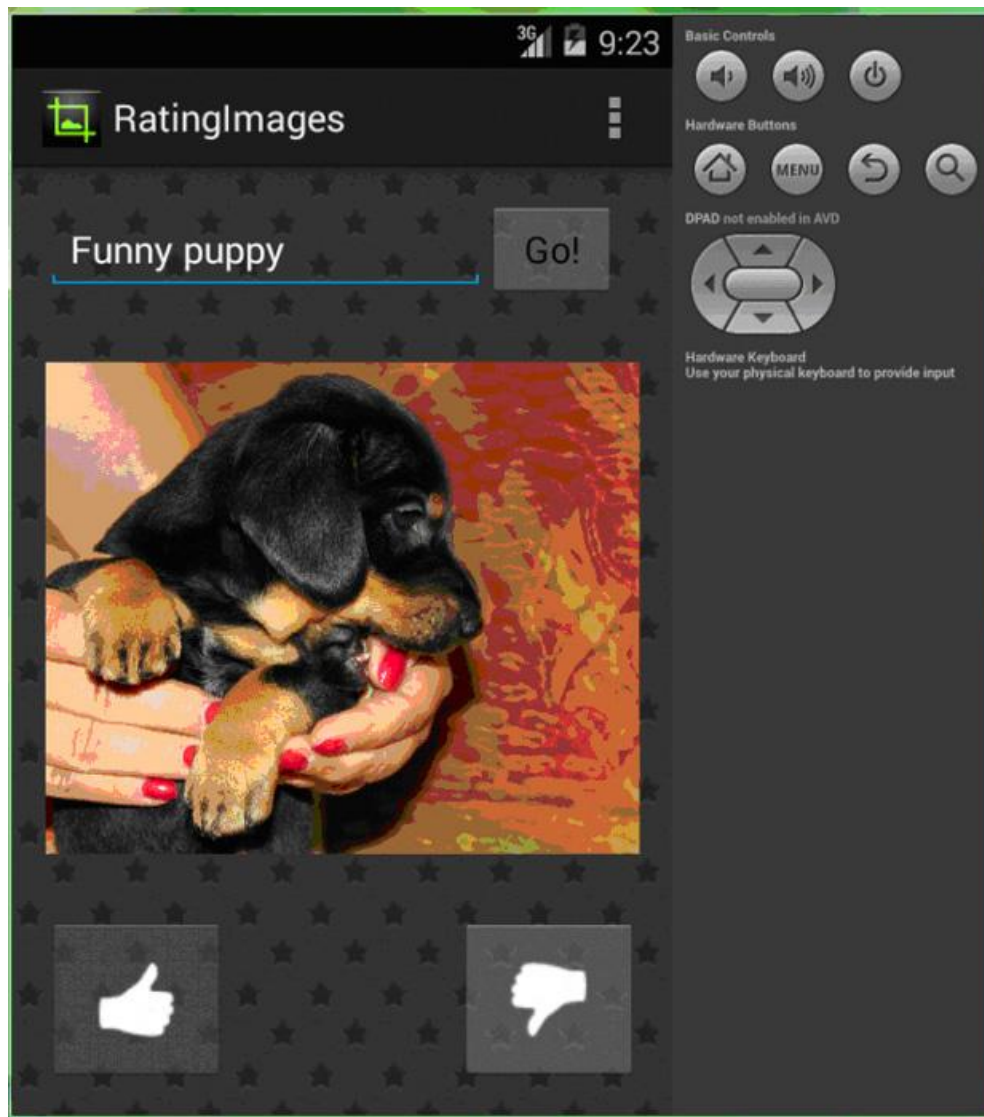


Рис. 6.48. Главная активность

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. 6.49а. Кнопка-изображение

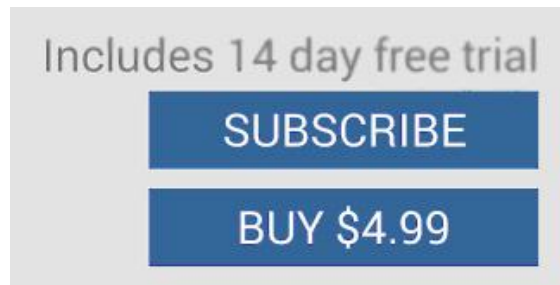


Рис. 6.49б. Кнопка-текст

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.

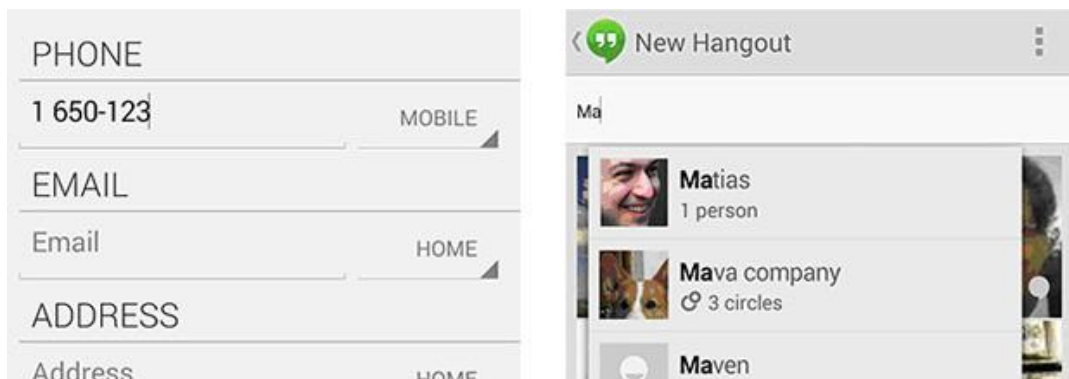


Рис. 6.50. 50 Текстовые поля, поиск контактов

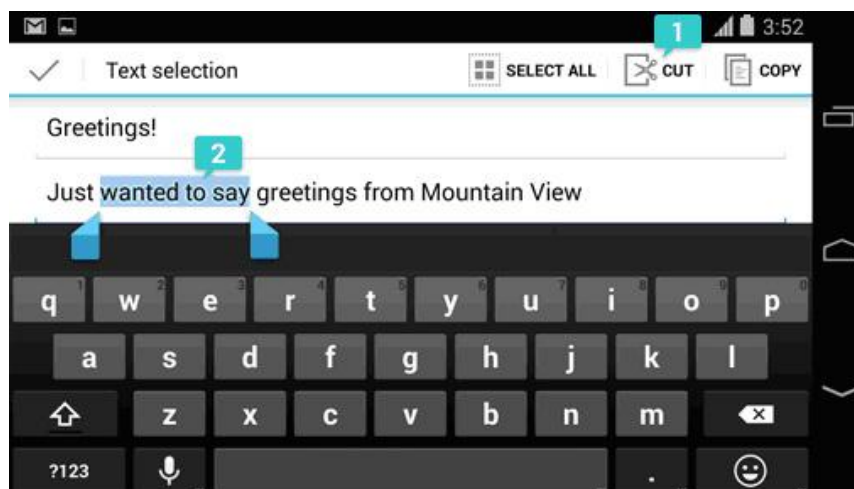


Рис. 6.51. Выделение текста

Ползунки и слайдеры (Seek Bars and Sliders)

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

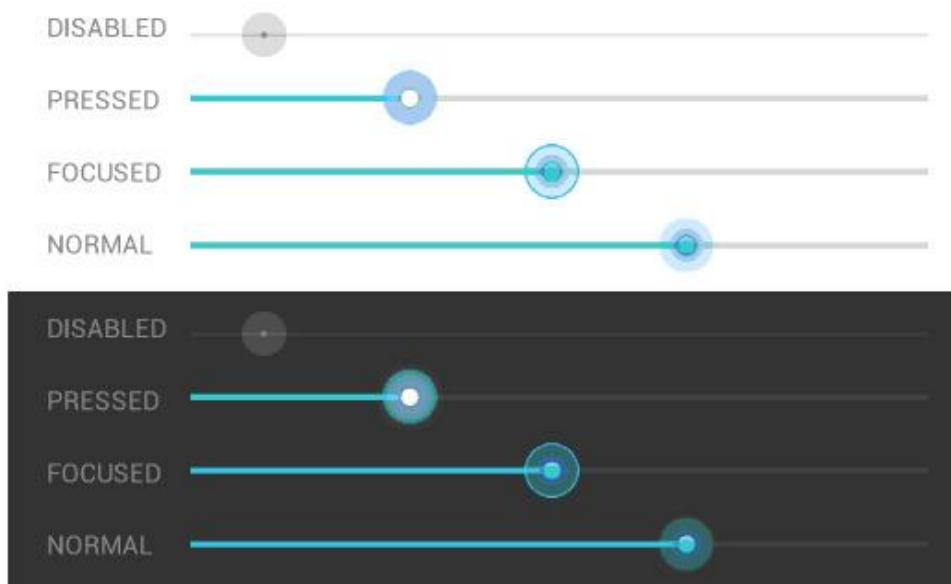


Рис. 6.52. Ползунки в светлой и темной темах

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.

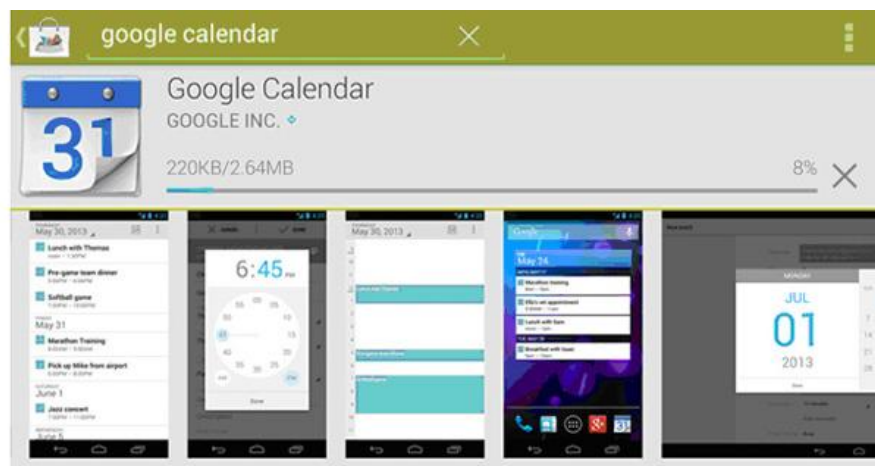


Рис. 6.54. Процесс загрузки приложения

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.

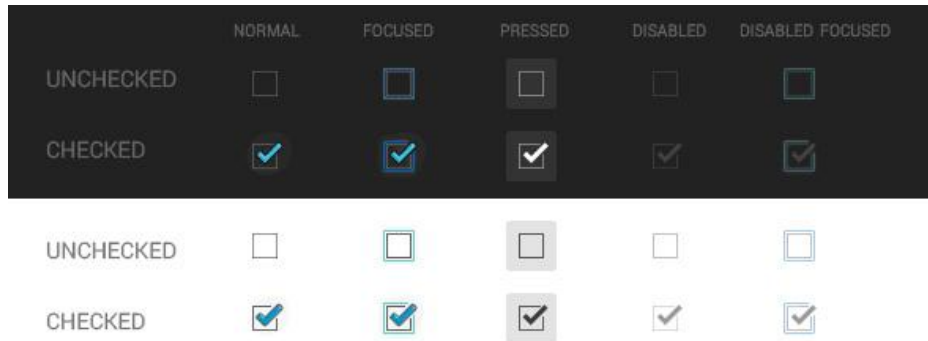


Рис. 6.55. Флажки

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

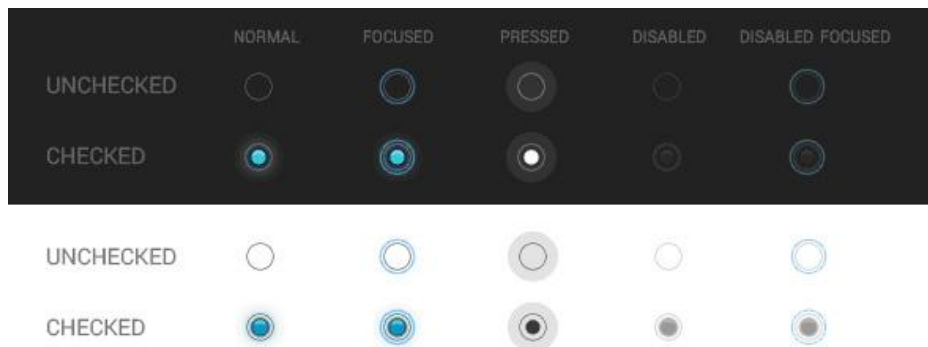


Рис. 6.56. Радио-кнопки

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 (ОК и Отмена), так и сложные окна с множеством настраиваемых пользователем параметров.

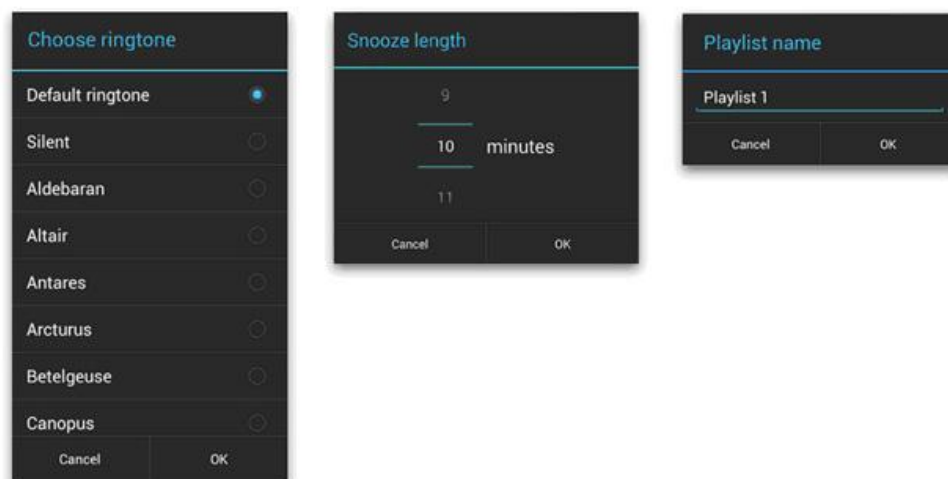


Рис. 6.58. Примеры диалоговых окон

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.

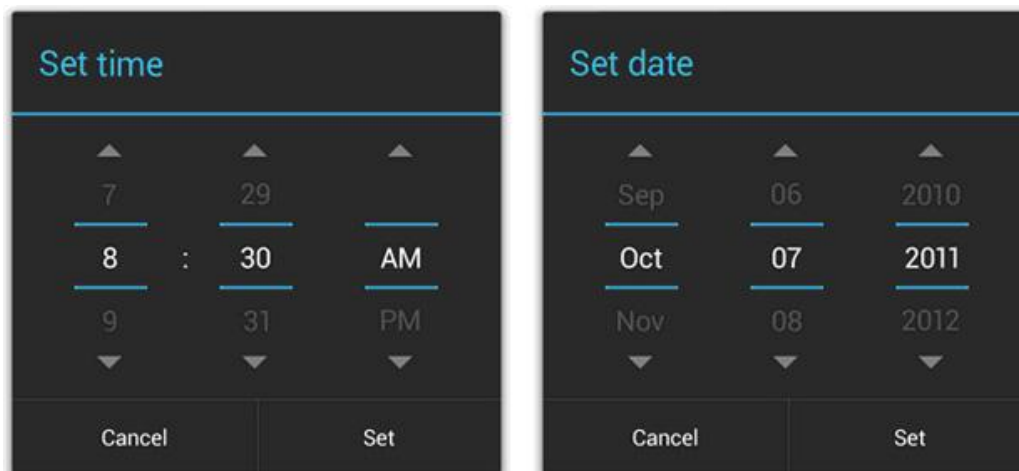


Рис. 6.59. Примеры пикеров

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.

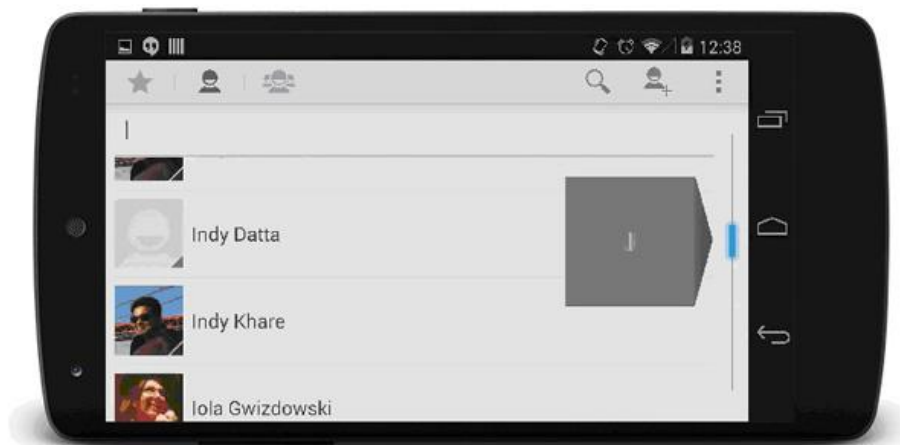


Рис. 6.60. Индикатор и индекс прокрутки

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

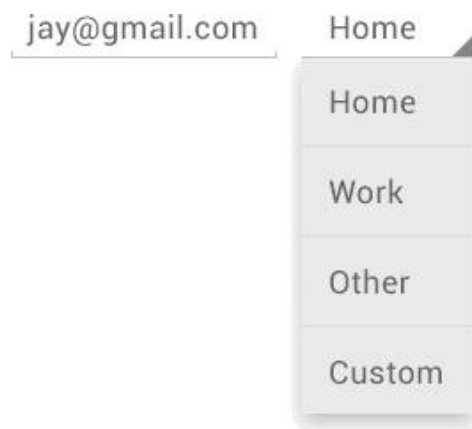


Рис. 6.61. Спиннер для выбора типа e-mail адреса

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

проще и приятнее.

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



Рис. 6.62. Вкладки

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

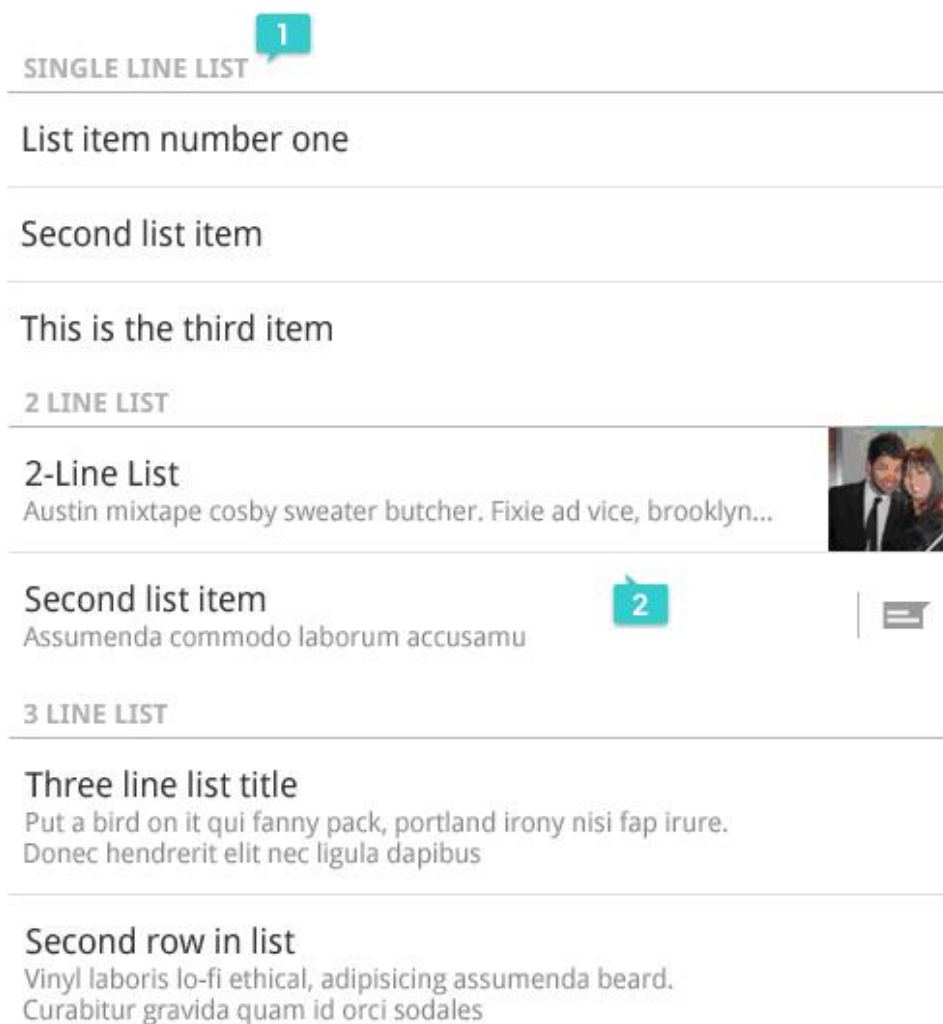


Рис. 6.63. Список

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.

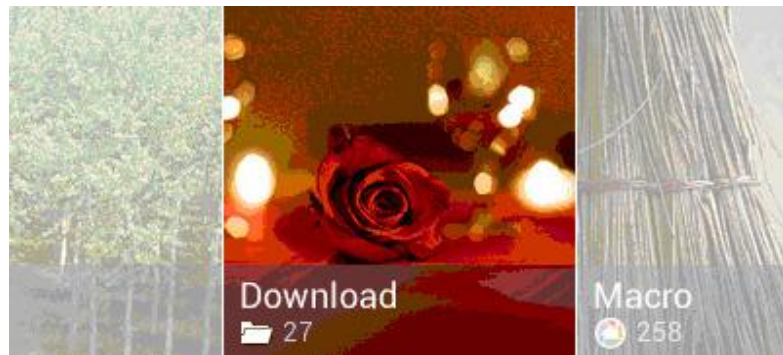


Рис. 6.64. Сетка с альбомами

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.

Лабораторная работа №5 Изменение элементов дизайна. Обработка событий: подсказки. Обработка событий: цветовая индикация

Цель: формирование у студентов знаний и навыков по работе с элементами экрана, обработке нажатий кнопок.

План занятия

1. Изучить теоретические сведения.
2. Выполнить практическое задание по лабораторной работе.
3. Оформить отчёт и ответить на контрольные вопросы.

Теоретические сведения

Мобильное приложение и мобильная платформа.

Мобильное приложение представляет собой специально разработанное под конкретную мобильную платформу (Android, iOS, Windows Phone) программное обеспечение. Такое приложение также называют нативным мобильным приложением: его разрабатывают на языке высокого уровня и компилируют в нативный код операционной системы, дающий максимальную производительность и функциональность. Нативные мобильные приложения распространяются через магазины (AppStore, Windows Store, Google Play), существенно расширяют способы монетизации бизнеса по сравнению с веб-приложениями, однако характеризуются высокой стоимостью и временем разработки.

Современные мобильные приложения создаются для линейки мобильных устройств: смартфонов, планшетов, электронных книг, цифровых проигрывателей, наручных часов, игровых приставок, нетбуков, смартбуков, очков, а также для телевизоров (тем самым выходя за рамки исключительно мобильных устройств). Работу таких устройств обеспечивает мобильная операционная система (мобильная платформа), сочетающая в себе функциональность операционной системы для персонального компьютера с функциями для мобильных и карманных устройств: сенсорный экран, сотовая связь, Bluetooth, Wi-Fi, GPS-навигация, камера, видеокамера, распознавание речи, диктофон, музыкальный плеер, NFC и инфракрасное дистанционное управление.

Наиболее распространённые операционные системы (платформы) для мобильных устройств: Android (платформа с открытым исходным кодом на основе ядра Linux и собственной реализации виртуальной машины Java от

Google), iOS (операционная система компании Apple, основанная на микроядре Mach и используемая в смартфонах iPhone), Windows Phone (разработка компании Microsoft).

Архитектура и основные компоненты мобильной платформы Android.

В рамках дисциплины «Эргономика мобильных приложений» рассматриваются вопросы проектирования интерфейсов и разработки мобильных приложений под платформу Android, получившую наибольшее распространение в мире.

Архитектуру Android принято делить на уровни ядра, библиотек и среды выполнения, каркаса приложений, собственно приложений (рисунок 1.1).

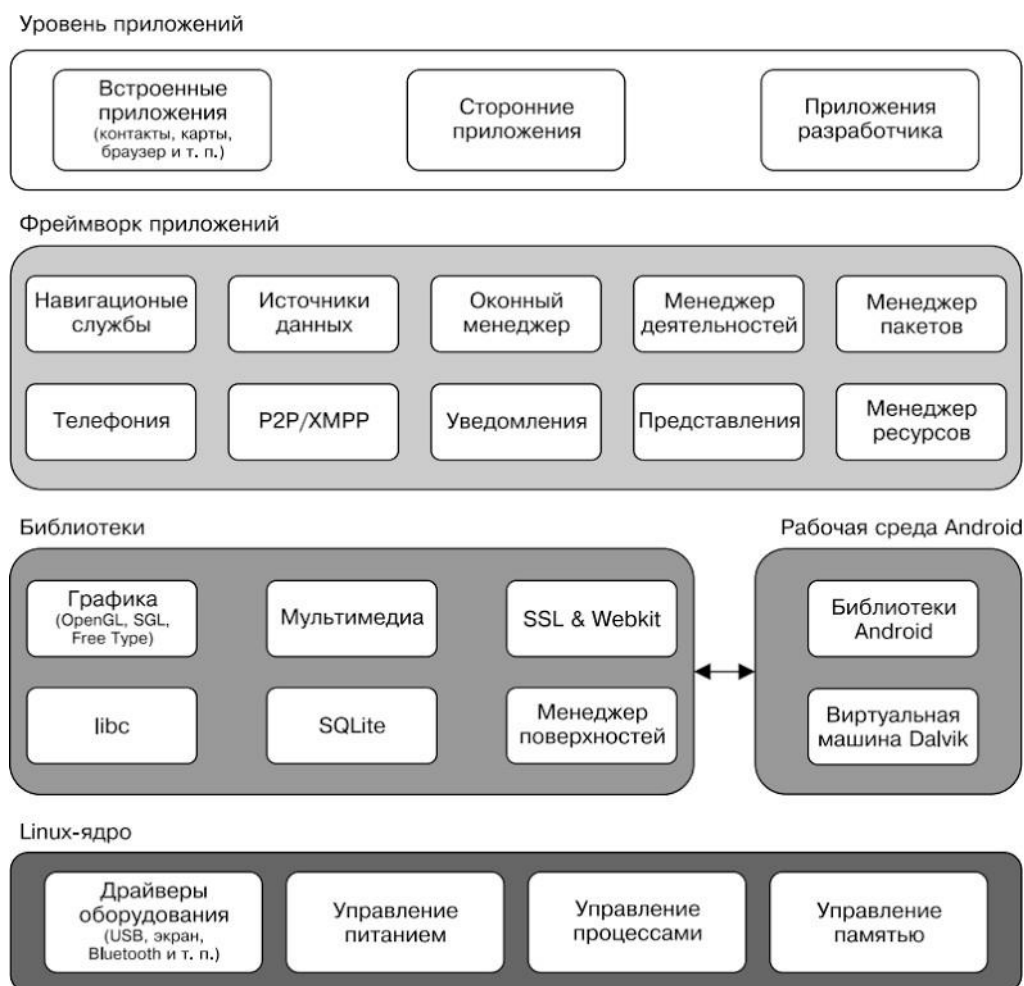


Рисунок 1.1 – Архитектура мобильной платформы Android

Ядро Linux обеспечивает функционирование системы и отвечает за безопасность, управление памятью, энергосистемой и процессами, а также предоставляет сетевой стек и модель драйверов.

Набор библиотек (Libraries) предназначен для обеспечения важнейшего базового функционала приложений и отвечает за поддержку файловых форматов, кодирование и декодирование информации (например, цифровой звук и видео), отображение графики, поддержку Web-компонентов (WebKit), SQL-СУБД (SQLite) и стандартной для Linux-систем функциональность библиотек

С. На этом же уровне располагается рабочая среда Android (Android Runtime). Каждое приложение в операционной системе Android запускается в собственном экземпляре виртуальной машины Dalvik. Таким образом, все работающие процессы изолированы от операционной системы и друг от друга. Благодаря этому осуществляется защита ядра операционной системы от возможного вреда со стороны других её составляющих.

Уровень каркаса приложений (Application Framework) включает основные службы Android для управления жизненным циклом приложений, пакетами, ресурсами и т. д. Программист имеет полный доступ к тем же API (Application Programming Interface), которые используются основными приложениями. Архитектура этих приложений разработана с целью упрощения многократного использования компонентов. Любое разрабатываемое приложение может использовать возможности базовых приложений и, соответственно, любое другое стороннее приложение может использовать возможности вашего приложения (с учетом установленных разрешений).

Уровень приложений (Applications) включает стандартные приложения Android (браузер WebKit, календарь Google, клиент Gmail, приложение Gmaps, SMS-мессенджер и e-mail клиент), а также дополнительно загруженные приложения (из магазина Android). Android не делает разницы между основными приложениями и сторонним программным обеспечением, в связи с чем ключевые компоненты, такие как набор номера, рабочий стол или почтовый клиент GMail, можно заменить альтернативными аналогами.

Всего в Android приложениях существует четыре типа компонентов из уровня каркаса приложений: деятельность (Activity), служба (Service), приемник широковещательных намерений (Broadcast Receiver), контент-провайдер (Content Provider). Взаимодействие компонентов осуществляется с помощью объектов Intent.

Activity представляет собой визуальный пользовательский интерфейс для приложения – окно. Activity может также использовать дополнительные окна, например всплывающее диалоговое окно. Все деятельности реализуются как подкласс базового класса Activity.

Компонент Service не имеет визуального интерфейса пользователя и выполняется в фоновом режиме в течение неопределенного периода времени,

пока не завершит свою работу. Service, как правило, требуется для длительных операций или для обеспечения работы удаленных процессов, но в общем случае это просто режим, который функционирует, когда приложение не в фокусе. Примером такого процесса может стать прослушивание музыки в то время, когда пользователь делает что-то другое или получение данных по сети без блокирования текущей активности. Приложения могут подключаться к компоненту Service или запускать его, если он не запущен, а также останавливать уже запущенные компоненты.

Broadcast Receiver используется для отслеживания внешних событий и реакции на них. Приложение может иметь несколько компонентов Broadcast Receiver, чтобы ответить на любые объявления, которые оно считает важными. При этом каждый компонент BroadcastReceiver будет определять код, который выполнится после возникновения конкретного внешнего события. С помощью класса NotificationManager можно сообщить пользователю информацию, требующую его внимания. Примером оповещений может быть сигнал о том, что информация загружена на устройство и доступна к использованию.

Content Provider делает определенный набор данных, используемых приложением, доступным для других приложений. Данные могут быть сохранены в файловой системе, в базе данных SQLite, в сети, или в любом другом месте, к которому приложение может иметь доступ. Контент-провайдеры для безопасного доступа к данным используют механизм разрешений. Посредством content provider другое приложение может запрашивать данные и, если выставлены соответствующие разрешения, изменять их. Например, система Android содержит content provider, который управляет пользовательской информацией о контактах.

Intent – специальные классы в коде приложения, которые определяют и описывают запросы приложения на выполнение каких-либо операций. Намерения добавляют слой, позволяющий оперировать компонентами с целью их повторного использования и замещения. В некоторых случаях это может быть очень мощным средством интеграции приложений. Главная особенность платформы Android состоит в том, что одно приложение может использовать элементы других приложений при условии, что эти приложения разрешают использовать свои компоненты. При этом ваше приложение не включает код другого приложения или ссылки на него, а просто запускает нужный элемент другого приложения.

Создание проекта в Android Studio.

Разработку приложений для платформы Android выполняют на языке Java. С 2013 года разработка выполняется в официальной среде компании Google – Android Studio, основанной на IntelliJ IDEA от JetBrains.

Для создания нового проекта в Android Studio выбираем в окне приветствия опцию New Project (для уже открытого проекта в меню File следует выбрать New Project). В появившемся окне Create New Project (рисунок 1.2) заполняем поля имени приложения (Application name), которое будет отображаться для пользователей, и квалификатора (Company Domain), который будет добавляться к имени пакета. Полное название проекта (Package name) формируется в соответствии с правилами именования пакетов в Java. При необходимости изменяем папку размещения проекта (Project location).

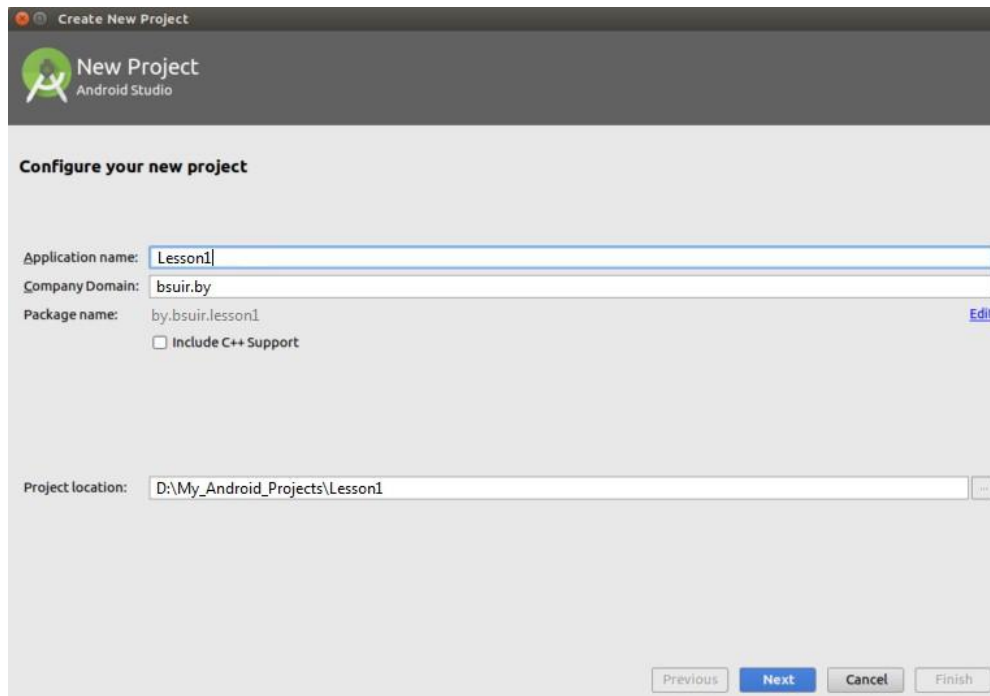


Рисунок 1.2 – Конфигурирование нового проекта в Android Studio

Далее (рисунок 1.3) указываем минимальную версию SDK платформы – самую раннюю версию Android, которую будет поддерживать приложение. Обратите внимание, что Minimum SDK не может быть выше версии API той мобильной платформы, на которой планируется тестирование и последующее использование приложения.

В появившемся окне Add an activity to Mobile в качестве основы будущего приложения выбираем Empty Activity, в результате чего будет создано приложение с одним Activity.

В форме Customize the Activity сгенерированы имена файлов класса Activity (Activity Name), графической разметки (Layout Name). Рекомендуется

оставить данные имена без изменений, т.к. для запускающего Activity они являются общепринятыми.

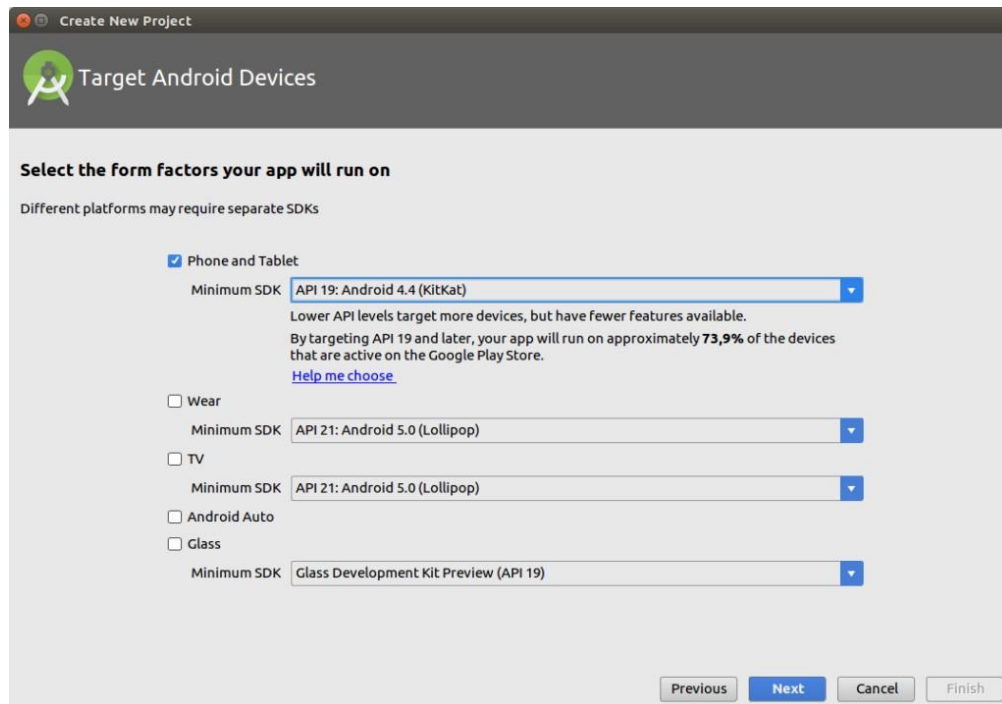


Рисунок 1.3 – Указание минимальной версии SDK для нового проекта в Android Studio

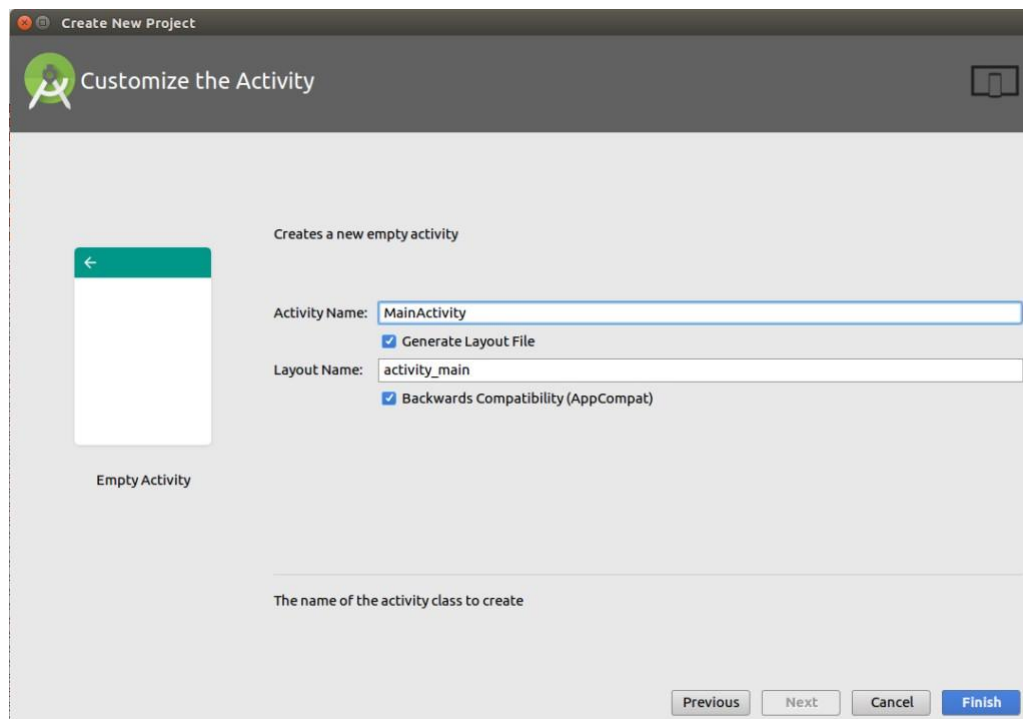


Рисунок 1.4 – Кастомизация имен для нового проекта в Android Studio

После этого Android Studio создает новый проект с указанными характеристиками. Структура проекта (рисунок 1.5) включает: файл `AndroidManifest.xml` (манифест приложения), папку `java` (содержит весь код приложения), папку `res` (используется для файлов-ресурсов различного типа: `res/drawable/` - для изображений (PNG, JPEG и т. д.); `res/layout/` - для xml- файлов графической разметки; `res/menu/` - для xml-файлов меню; `res/values/` - для строковых ресурсов, стилей и т. д.).

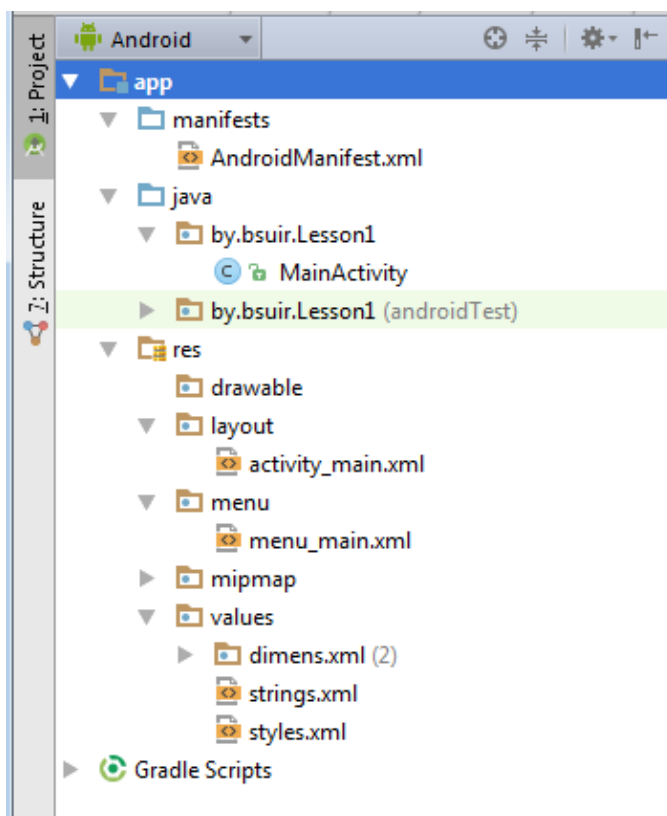


Рисунок 1.5 – Структура проекта в Android Studio Графическое

представление Activity.

Основу Android-приложения составляет Activity – рабочее окно. В конкретный момент времени обычно отображается одно Activity и занимает весь экран. Работа с набором окон осуществляется путем переключения между различными Activity. В качестве примера можно рассмотреть почтовое приложение: в нем одно Activity – список писем, другое – просмотр письма, третье – настройки ящика.

С программной точки зрения Activity представлено файлом графической разметки (`activity_main.xml` на рисунке 1.5) и соответствующим java-классом

(MainActivity на рисунке 1.5), реализующем запуск данного Activity с требуемым графическим представлением и последующей обработкой событий.

Графическое представление Activity формируется из различных компонентов (кнопка, поле ввода, чекбокс и т.д.), называемых View (рисунок 1.6).

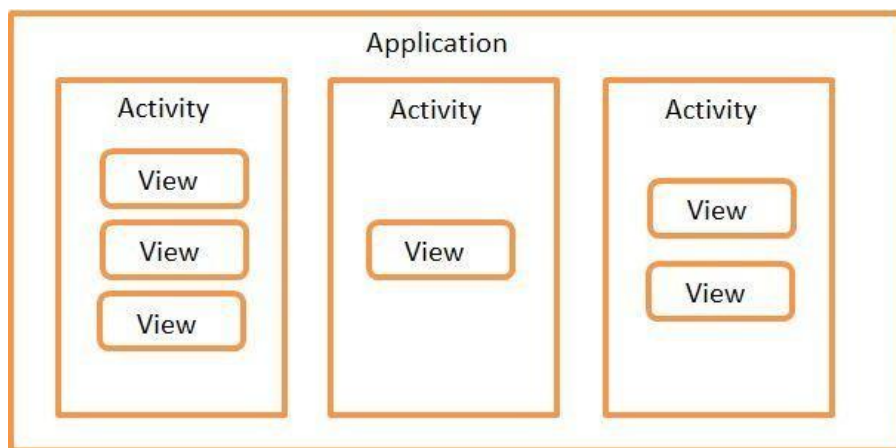


Рисунок 1.6 – Представление приложения Android как набора окон (Activity) с View-компонентами

Графическое представление каждого Activity в виде требуемого набора и взаимного расположения View-элементов хранится в xml-файле папки layout. Данный файл графического представления прописывается в соответствующем java-классе. При запуске приложения Activity читает этот файл и отображает его содержимое.

После создания проекта файл activity_main.xml открывается по умолчанию (рисунок 1.7) в режиме конструктора (вкладка Design), помимо которого также существует соответствующее xml-описание графического представления (вкладка Text).

На вкладке Design слева отображен список всех возможных виджетов (View-элементов), разделенный на группы. Справа на вкладке Component Tree отображаются все выбранные элементы, которые описаны в этом layout- файле. Вкладка Properties отображает набор свойств для каждого выбранного элемента.

View-компоненты обычно группируют в ViewGroup, называемые Layout. Layout бывают различных типов и отвечают за то, как будут расположены их дочерние View-компоненты на экране (таблицей, строкой, столбцом). Рассмотрим основные виды Layout.

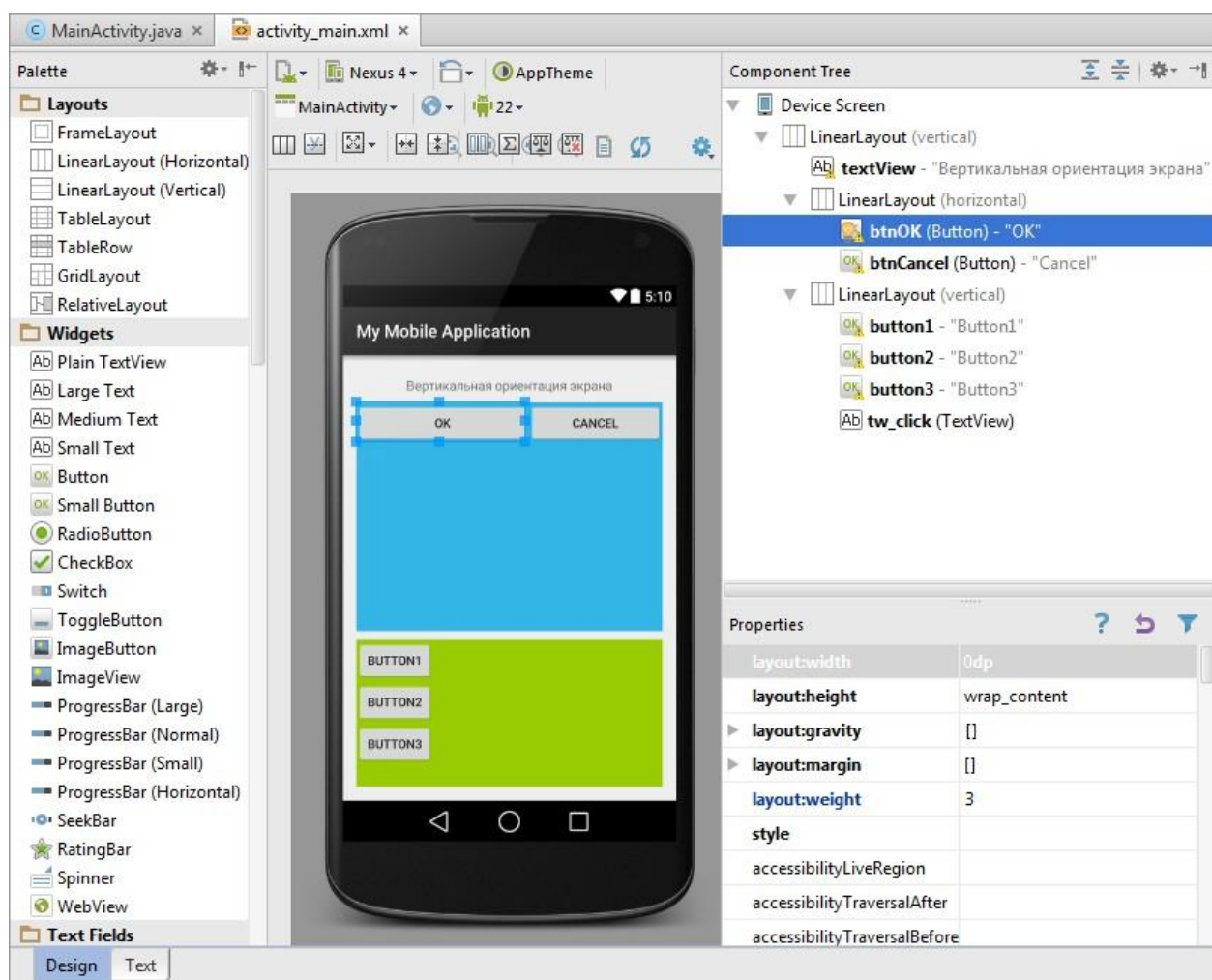


Рисунок 1.7 – Графическое представление Activity, вкладка Design

LinearLayout (LL) отображает View-элементы в виде одной строки (Horizontal) или одного столбца (Vertical). Он удобен и достаточно гибок, что- бы создавать экраны различной сложности. LL имеет свойство Orientation, ко- торое определяет, как будут расположены дочерние элементы – горизонталь- ной или вертикальной линией.

TableLayout (TL) отображает элементы в виде таблицы. TL состоит из строк TableRow (TR), каждая TR в свою очередь содержит View-элементы, формирующие столбцы. Т.е. кол-во View в TR - это кол-во столбцов. Но кол- во столбцов в таблице должно быть равным для всех строк. Поэтому, если в разных TR разное кол-во View-элементов (столбцов), то общее кол-во опреде- ляется по TR с максимальным кол- вом.

RelativeLayout (RL) предусматривает настройку для каждого элемента его положения относительно других элементов. Виды отношений:

1) слева, справа, сверху, снизу указанного элемента (`layout_toLeftOf`, `layout_toRightOf`, `layout_above`, `layout_below`).

2) выравненным по левому, правому, верхнему, нижнему краю указанного элемента (`layout_alignLeft`, `layout_alignRight`, `layout_alignTop`, `layout_alignBottom`).

3) выравненным по левому, правому, верхнему, нижнему краю родителя (`layout_alignParentLeft`, `layout_alignParentRight`, `layout_alignParentTop`, `layout_alignParentBottom`).

4) выравненным по центру вертикально, по центру горизонтально, по центру вертикально и горизонтально относительно родителя (`layout_centerVertical`, `layout_centerHorizontal`, `layout_centerInParent`).

`AbsoluteLayout` (AL) обеспечивает абсолютное позиционирование элементов на экране путем указания координат для левого верхнего угла компонента. Использование AL оправдано в случае разработки для экрана с конкретным разрешением. Если открыть такое приложение на другом экране, все элементы сместятся. Поэтому в общем случае AL не рекомендуется использовать. Его совместимость с будущими версиями Android не гарантируется.

Путем вложения в один `Layout` других `Layout` можно создавать экраны различной сложности. В рамках одного `Layout` располагают требуемые `View`-элементы и настраивают их параметры.

Рассмотрим параметры `View`-элементов.

Параметр `id` – это ID элемента. Чтобы обратиться к элементу экрана из кода, потребуется его ID. Для ID существует четкий формат: `@+id/name`, где `+` означает, что это новый ресурс и он должен добавиться в `R.java` класс, если он там еще не существует.

Параметры `layout_width` (ширина элемента) и `layout_height` (высота элемента) могут быть следующими: `fill_parent` или `match_parent` (максимально возможная ширина или высота в пределах родителя), `wrap_content` (ширина или высота определяется по содержимому элемента), а могут задаваться в значениях:

`dp` (Density-independent Pixels) – абстрактная единица измерения, позволяющая приложениям выглядеть одинаково на различных экранах и разрешениях;

`sp` (Scale-independent Pixels) – то же, что и `dp`, только используется для размеров шрифта в `View` элементах;

`pt` – 1/72 дюйма, определяется по физическому размеру экрана;

px – пиксел, не рекомендуется использовать т.к. на разных экранах приложение будет выглядеть по-разному;

mm – миллиметр, определяется по физическому размеру экрана; in – дюйм, определяется по физическому размеру экрана.

Если родитель содержит несколько элементов и необходимо, чтобы они заняли все пространство, целесообразно использовать параметр Layout weight – вес. Свободное пространство распределяется между элементами пропорционально их weight-значениям. При использовании weight вы можете указать значение height или width = 0dp. В этом случае не будет учитываться содержание элементов и результат будет более соответствующий коэффициентам веса.

Параметр Layout gravity – это выравнивание. Параметр Layout margin – это отступ.

При переходе на вкладку Text отображается xml-описание всех элементов layout-файла (рисунок 1.8).

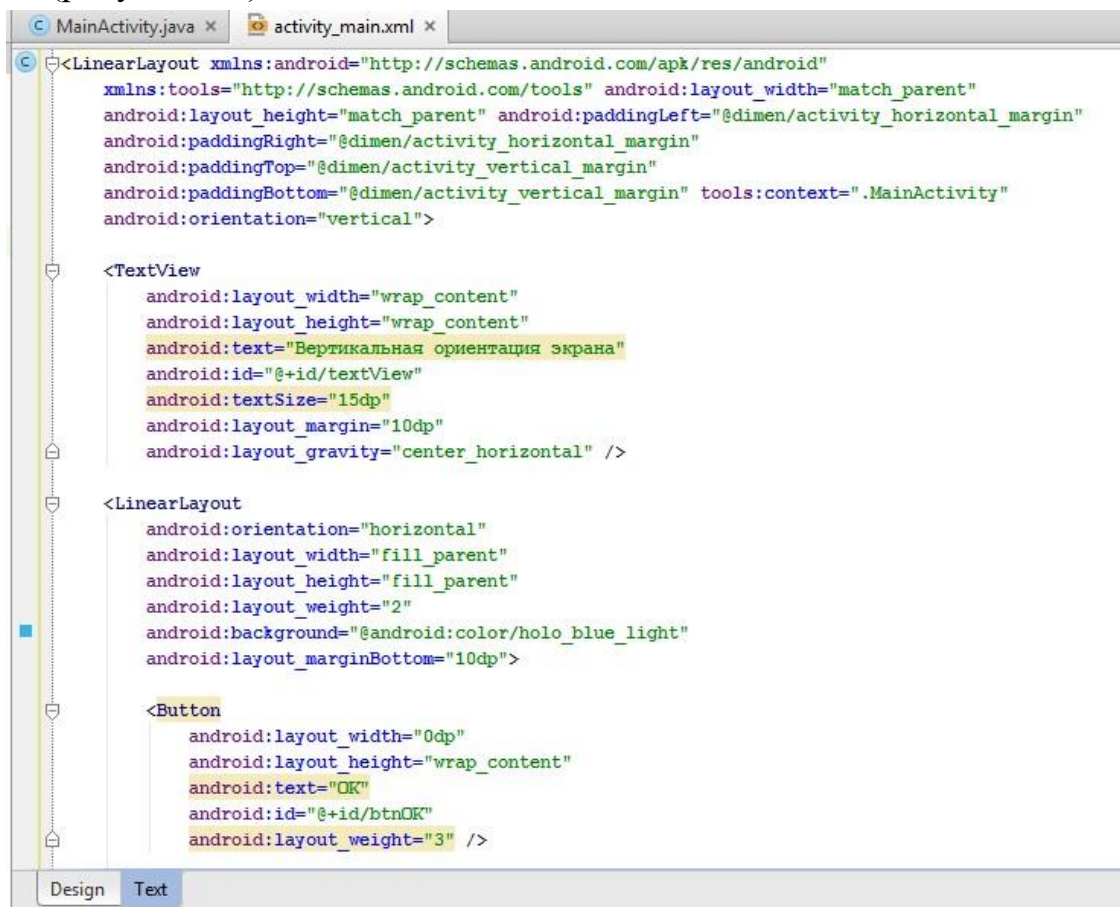
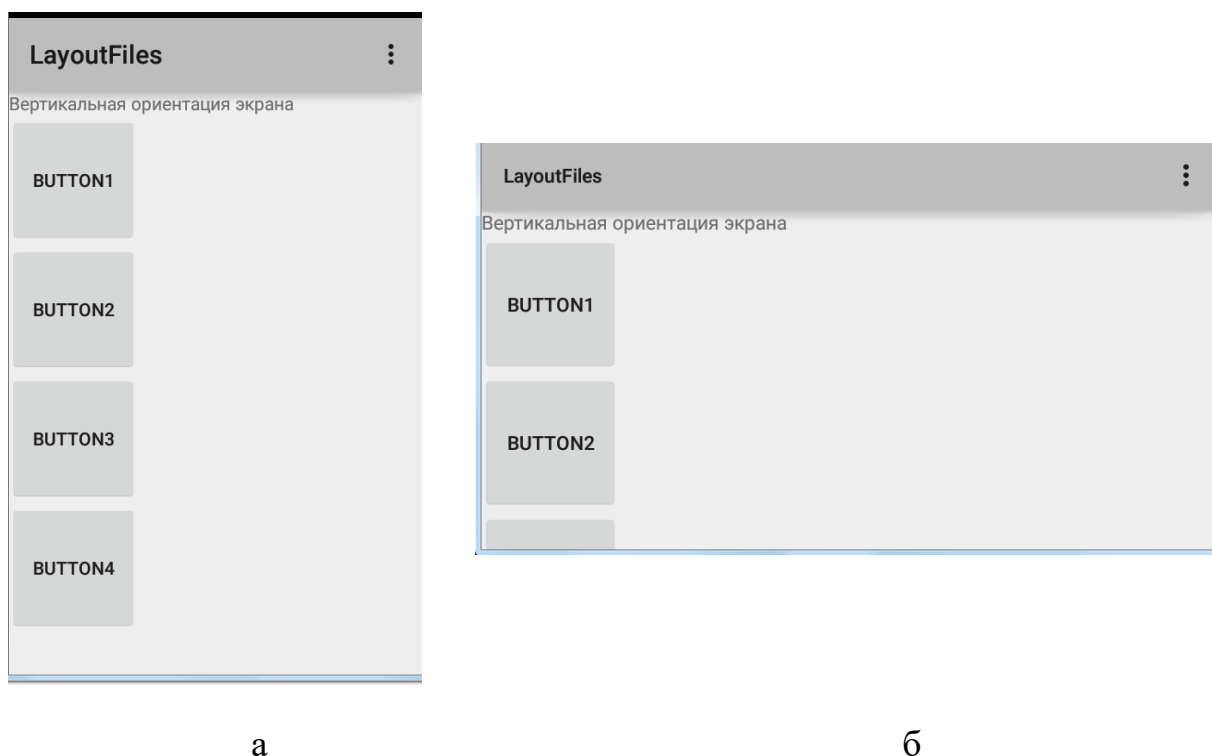


Рисунок 1.8 – Графическое представление Activity, вкладка Text

Названия xml-элементов - это классы View-элементов, xml-атрибуты - это параметры View-элементов, т.е. все те параметры, которые можно менять через вкладку Properties. Также можно вносить изменения прямо в xml-код, соответствующие изменения будут отображаться и во вкладке Design.

Layout-файл при смене ориентации экрана.

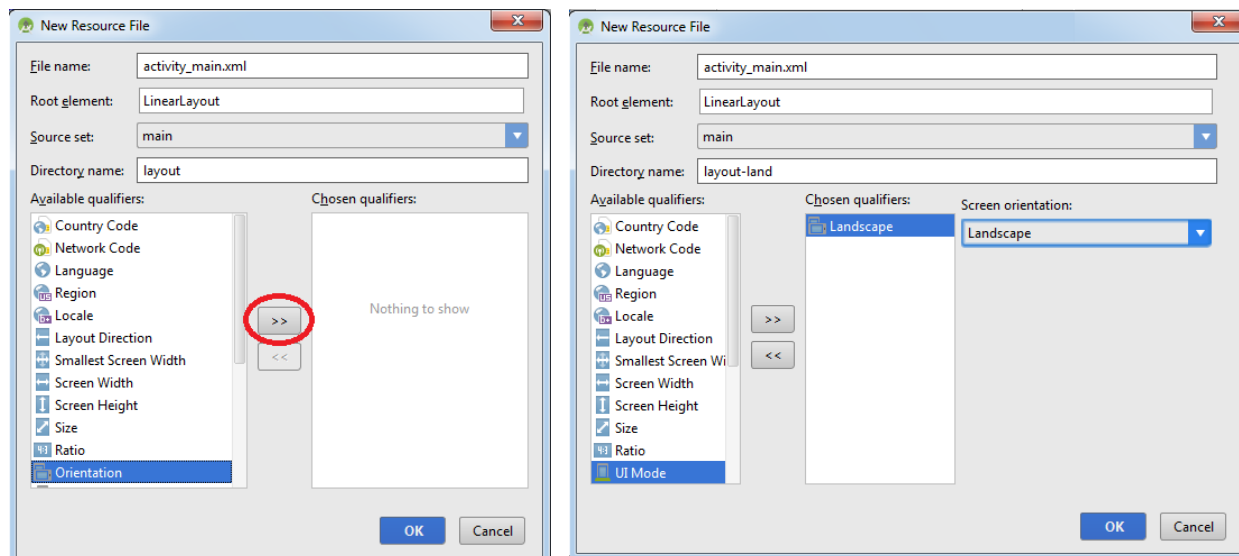
По умолчанию мы настраиваем layout-файл под вертикальную ориентацию экрана. Однако при повороте смартфона включится горизонтальная ориентация, что может привести к некорректному отображению View-элементов (рисунок 1.9).



а б
Рисунок 1.9 – Пример некорректного отображения View-элементов при смене ориентации экрана с вертикальной (а) на горизонтальную (б)

Для устранения данной проблемы необходимо создать еще один layout-файл для горизонтальной ориентации экрана.

В папке res/layout создаем новый Layout resource file. Название файла указываем то же, что и для xml-представления в вертикальной ориентации. В списке квалификаторов слева находим Orientation (рисунок 1.10, а) и нажимаем кнопку со стрелкой вправо. Далее выбираем параметр Landscape из выпадающего списка (рисунок 1.10, б). Обратите внимание на автоматическое изменение значения поля Directory name.



а

б

Рисунок 1.10 – Создание графического представления для горизонтальной ориентации экрана: выбор квалификатора Orientation (а) и установка параметра Landscape (б)

В структуре проекта появится новый xml-файл `activity_main.xml(land)`. Копируем в него содержимое файла `activity_main.xml` и меняем параметры `layout` и `view`-элементов таким образом, чтобы в горизонтальной ориентации экрана они отображались корректно. При этом `id` элементов не меняем!

При запуске приложения Activity прочитает подключенный в коде `layout`-файл и отобразит его содержимое. При этом будет учтена ориентация экрана, и в случае горизонтального расположения автоматически отобразится файл `land`.

Файл MainActivity.java. Подключение графического представления к Activity.

При запуске деятельности система должна получить ссылку на корневой узел дерева разметки, который будет использоваться для прорисовки графического интерфейса на экране мобильного устройства. Для этого в методе `onCreate()` необходимо вызвать метод `setContentView()`.

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

Метод `setContentView(int)` – устанавливает содержимое Activity из layout-файла. Но в качестве аргумента указывается не путь к layout-файлу (`res/layout/activity_main.xml`), а константа, которая является ID файла и хранится в файле `R.java`. Имена этих ID-констант совпадают с именами файлов ресурсов (без расширений).

Можно создать новый xml-файл в папке `res > layout` и прописать его вместо `activity_main` в методе `setContentView(int)`. После запуска приложения отобразится новый файл разметки.

Доступ к элементам экрана из кода.

Чтобы обратиться к элементу экрана из кода, необходим его ID. Он прописывается в `Properties` либо в layout-файлах:

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

Важно давать уникальные и осмысленные ID-имена элементам экрана! Зная ID View-элемента, обратиться к нему из кода можно по константе `R.id.btnOK`. Для этого понадобится метод `findViewById`:

```
public class MainActivity extends ActionBarActivity{ private Button btnOK;

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

    private void initView() {
        // находим View-элементы
        btnOK = (Button) findViewById(R.id.btnOK);
    }
}
```

Следует отметить, что в вышеприведенном фрагменте кода обнаружение View-элементов вынесено в отдельный пользовательский (не являющийся библиотечным) метод `initViews()` для реализации принципа модульного программирования.

Обработка событий на примере нажатия кнопки.

Механизм обработки нажатия на кнопку основан на использовании интерфейса `View.OnClickListener` и реализации метода `onClick`, в котором и прописывается логика действий в ответ на нажатие (рисунок 1.11).

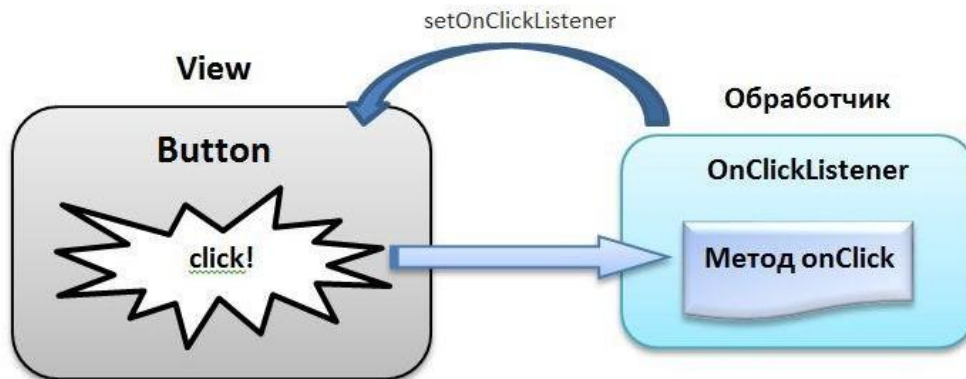


Рисунок 1.11 – Механизм обработки нажатия на кнопку на основе интерфейса `View.OnClickListener`

При этом обработчик (его также называют слушателем - `listener`) присваивается кнопке с помощью метода `setOnClickListener`. Когда на кнопку нажимают, обработчик реагирует и выполняет код из метода `onClick`.

Для реализации вышеописанного механизма необходимо выполнить следующие шаги:

- создать обработчик (объект от интерфейса `View.OnClickListener`),
- заполнить метод `onClick` (т.к. в интерфейсе он не реализован, а только заявлен),
- присвоить обработчик кнопке (используем метод `setOnClickListener`).

Система обработки событий готова: когда на кнопку нажимают, обработчик реагирует и выполняет код из метода `onClick`.

Пример:

```
OnClickListener listener = new OnClickListener() { @Override
    public void onClick(View v) {
        // метод, который будет вызван по нажатию
```

```
    }  
};
```

```
button.setOnClickListener(listener);
```

Существуют различные способы программной реализации обработки нажатий от требуемого набора кнопок одного Activity (с использованием в xml-представлении атрибута `onClick`, своего обработчика для каждого View-элемента, одного обработчика для нескольких View-элементов). Однако наиболее оптимальным является использование Activity в качестве единого обработчика. В данном случае сам класс Activity реализует интерфейс `View.OnClickListener`:

```
public class MainActivity extends ActionBarActivity implements  
View.OnClickListener{  
  
    private Button btnOK, btnCancel; @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main); initViews();  
    }  
  
    private void initViews() {  
        // находим View-элементы  
        btnOK = (Button) findViewById(R.id.btnOK); btnCancel = (Button)  
        findViewById(R.id.btnCancel);  
        // подключаем обработчик к кнопкам btnOK.setOnClickListener(this);  
        btnCancel.setOnClickListener(this);  
    }  
  
    @Override  
    public void onClick(View v) { switch (v.getId()) {
```

```

        case R.id.btnOK: btnOK.setText("Hello"); btnOK.setEnabled(false);
            btnCancel.setEnabled(true); break;
        case R.id.btnCancel: btnCancel.setText("Goodbye");
            btnCancel.setEnabled(false); btnOK.setEnabled(true);
            break;
    }
}
}

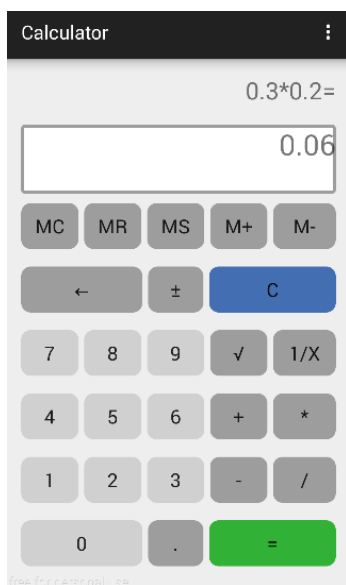
```

Для считывания данных из текстовых полей (например, из поля `et_message`) используется следующий код:

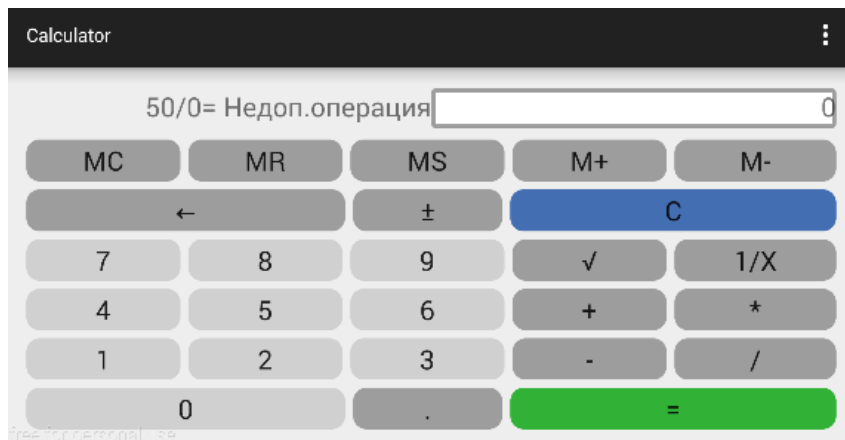
```
String message = et_message.getText().toString();
```

Практическое задание

1. Разработать приложение Calculator с одним Activity.
2. Графическое представление приложения Calculator реализовать с кнопками цифр, математических операций (сложения, вычитания, умножения, деления и др.), получения результата (пример приведен на рисунке 1.12, а).
3. Предусмотреть возможность ввода дробных чисел через точку (например, 0.5).
4. Предусмотреть возможность ввода отрицательных чисел.
5. Создать графическое представление приложения Calculator для горизонтальной ориентации экрана (пример приведен на рисунке 1.12, б).
6. Программно реализовать обработку нажатий на кнопки с использованием Activity в качестве единого обработчика.
7. В случае деления на 0 выводить вместо результата сообщение о недопустимости операции.
8. Разработать перечень проверок и протестировать приложение Calculator.
9. Продемонстрировать работу приложения Calculator на эмуляторе или реальном устройстве.



а



б

Рисунок 1.12 – Пример работы приложения Калькулятор в вертикальной (а) и горизонтальной (б) ориентации экрана

Содержание отчета

1. Скриншоты графических представлений приложения Calculator в вертикальной и горизонтальной ориентациях экрана в Android Studio.
2. Код xml-файла графического представления приложения Calculator в вертикальной ориентации экрана.
3. Код xml-файла графического представления приложения Calculator в горизонтальной ориентации экрана.
4. Код java-файла Activity приложения Calculator.

Контрольные вопросы

1. Что такое нативное мобильное приложение, мобильная платформа?
2. Что собой представляет архитектура мобильной платформы Android?
3. Какие основные компоненты Android-приложения Вы знаете?
4. Что собой представляет структура Android-проекта? Что содержит файл конфигурации AndroidManifest.xml, папка java, папка res?
5. Что такое графическое представление Activity?
6. Что такое Layout? Какие существуют виды Layout?
7. Какие параметры (атрибуты) имеют View-элементы?
8. Как создать Layout-файл для работы в горизонтальной ориентации экрана мобильного устройства? В каких случаях это необходимо?
9. Для чего нужны методы setContentView, findViewById?
10. Какие существуют способы обработки событий в Activity?

Лабораторная работа №6 Подготовка стандартных модулей. Обработка событий: переключение между экранами. Передача данных между модулями

Цель: формирование у студентов знаний о жизненном цикле Activity, навыков создания и вызова нового Activity, передачи данных между Activity, хранения данных с помощью класса SharedPreferences, создания всплывающих сообщений и логирования.

План занятия

1. Изучить теоретические сведения.
2. Выполнить практическое задание по лабораторной работе.
3. Оформить отчет и ответить на контрольные вопросы.

Теоретические сведения

Работа мобильного приложения может сопровождаться возникновением всплывающих сообщений, диалоговых окон, работой с меню и т.д. Рассмотрим реализацию простейшего вспомогательного элемента – всплывающего сообщения.

Всплывающие сообщения

Приложение может показывать всплывающие сообщения с помощью класса Toast:

```
Toast.makeText(context, text , duration).show();
```

Статический метод `makeText` создает View-элемент `Toast`. Далее рассмотрим параметры метода.

`context` – объект, который предоставляет доступ к базовым функциям приложения, т.к. `Activity` является подклассом `Context`, то в качестве объекта от `Context` используем текущее `Activity`, т.е. `this`.

`text` – текст, который надо вывести.

`duration` – продолжительность показа (`Toast.LENGTH_LONG` – длинная (3,5 сек), `Toast.LENGTH_SHORT` – короткая (2 сек)).

Чтобы `Toast` отобразился на экране, вызывается метод `show()`. Пример реализации (рисунок 2.1):

```
Toast.makeText(this, "Show Text Views!",
Toast.LENGTH_SHORT).show();
```

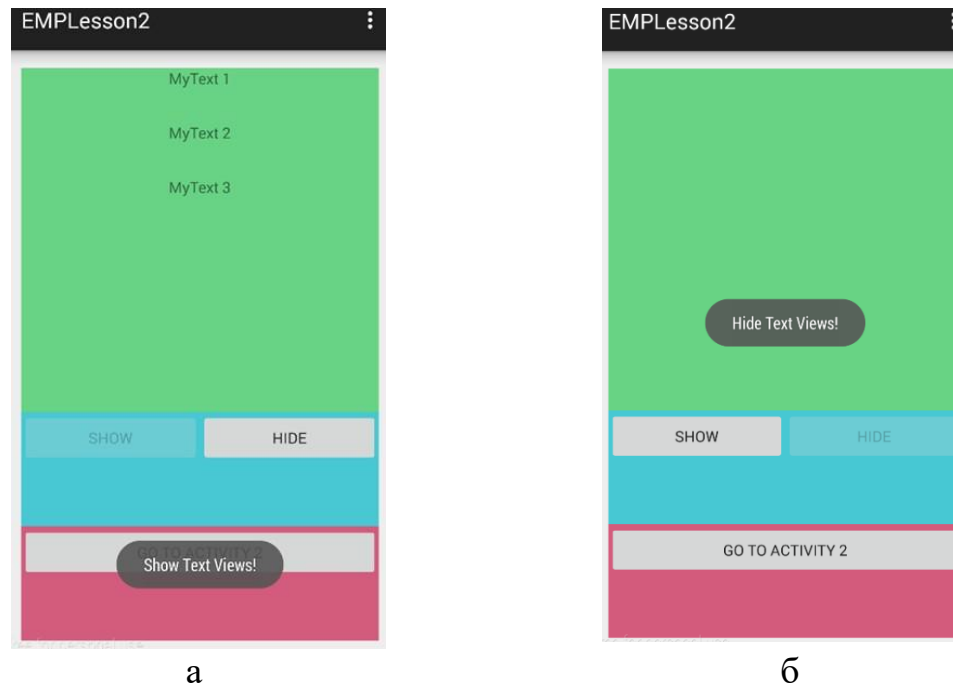


Рисунок 2.1 – Пример отображения всплывающих сообщений:
а – вывод сообщения "Show Text Views!" внизу окна после нажатия на кнопку Show и его размещаемого по умолчанию; б – вывод сообщения "Hide Text Views!" в центре окна после нажатия на кнопку Hide

По умолчанию всплывающее сообщение отображается в нижней части рабочего окна по центру. Для изменения места расположения всплывающего сообщения используется метод `setGravity`:

```
Toast toast = Toast.makeText(this, "Hide Text Views!",
Toast.LENGTH_SHORT);
toast.setGravity(Gravity.CENTER, 0, 0); toast.show();
```

Логирование.

При тестировании работы приложения можно отслеживать логи в окне logcat (рисунок 2.2). Логи имеют разные уровни важности: error, warn, info, debug, verbose (по убыванию). Установка Log level в окне logcat приводит к

фильтрации логов указанного уровня, а также уровней более высокой важности. Имеется возможность создавать, редактировать и удалять свои фильтры.

Логирование является одним из инструментов разработчика на этапе тестирования приложения. Для этого программно создаются логи, по их выводу в окне logcat отслеживается логика работы приложения, а перед выпуском финальной версии созданные логи из программы удаляются.

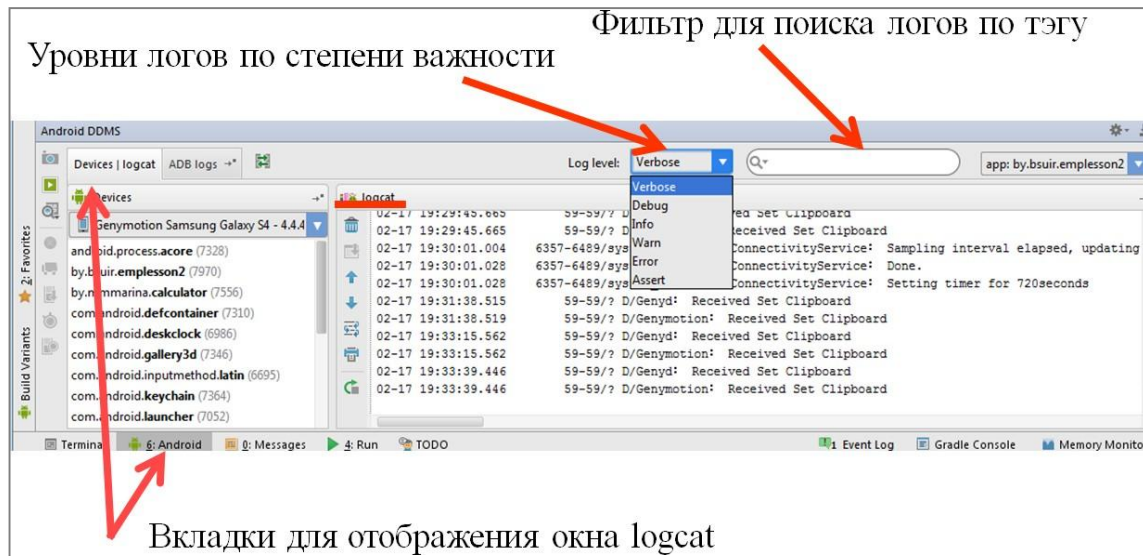


Рисунок 2.2 – Работа с окном logcat

Программно логирование выполняется с помощью класса Log и его методов Log.v() Log.d() Log.i() Log.w() and Log.e(). Названия методов соответствуют уровню логов. Пример синтаксиса методов:

```
Log.d(tag, text);
```

Методы требуют на вход тэг и текст сообщения. Тэг – это метка для реализации возможности поиска конкретного лога путем создания собственного фильтра.

Пример реализации логирования в коде: private static final String TAG =

```
"myLogs";
```

```
public void onClick(View v) { switch (v.getId()) {
```

```

        case R.id.btnShow:
            Log.d(TAG, "Нажата кнопка Show"); break;
        case R.id.btnHide:
            Log.d(TAG, "Нажата кнопка Hide"); break;
    }
}

```

Жизненный цикл Activity.

Созданное при работе приложения Activity может быть в одном из трех состояний:

1. Resumed: Activity видно на экране, оно находится в фокусе, пользователь может с ним взаимодействовать. Это состояние также иногда называют Running.

2. Paused: Activity не в фокусе, пользователь не может с ним взаимодействовать, но его видно (оно перекрыто другим Activity, которое занимает не весь экран или полупрозрачно).

3. Stopped: Activity не видно (полностью перекрывается другим Activity), соответственно оно не в фокусе и пользователь не может с ним взаимодействовать.

Когда Activity переходит из одного состояния в другое, система автоматически вызывает соответствующие методы, в которые можно добавлять свой код. Методы Activity, которые вызывает система (рисунок 2.3):

- onCreate() – вызывается при первом создании Activity.
- onStart() – вызывается перед тем, как Activity будет видно пользователю.
- onResume() – вызывается перед тем как будет доступно для активности пользователя (взаимодействие).
- onPause() – вызывается перед тем, как будет показано другое Activity.
- onStop() – вызывается когда Activity становится не видно пользователю.
- onDestroy() – вызывается перед тем, как Activity будет уничтожено.

Обратите внимание на то, что сами методы НЕ вызывают смену состояния. Наоборот, смена состояния Activity является триггером, который вызывает эти методы. Тем самым мы можем реагировать на произошедшее событие необходимым образом.

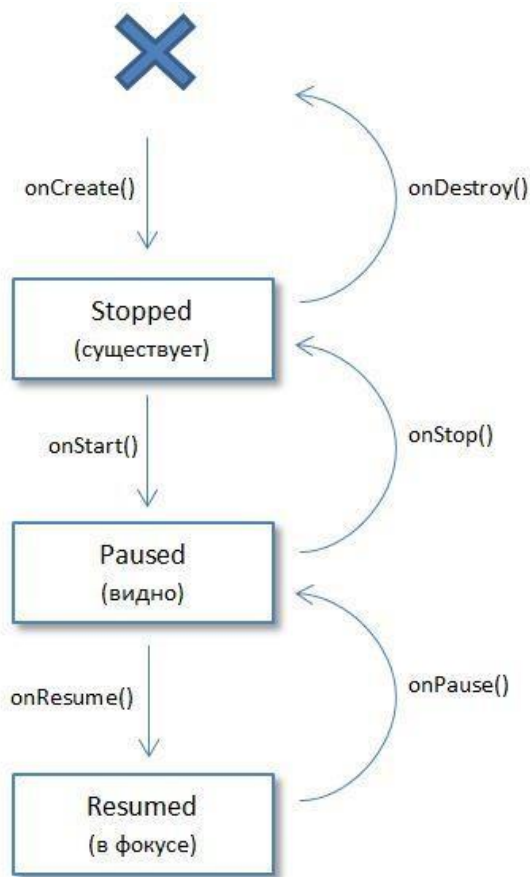


Рисунок 2.3 – Жизненный цикл Activity

Для того, чтобы в ответ на происходящие изменения состояния Activity отработал требуемый код, необходимо переопределить вышеописанные методы жизненного цикла Activity (вызов и вставка в код переопределяемого метода производится с помощью комбинации клавиш `Ctrl + O`). Далее приведен фрагмент кода, переопределяющего методы `onStart()` и `onResume()` путем добавления логирования.

```
@Override
protected void onStart() {
    super.onStart(); // метод родительского класса (его нельзя удалять!)
    Log.d(TAG, "MainActivity: onStart"); // добавляем логи
}
@Override
protected void onResume() { super.onResume();
```

```
Log.d(TAG, "MainActivity: onResume"); // добавляем логи
}
```

Аналогичным образом можно переопределить все 6 методов, включая onCreate(). Следует отметить, что добавочный код следует располагать только после вызова родительского метода (например, super.onStart() для onStart()).

Создание нового Activity.

Мобильное приложение, как правило, содержит несколько Activity и реализует переход между ними с передачей данных при необходимости.

Создать новое Activity можно вручную или воспользоваться встроенной функцией добавления нового компонента – Activity – в существующий проект. Далее рассмотрим оба способа.

Алгоритм создания нового Activity вручную включает следующие шаги:

- 1.Создаем новый xml файл разметки (например, activity_second.xml).
- 2.Создаем новый java-класс второго Activity (например, SecondActivity).
- 3.Прописываем логику SecondActivity.java (наследуемся от суперкласса, подключаем графическую разметку):

```
public class SecondActivity extends ActionBarActivity{ @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second);
    }
}
```

4.Регистрируем новое Activity в манифест-файле (это обязательно):

```
<activity
    android:name=".MainActivity" android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

```
<activity
    android:name=".SecondActivity" android:label="@string/app_name">
</activity>
```

Программный способ добавления нового компонента в Android- приложение реализуется с помощью меню File → New или путем нажатия комбинации клавиш Alt + Insert. Далее необходимо заполнить форму кастомизации нового Activity с указанием имен java-файла, xml-файла по аналогии с созданием Activity в новом проекте (см. рисунок 1.4). Запись в файле манифеста будет создана автоматически.

Явный вызов нового Activity.

Следующий шаг после создания нового Activity – научиться вызывать это Activity (например, в ответ на нажатие на одноименную кнопку в исходном Activity) (рисунок 2.4).

Для того чтобы из одного Activity вызвать другое, существует два способа: явный и неявный вызов. Далее рассмотрим оба способа.

Реализуем явный вызов нового Activity из основного:

```
Intent intent = new Intent(this, SecondActivity.class); startActivity(intent);
```

Намерение (Intent) – это механизм для описания одной операции (выбрать фотографию, отправить письмо, сделать звонок, запустить браузер и перейти по указанному адресу). В Android-приложениях многие операции работают через намерения. Наиболее распространенный сценарий использования намерения - запуск другой активности в своём приложении.

С помощью класса Intent явно указывают, какое Activity необходимо отобразить (это обычно используется внутри одного приложения):

```
Intent (Context packageContext, Class cls)
```

Context – это объект, который предоставляет доступ к базовым функциям приложения, таким как: доступ к ресурсам, к файловой системе, вызов Activity и т.д. Activity является подклассом Context, поэтому можно использовать ее – this.

Class – имя класса, указанное в манифест-файле. Так как при создании записи Activity в манифест-файле было указано имя класса, то это же имя следует указать во втором параметре Intent. Далее просмотрев манифест-файл система обнаружит соответствие и покажет соответствующее Activity.



Рисунок 2.4 – Пример вызова нового Activity в ответ на нажатие на одноименную кнопку в исходном Activity

Неявный вызов нового Activity.

В отличие от явного вызова, основанного на поиске нового Activity по имени класса в манифест-файле, неявный вызов работает с настройками Intent Filter искомого Activity в манифест-файле.

Intent Filter включает ряд параметров (action, data, category), комбинация которых определяет желаемую цель (отправка письма, открытие гиперссылки, редактирование текста, просмотр картинки, звонок по определенному номеру и т.д.).

Для настройки этих параметров в манифест-файле для нового Activity добавляем Intent Filter:

<activity

```
android:name=".SecondActivity" android:label="@string/app_name">
```

```
<intent-filter>
```

```
<action android:name="android.intent.action.SecondActivity" />
```

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

```
</intent-filter>
```

```
</activity>
```

Тогда неявно вызвать SecondActivity из MainActivity можно следующим образом:

```
intent = new Intent("android.intent.action.SecondActivity"); startActivity(intent);
```

Параметры объекта Intent из вышеприведенного кода совпадают с условиями фильтра в манифест-файле, и SecondActivity будет вызвано.

Следует отметить, что в случае с неявным вызовом поиск идет уже по всем Activity всех приложений в системе. Если таковых находится несколько, то система предоставляет выбор, какой именно программой необходимо воспользоваться.

Передача данных между Activity.

Передача данных между Activity осуществляется с помощью класса Intent. Сохранение данных в Intent реализуется с помощью метода putExtra(), а извлечение – getExtra().

Метод putExtra: добавляет к объекту пару: первый параметр – это ключ(имя), второй - значение.

Пример кода из MainActivity.java:

```
Intent intent = new Intent(this, SecondActivity.class); intent.putExtra("Name",  
et_Name.getText().toString()); intent.putExtra("Email",  
et_Email.getText().toString()); startActivity(intent);
```

Метод getExtra: извлекает значение по ключу. Пример кода из SecondActivity.java:

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

    Intent intent = getIntent();
    tv_Name.setText(intent.getStringExtra("Name"));
    tv_Email.setText(intent.getStringExtra("Email"));
}
```

Поместить в Intent можно данные не только типа String. В списке методов Intent:

<http://developer.android.com/reference/android/content/Intent.html#pubmethods>

можно посмотреть все многообразие типов, которые умеет принимать на вход метод putExtra.

Чтобы сохранить пользовательский тип данных используется метод putSerializable(), а извлечение – getSerializable() соответственно.

Метод startActivityForResult.

Запуск другой Activity не обязательно должен быть односторонним. Можно запустить другое Activity и получить обратно результат. Для этого используется метод startActivityForResult() вместо startActivity() (рисунок 2.5).

Алгоритм действий следующий:

1. Вызвать из исходного Activity новое Activity не с помощью метода startActivity, а с помощью startActivityForResult.

Синтаксис метода startActivityForResult: startActivityForResult(Intent

intent, int requestCode);

requestCode – идентификатор для определения, с какого Activity пришел результат.

2. Реализовать в новом Activity метод setResult для указания, какие данные необходимо вернуть в «родительское» Activity и вызвать finish().

Синтаксис метода setResult:

setResult(int resultCode, Intent intent);

resultCode – код возврата. Определяет успешно прошел вызов или нет. intent адресует данные в «родительское» Activity.

3. В исходном Activity в методе onActivityResult прописать логику обработки результата, который придет из нового Activity.

Синтаксис метода onActivityResult:

```
protected void onActivityResult(int requestCode, int resultCode, Intent data);
```

requestCode – тот же идентификатор, что и в startActivityForResult. По нему определяем, с какого Activity пришел результат. resultCode – код возврата. data – Intent, в котором возвращаются данные.

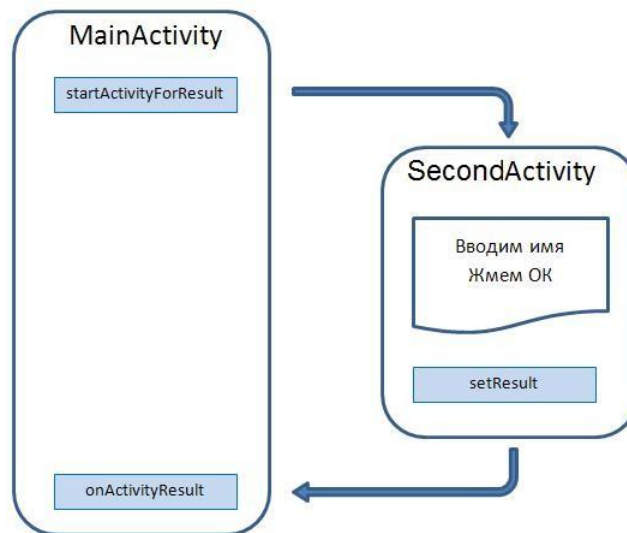


Рисунок 2.5 – Механизм вызова нового Activity и возврат из него с результатом

Например, необходимо из MainActivity вызвать SecondActivity, в SecondActivity ввести имя пользователя и вернуться в MainActivity, чтобы отобразить это имя в текстовом поле.

MainActivity.java:

@Override

```
public void onClick(View v) { switch (v.getId()) {
    case R.id.btnGotoActivity2:
        Intent intent = new Intent(this, SecondActivity.class);
        startActivityForResult(intent, 1);
        break;
```

```

    }
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data){
    if (data == null) { return;
    }
    String name = data.getStringExtra("Name"); tv_Name.setText(name);
}

```

SecondActivity.java:

```

@Override
public void onClick(View view) { Intent intent = new Intent();
    intent.putExtra("Name", et_Name.getText().toString());
    setResult(RESULT_OK, intent);
    finish();
}

```

Хранение данных в виде пары ключ, значение с помощью класса SharedPreferences.

Если имеется относительно небольшая коллекция пар ключ-значение, которую требуется сохранить, целесообразно использовать SharedPreferences API. Объект SharedPreferences указывает на файл, содержащий пары ключ-значение и обеспечивает простые методы для их чтения и записи. Каждый SharedPreferences файл управляется библиотекой и может быть личным или общим. Примером использования SharedPreferences может служить задача сохранения персональных данных пользователя и их восстановления при последующих запусках приложения.

Для реализации задачи хранения данных создадим два пользовательских метода saveText() – сохранение данных и loadText() – загрузка данных. Загрузку будем выполнять при запуске приложения (т.е. в методе onCreate), а сохранение – при закрытии приложения (т.е. в методе onDestroy).

Для сохранения данных сначала с помощью метода `getPreferences` получаем объект `sPref` класса `SharedPreferences`, который позволяет работать с данными (читать и писать):

```
sPref = getPreferences(MODE_PRIVATE);
```

Константа `MODE_PRIVATE` используется для настройки доступа и означает, что после сохранения, данные будут доступны только этому приложению.

Далее, чтобы редактировать данные, необходим объект `Editor` – получаем его из `sPref`:

```
SharedPreferences.Editor editor = sPref.edit();
```

В метод `putString` указываем наименование переменной – это константа `SAVED_TEXT`, и значение – содержимое поля `myEditText`. Чтобы данные сохранились, необходимо выполнить `commit`. И для наглядности выведем сообщение, что данные сохранены:

```
editor.putString(SAVED_TEXT, myEditText.getText().toString()); editor.commit();  
Toast.makeText(this, "Text saved", Toast.LENGTH_SHORT).show();
```

Для загрузки данных получаем объект `sPref` класса `SharedPreferences`. Чтение осуществляется с помощью метода `getString`: в параметрах указываем константу – это имя, и значение по умолчанию (пустая строка). Далее пишем значение в поле ввода `myEditText` и выведем сообщение, что данные считаны:

```
sPref = getPreferences(MODE_PRIVATE);  
String savedText = sPref.getString(SAVED_TEXT, "");  
myEditText.setText(savedText);  
Toast.makeText(this, "Text loaded", Toast.LENGTH_SHORT).show();
```

Работа приложения продемонстрирована на рисунке 2.6. Полный код приведен ниже:

```
public class MainActivity extends ActionBarActivity { private EditText myEditText;
```

```

SharedPreferences sPref;
final String SAVED_TEXT = "my_saved_text";

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

private void initView() {
    myEditText = (EditText) findViewById(R.id.myEditText);
}

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

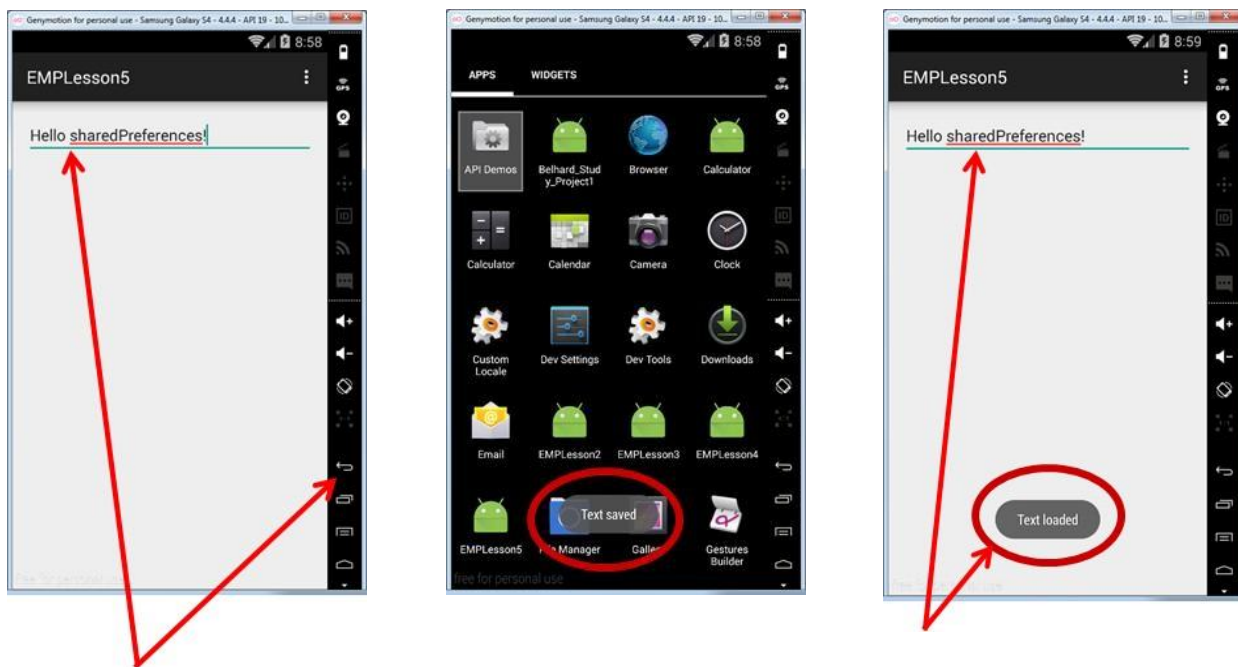
private void saveText() {
    sPref = getPreferences(MODE_PRIVATE); SharedPreferences.Editor
    editor = sPref.edit(); editor.putString(SAVED_TEXT,
    myEditText.getText().toString()); editor.commit();

    Toast.makeText(this, "Text saved", Toast.LENGTH_SHORT).show();
}

private void loadText() {
    sPref = getPreferences(MODE_PRIVATE);
    String savedText = sPref.getString(SAVED_TEXT, "");
    myEditText.setText(savedText);

    Toast.makeText(this, "Text loaded", Toast.LENGTH_SHORT).show();
}
}

```



а

б

в

Рисунок 2.6 – Хранение данных в виде пары: имя, значение
с использованием класса SharedPreferences:

а – сохраняем данные и выходим из приложения; б – выводится
всплывающее сообщение;

в – открываем приложение и видим восстановленную информацию

Практическое задание

1. Разработать приложение Taxi, состоящее из трех Activity (рисунок 2.7).
2. В первом Activity создать три редактируемых текстовых поля (EditText) для ввода пользователем регистрационных данных (телефона, имени и фамилии), кнопку Registration для запуска второго Activity.
3. При нажатии на кнопку Registration выполнить явный вызов второго Activity с передачей данных о пользователе (телефон, имя и фамилия).
4. Во втором Activity создать два текстовых поля (TextView) для вывода переданной информации о пользователе (имя+фамилия, телефон), пустое по умолчанию текстовое поле (TextView) для вывода маршрута движения, кнопку Set path для ввода этого маршрута, кнопку вызова такси Call Taxi (не- доступна, пока не введен маршрут движения).



Рисунок 2.7 – Пример работы приложения Taxi: а – исходное Activity (вид по умолчанию); б – исходное Activity с введенными регистрационными данными; в – второе Activity с выводом регистрационных данных; г – третье Activity для ввода маршрута движения; д – возврат во второе Activity для вызова такси; е – исходное Activity при повторных запусках

5. При нажатии на кнопку Set path выполнить неявный вызов третьего Activity с помощью метода startActivityForResult.

6. В третьем Activity создать шесть редактируемых текстовых полей (EditText) для ввода параметров маршрута движения, кнопку ОК для возврата во второе Activity.

7. При нажатии на кнопку ОК реализовать возврат во второе Activity с передачей в качестве результата параметров маршрута движения.

8. После возврата во второе Activity в текстовое поле вывести информация о маршруте движения и предложение вызвать такси, кнопку вызова такси Call taxi сделать доступной.

9. При нажатии на кнопку Call Taxi вывести всплывающее сообщение об успешной отправке такси.

10. Реализовать сохранение регистрационных данных пользователя в исходном Activity с помощью класса SharedPreferences и восстанавливать эту информацию при повторных запусках приложения. При этом название кнопки Registration должно программно меняться на Log in.

11. Вывести в лог очередность вызовов методов жизненного цикла первого, второго и третьего Activity.

12. Сделать вывод на основании логов о жизненном цикле Activity.

13. Продемонстрировать работу приложения Taxi на эмуляторе или реальном устройстве.

14. Дополнительное задание, предполагающее самостоятельное углубленное освоение материала: реализовать функционал geo-локации для автоматического определения исходной точки маршрута движения и/или Google-карт для обозначения конечной точки маршрута движения пользователя.

Содержание отчета

1. Скриншоты графических представлений первого, второго и третьего Activity в Android Studio, демонстрирующие логику работы приложения Taxi.

2. Код xml-файлов графических представлений первого, второго и третьего Activity приложения Taxi.

3. Код java-файлов первого, второго и третьего Activity приложения Taxi.

4. Результаты вывода в лог очередности вызовов методов жизненного цикла первого, второго и третьего Activity приложения Taxi.

Контрольные вопросы

1. Как с помощью класса Toast создать всплывающее сообщение?
2. В каких случаях необходимо логирование?
3. Что представляет собой окно LogCat? Какие существуют уровни логирования?
4. Как программно реализовать логирование?
5. Как создать новое Activity (опишите работу с java-классом, layout-файлом, файлом конфигурации AndroidManifest.xml)?
6. Для чего используется контекст приложения Context?
7. Для чего используются объекты класса Intent?
8. Как выполнить явный вызов Activity?
9. Как выполнить неявный вызов Activity?
10. Какие состояния предусмотрены жизненным циклом Activity?
11. Какие методы автоматически срабатывают при смене состояния Activity? Как можно использовать эти методы?
12. Как выполняется передача данных с помощью Intent?
13. В каком виде хранятся данные с помощью класса SharedPreferences?
14. В каких случаях целесообразно хранить данные с помощью класса SharedPreferences?

Лабораторная работа №7 Тестирование и оптимизация мобильного приложения

Цель: формирование у студентов знаний и навыков создания кастомизированных списков на основе собственного адаптера, реализации механизмов обратного вызова для отслеживания событий в многофункциональных Android-приложениях.

План занятия

1. Изучить теоретические сведения.
2. Выполнить практическое задание по лабораторной работе.
3. Оформить отчёт и ответить на контрольные вопросы.

Теоретические сведения

Для разработки многофункциональных интерфейсов Android-приложений необходимо научиться создавать свои собственные наборы View-элементов в отдельном xml-файле с последующим подключением созданной графики в родительскую разметку.

Создание View-элемента из содержимого layout-файла.

Класс `LayoutInflater` используется для создания View-элемента из содержимого отдельного layout-файла.

Алгоритм действий для достижения вышеуказанной цели следующий:

1. Создаем xml-файл с требуемым View-элементом или набором View-элементов (например, `my_view.xml`).
2. В Activity, к которой требуется добавить содержимое `my_view.xml`, находим id того layout, в котором и разместиться View-элемент (например, `lin_layout`).
3. В методе `onCreate` создаем объект класса `LayoutInflater`: `LayoutInflater`

`LayoutInflater = LayoutInflater.from(this);` 4. Вызываем из-под объекта

`LayoutInflater` метод `inflate`:

`View view = LayoutInflater.inflate(R.layout.my_view, lin_layout, true);`

Синтаксис метода `inflate`:

```
public View inflate (int resource, ViewGroup root, boolean attachToRoot);
```

resource – ID layout-файла, который будет использован для создания View.

root – родительский ViewGroup-элемент для создаваемого View.

attachToRoot – параметр, определяющий присоединять ли создаваемый View к root. Чтобы root стал родителем создаваемого View необходимо указать true.

Пример полного варианта кода с добавлением View-элемента из содержимого layout-файла как дочернего приведен ниже (рисунок 3.1).



Рисунок 3.1 – Пример первоначального интерфейса (а) и интерфейса с добавлением дочернего View-элемента из содержимого layout-файла (б)

```
public class MainActivity extends ActionBarActivity { private LinearLayout  
  
    lin_layout;  
    private LayoutInflater layoutInflater;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {
```

```

        super.onCreate(savedInstanceState); setContentView(R.layout.activity_main);
        initView();
    }

    private void initView() {
        lin_layout = (LinearLayout) findViewById(R.id.lin_layout);

        layoutInflater = LayoutInflater.from(this);
        View view = layoutInflater.inflate(R.layout.my_view, lin_layout, true);
    }
}

```

Список ListView.

Список `ListView` содержит однотипные по дизайну пункты и логику обработки взаимодействия с элементами этих пунктов (рисунок 3.2). Каждый пункт списка представляет собой `ViewGroup`, создаваемый в отдельном `layout`- файле.

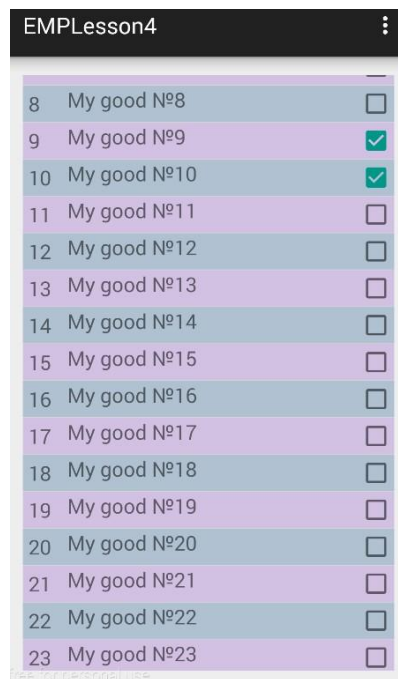


Рисунок 3.2 – Пример реализации списка `ListView`

Стратегия создания списка следующая:

1. Создаем в папке layout xml-файл с графическим отображением одного пункта списка. Для корректного отображения набора пунктов в результирующем списке в качестве layout_height указываем wrap_content или высоту в dp.
2. Создаем набор данных для последующего наполнения списка.
3. Програмируем свой собственный адаптер. Адаптер – структурный шаблон проектирования, предназначенный для создания класса-оболочки с требуемым интерфейсом.

Адаптеру назначаем набор данных для наполнения списка и layout-ресурс с визуальным представлением одного пункта списка.

4. Присваиваем адаптер списку ListView. Список при построении запрашивает у адаптера пункты, адаптер их создает (используя данные и layout-файл) и возвращает списку. В итоге мы видим готовый список.

Рассмотрим подробно вышеизложенную стратегию. В качестве примера разработаем магазин товаров (см. рисунок 3.2).

Создадим модель данных – класс Good с полями int id (номер товара), String name (наименование товара), boolean check (флаг, отображающий выбор товара пользователем), конструктором класса и методами getId(), getName(), isCheck(), setId(int id), setName(String name), setCheck(boolean check).

Разработаем графическое представление одного пункта списка – файл item_good.xml, содержащий два TextView и один CheckBox для отображения параметров товара.

В activity_main.xml в корневой layout добавляем ListView с размерами wrap content.

Создаем класс GoodsAdapter для реализации кастомизированного адаптера.

В MainActivity.java находим listView по id; создаем динамический массив ArrayList<Good> arr_goods и имитируем интернет-магазин путем генерации объектов класса Good. Создаем объект goodsAdapter от класса GoodsAdapter: для этого в конструктор класса передаем параметр context (т.е. this) и коллекцию объектов arr_goods. Далее присваиваем goodsAdapter списку с помощью метода setAdapter.

```
public class MainActivity extends ActionBarActivity { private ListView  
  
listView;  
private ArrayList<Good> arr_goods = new ArrayList<Good>();  
private final int SIZE_OF_ARR = 25;
```

```

private GoodsAdapter goodsAdapter;

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

private void initView() {
    listView = (ListView) findViewById(R.id.listView);
}

private void createMyListView() { fillData();
    goodsAdapter = new GoodsAdapter(this, arr_goods);
    listView.setAdapter(goodsAdapter);
}

private void fillData(){ int i=0;
    while (i<SIZE_OF_ARR) { i++;
        arr_goods.add(new Good(i," " + "My good №" + i, false));
    }
}
}

```

Рассмотрим реализацию класса GoodsAdapter. Наследуемся от базового адаптера BaseAdapter. Для обработки событий от чек-боксов реализуем интерфейс CompoundButton.OnCheckedChangeListener.

```

public class GoodsAdapter extends BaseAdapter implements CompoundButton.OnCheckedChangeListener{

    private Context context;

```

```

private ArrayList<Good> arr_goods_adapter; private LayoutInflater
layoutInflater;

public GoodsAdapter(Context context, ArrayList<Good>
arr_goods_adapter) {
    this.context = context; this.arr_goods_adapter = arr_goods_adapter;
    this.layoutInflater = LayoutInflater.from(context);
}

// кол-во элементов @Override
public int getCount() {
    return arr_goods_adapter.size();
}

// элемент по позиции @Override
public Object getItem(int position) { return arr_goods_adapter.get(position);
}

// id по позиции @Override
public long getItemId(int position) { return position;
}

// пункт списка @Override
public View getView(int position, View convertView, ViewGroup view-
Group) {
    View view = convertView; if (view == null) {
        view = layoutInflater.inflate(R.layout.item_good, null, false);
    }
}

```

```
Good good_temp = arr_goods_adapter.get(position);
```

```
TextView tv_goodId = (TextView)  
view.findViewById(R.id.tv_goodId);  
tv_goodId.setText(Integer.toString(good_temp.getId()));
```

```
TextView tv_goodName = (TextView)  
view.findViewById(R.id.tv_goodName);  
tv_goodName.setText(good_temp.getName());
```

```
CheckBox cb_good = (CheckBox) view.findViewById(R.id.cb_good);  
cb_good.setChecked(good_temp.isChecked()); cb_good.setTag(position);  
cb_good.setOnCheckedChangeListener(this);
```

```
return view;  
}
```

```
@Override  
public void onCheckedChanged(CompoundButton compoundButton, boolean  
isChecked) {  
    if (compoundButton.isShown()) {  
        int i = (int) compoundButton.getTag();  
        arr_goods_adapter.get(i).setCheck(isChecked); notifyDataSetChanged();  
    }  
}
```

В конструкторе заполняем внутренние переменные и получаем `LayoutInflater` для работы с layout-ресурсами. В `arr_goods_adapter` хранится список товаров.

Методы, отмеченные аннотацией `@Override`, необходимо реализовать при наследовании `BaseAdapter`. Эти методы используются списком и должны работать корректно.

Метод `getCount` должен возвращать количество элементов списка (в текущем примере это количество товаров).

Метод `getItem` должен возвращать элемент по указанной позиции. Используя позицию, получаем конкретный элемент из `arr_goods_adapter`.

Метод `getItemId` должен возвращать `id` элемента (возвращаем текущую позицию).

Метод `getView` должен возвращать `View` пункта списка. Для этого ранее создавался `layout-ресурс R.layout.item_good`. В этом методе необходимо из `R.layout.item_good` создать `View`, заполнить его данными и отдать списку. Но перед тем как создавать, пробуем использовать `convertView`, который идет на вход метода. Это уже созданное ранее `View`, но неиспользуемое в данный момент. Например, при прокрутке списка, часть пунктов уходит за экран и их уже не надо прорисовывать. `View` из этих «невидимых» пунктов используются для воссоздания графики и обновления данных. Это значительно ускоряет работу приложения, т.к. не надо прорисовывать `inflate` лишний раз.

Если же `convertView` не поступил (`null`), то создаем сами `view`.

Далее заполняем параметры товара, чекбоксу присваиваем обработчик, сохраняем в `Tag` позицию элемента. `Tag` – это подобие `Object`-хранилища у каждого `View`, куда можно поместить требуемые данные. В нашем случае для каждого чекбокса помещаем в его `Tag` номер позиции пункта списка. Далее в обработчике чекбокса будет возможность этот номер позиции извлечь и определить, в каком пункте списка был нажат чекбокс.

В итоге, метод `getView` возвращает списку полностью заполненное `view`, и список его отобразит как очередной пункт.

`onCheckedChanged` – обработчик для чекбоксов. При нажатии на чекбокс в списке он срабатывает, читает из `Tag` позицию пункта списка и помечает соответствующий товар как положенный в корзину. Без этого обработчика не работало бы помещение товаров в корзину, а на экране значения чекбоксов в списке терялись бы при прокрутке. Потому что пункты списка пересоздаются, если они уходят «за экран» и снова появляются. Это пересоздание обеспечивает метод `getView`, а он для заполнения `View` берет данные из товаров. Значит при нажатии на чекбокс, обязательно надо сохранить в данных о товаре то, что он теперь в корзине. Метод `notifyDataSetChanged()` уведомляет список об изменении данных для обновления списка на экране.

Header и Footer в списках.

`Header` и `Footer` – это `View`-элементы, которые могут быть добавлены к списку сверху и снизу. Для этого необходимо создать графические представления `header_mygoods.xml` и `footer_mygoods.xml`, преобразовать их в `View`-элементы и предоставить списку с помощью методов `addHeader` или

addFooter. Обязательным условием отображения Header и Footer является их добавление к списку до того, как присваивается адаптер.

Модифицируем код MainActivity.java:

```
public class MainActivity extends ActionBarActivity implements
View.OnClickListener {

    // добавляем новые переменные класса private LayoutInflater inflater;
    private View view_header, view_footer; private Button btnShow;
    private TextView tv_count;

    // добавляем Header и Footer private void createMyListView() {
        fillData();
        goodsAdapter = new GoodsAdapter(this, arr_goods, this);

        inflater = LayoutInflater.from(this);
        view_header = inflater.inflate(R.layout.header_mygoods, null);
        view_footer = inflater.inflate(R.layout.footer_mygoods, null); btnShow =
        (Button) view_footer.findViewById(R.id.btnShow);
        btnShow.setOnClickListener(this);
        tv_count = (TextView) view_footer.findViewById(R.id.tv_count);

        listView.addHeaderView(view_header); listView.addFooterView(view_footer);

        listView.setAdapter(goodsAdapter);
    }
```

Пример реализации списка с Header приведен на рисунке 3.3, а, с Footer – на рисунке 3.3, б.

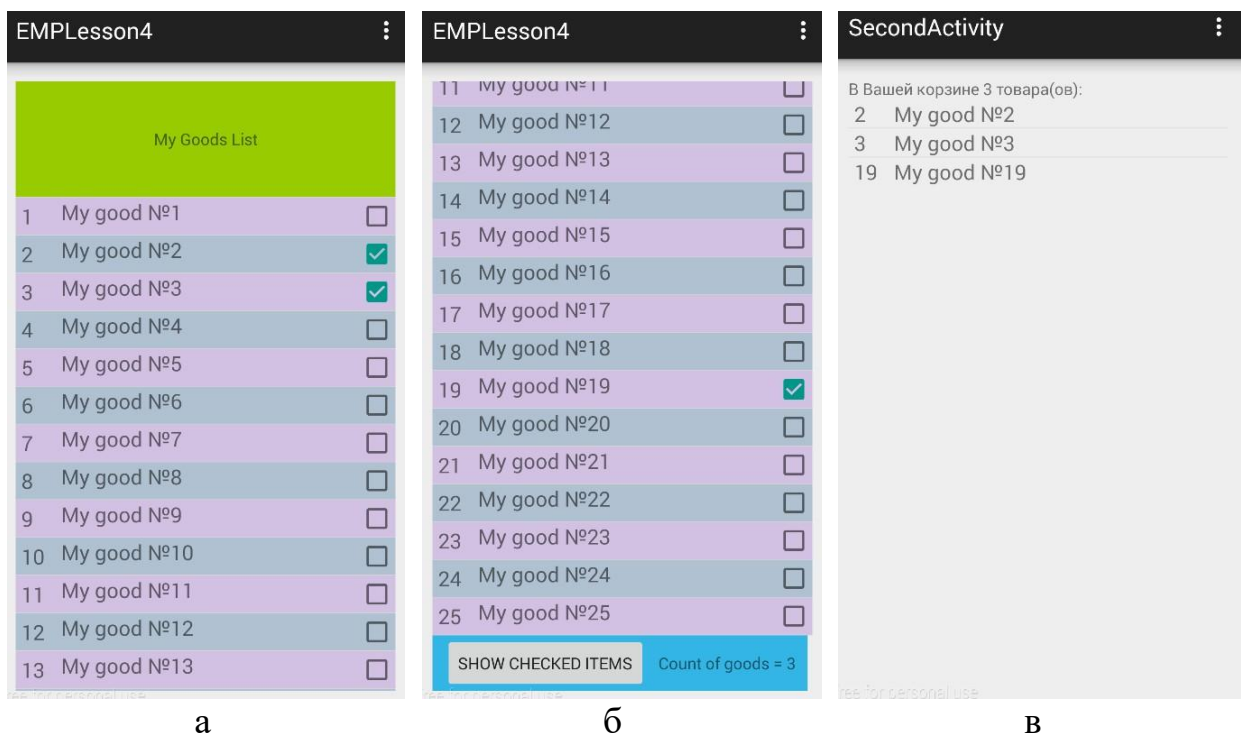


Рисунок 3.3 – Пример реализации списка ListView с Header (а), Footer (б) и переходом в корзину товаров (в)

Механизмы обратного вызова для обработки событий в Android-приложениях.

В продолжение примера со списком товаров магазина поставим следующую задачу: отмеченные товары должны сохраняться в отдельном динамическом массиве для последующего отображения корзины товаров пользователя в новом Activity (см. рисунок 3.3, в).

Особенностью реализации данной задачи является необходимость обработки события, наступление которого отслеживает класс GoodsAdapter, в классе MainActivity.

Простейший механизм callback (обратный вызов функции) позволяет вызывать из одного класса метод другого класса. Для этого достаточно иметь объект от класса, что дает полный доступ ко всем его методам.

В частности, в GoodsAdapter добавляем логику формирования динамического массива выбранных товаров `arr_checked_goods_adapter` и создаем метод `getCheckedGoods()`, возвращающий данный массив.

GoodsAdapter.java:

```
private ArrayList<Good> arr_checked_goods_adapter = new ArrayList<Good>();

@Override
public void onCheckedChanged(CompoundButton compoundButton, boolean
isChecked) {
    if (compoundButton.isShown()) {
        int i = (int) compoundButton.getTag();
        arr_goods_adapter.get(i).setCheck(isChecked); notifyDataSetChanged();

        if(isChecked){ arr_checked_goods_adapter.add(arr_goods_adapter.get(i));
        }else {
            arr_checked_goods_adapter.remove(arr_goods_adapter.get(i));
        }
    }
}

public ArrayList<Good> getCheckedGoods() { return
    arr_checked_goods_adapter;
}
```

Тогда в MainActivity реализуем логику обработки нажатия на кнопку Show checked items.

MainActivity.java:

```
private ArrayList<Good> arr_checked_goods = new ArrayList<Good>();

@Override
public void onClick(View view) {
    arr_checked_goods = goodsAdapter.getCheckedGoods();

    Intent intent = new Intent(this, SecondActivity.class);
```

```

        intent.putParcelableArrayListExtra("MyList", arr_checked_goods);
        startActivity(intent);
    }

```

Усложним задачу: количество выбранных товаров должно динамически изменяться в TextView, размещенным в Footer списка (см. рисунок 3.3, б). В этом случае потребуется использовать интерфейс в качестве прослойки для отслеживания событий. Такой подход является распространенным и подробно описан в шаблоне проектирования Наблюдатель (Observer). Для этого класс реализует специально созданный интерфейс; тем самым у класса появляется механизм, который позволяет получать экземпляру объекта этого класса оповещения от объектов других классов об изменении их состояния, тем самым наблюдая за ними.

Создаем свой собственный интерфейс:

```

public interface OnChangeListener { public void onChanged();

}

```

В MainActivity.java имплементируем данный интерфейс, а в методе onChanged() запрашиваем размер массива с выбранными пользователем товарами и выводим это число в соответствующий TextView.

```

public class MainActivity extends ActionBarActivity implements On-
ChangeListener, View.OnClickListener {

    @Override
    public void onChanged() {
        int size = goodsAdapter.getCheckedGoods().size();
        tv_count.setText("Count of goods = " + size + "");
    }
}

```

В конструктор GoodsAdapter.java необходимо передать объект от интерфейса OnChangeListener. Так как MainActivity.java имплементирует ука-

занный интерфейс, то в качестве объекта от интерфейса `OnChangeListener` выступает собственно `MainActivity.java`, т.е. передаем `this`:

```
goodsAdapter = new GoodsAdapter(this, arr_goods, this);
```

Изменения в `GoodsAdapter.java` следующие: добавляем в поля класса и, соответственно, конструктор, объект от интерфейса `OnChangeListener: onChangeListener`. В методе обработки нажатий на чекбоксы вызываем из-под объекта `onChangeListener` метод `onDataChanged()`.

`GoodsAdapter.java`:

```
private OnChangeListener onChangeListener;
```

```
public GoodsAdapter(Context context, ArrayList<Good> arr_goods_adapter,
OnChangeListener onChangeListener) { this.context = context;
    this.arr_goods_adapter = arr_goods_adapter; this.layoutInflater =
    LayoutInflater.from(context); this.onChangeListener = onChangeListener;
}
```

```
@Override
```

```
public void onCheckedChanged(CompoundButton compoundButton, boolean
isChecked) {
    if (compoundButton.isShown()) {
        int i = (int) compoundButton.getTag();
        arr_goods_adapter.get(i).setCheck(isChecked); notifyDataSetChanged();

        if(isChecked){ arr_checked_goods_adapter.add(arr_goods_adapter.get(i));
        }else {
            arr_checked_goods_adapter.remove(arr_goods_adapter.get(i));
        }
        onChangeListener.onDataChanged();
    }
}
```

Таким образом, при каждом нажатии на чек-бокс любого товара автоматически работает реализация метода `onDataChanged()` в `MainActivity` и произойдет перерасчет количества выбранных пользователем товаров.

Передаем Parcelable объекты с помощью Intent.

Для передачи динамического массива объектов между `Activity` используются следующие методы:

```
// для отправки данных intent.putParcelableArrayListExtra("MyList",  
arr_checked_goods);
```

```
// для извлечения данных  
arr_checked_goods = getIntent().getParcelableArrayListExtra("MyList"); При
```

этом в классе `Good` необходимо реализовать интерфейс `Parcelable`:

```
public class Good implements Parcelable{ private int id;  
    private String name; private boolean check;
```

```
    // обычный конструктор  
    public Good(int id, String name, boolean check){ this.id = id;  
        this.name = name; this.check = check;  
    }
```

@Override

```
public int describeContents() { return 0;  
}
```

```
// упаковываем объект в Parcel
```

@Override

```
public void writeToParcel(Parcel parcel, int i) { parcel.writeInt(id);
```

```

        parcel.writeString(name);
    }

    public static final Parcelable.Creator<Good> CREATOR = new
Parcelable.Creator<Good>() {
        // распаковываем объект из Parcel
        public Good createFromParcel(Parcel in) { return new Good(in);
        }

        public Good[] newArray(int size) { return new Good[size];
        }
    };

    // конструктор, считывающий данные из Parcel
    private Good(Parcel parcel) { id = parcel.readInt();
        name = parcel.readString(); check = false;
    }
}

```

В методе `writeToParcel` в объект `Parcel` упаковывают поля объекта `Good`. `CREATOR` типа `Parcelable.Creator<MyObject>` используется для создания экземпляра `Good` и заполнения его данными из `Parcel`. Для этого используется его метод `createFromParcel`, который необходимо реализовать. На вход поступает `Parcel`, а вернуть нужно готовый `Good`. В нашем примере используется конструктор `Good(Parcel parcel)`, который реализован отдельно.

Конструктор `Good(Parcel parcel)` принимает на вход `Parcel` и заполняет объект `Good` данными из `Parcel`.

Итоговая структура проекта приведена на рисунке 3.4.

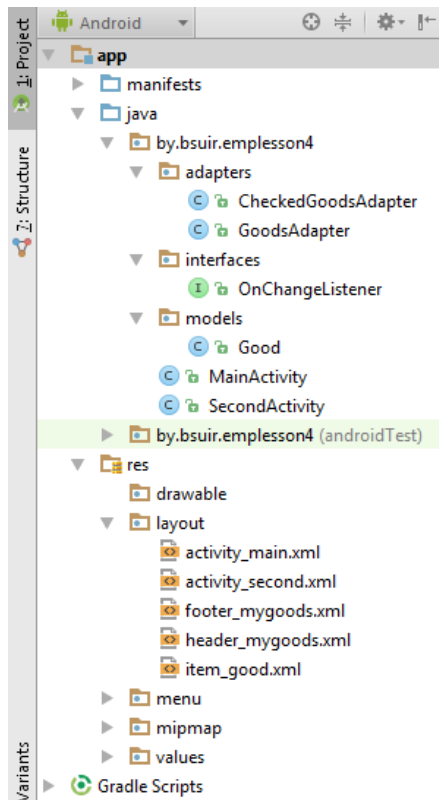


Рисунок 3.4 – Итоговая структура проекта MiniShop

Практическое задание

1. Разработать приложение MiniShop, состоящее из двух Activity (рисунки 3.3, 3.4).
2. В первом Activity создать список ListView с Header и Footer.
3. В Footer разместить текстовое поле (TextView) для ввода количества активированных пользователем товаров, кнопку Show Checked Items для перехода в корзину товаров.
4. Реализовать кастомизированный список ListView с помощью собственного адаптера, наследующего класс BaseAdapter.
5. В каждом пункте списка отобразить следующую информацию о товаре: идентификационный номер, название, стоимость, чек-бокс для возможности выбора товара пользователем.
6. В текстовом поле (TextView) Footer списка динамически отображать общее текущее количество активированных товаров.
7. При нажатии на кнопку Show Checked Items реализовать переход во второе Activity с корзиной товаров.

8. Корзину товаров реализовать в виде нового кастомизированного списка с выбранными товарами.

9. Продемонстрировать работу приложения MiniShop на эмуляторе или реальном устройстве.

Содержание отчета

1. Скриншоты графических представлений первого и второго Activity в Android Studio, демонстрирующие логику работы приложения MiniShop.

2. Код xml-файлов графических представлений, используемых в приложении MiniShop.

3. Код java-файлов приложения MiniShop, включая классы Activity, собственные интерфейсы, адаптеры, модели данных.

Контрольные вопросы

1. Как создать View-элемент из содержимого layout-файла? В каких случаях это необходимо?

2. Как создать и обеспечить работу списка ListView?

3. Как реализовать собственный кастомизированный список?

4. Что собой представляет и для чего нужен адаптер в Android-приложениях?

5. Какие методы класса BaseAdapter необходимо переопределить при создании кастомизированных списков?

6. Для чего нужен метод getView в адаптере?

7. Что собой представляет и как реализовать Header в списках?

8. Что собой представляет и как реализовать Footer в списках?

9. Какие Вы знаете механизмы обратного вызова для обработки событий в Android-приложениях?

10. Как передать динамический массив объектов из одного Activity в другое с помощью Intent?

Литература

Перечень основной литературы:

1. Введение в разработку приложений для ОС Android : учебное пособие / Ю. В. Березовская, О. А. Юфрякова, В. Г. Вологодина [и др.]. - Введение в разработку приложений для ОС Android, 2022-12-24. - Электрон. дан. (1 файл). - Москва: Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2021. - 427 с.

2. Семакова,, А. Введение в разработку приложений для смартфонов на ОС Android : учебное пособие / А. Семакова. - Введение в разработку приложений для смартфонов на ОС Android, 2022-12-24. - Электрон. дан. (1 файл). - Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2021. - 102 с

3. Нужный,, А. М. Разработка мобильных приложений на языке Java с использованием Android Studio: учебное пособие / А. М. Нужный, Н. И. Гребенникова, В. В. Сафронов. - Разработка мобильных приложений на языке Java с использованием Android Studio, 2026-09-08. - Электрон. дан. (1 файл). - Воронеж : Воронежский государственный технический университет, ЭБС АСВ, 2020. - 93 с.

Перечень дополнительной литературы:

1. Гарибов, А. И. Основы разработки приложений для мобильных устройств на платформе Windows Phone / А.И. Гарибов. - 2-е изд., испр. - Москва : Национальный Открытый Университет «ИНТУИТ», 2016. - 460 с.

2. Вязовик,, Н. А. Программирование на Java: учебное пособие / Н. А. Вязовик. - Программирование на Java, 2022-12-24. - Электрон. дан. (1 файл). - Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2021. - 601 с. - электронный. - Книга находится в премиум-версии ЭБС IPR BOOKS.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
по организации и проведению самостоятельной работы
по дисциплине «Программирование мобильных устройств»
для студентов направления подготовки
09.03.01 Информатика и вычислительная техника

Ставрополь 2026

ОБЩАЯ ХАРАКТЕРИСТИКА ДИСЦИПЛИНЫ

Цель и задачи дисциплины

Целями освоения дисциплины является изучение принципов построения архитектуры мобильных приложений и методов и средств проектирования программного обеспечения с использованием платформы Android.

Задачи освоения дисциплины:

изучить основы программирования мобильных устройств в современных средах программирования: создание модульных программ, элементы теории модульного программирования, объектно-ориентированное проектирование.

Место дисциплины в структуре образовательной программы

Дисциплина «Программирование мобильных устройств» относится к дисциплинам части, формируемой участниками образовательных отношений.

Перечень планируемых результатов обучения по дисциплине (модулю), соотнесённых с планируемыми результатами освоения образовательной программы

Код, формулировка компетенции	Код, формулировка индикатора	Планируемые результаты обучения по дисциплине (модулю), характеризующие этапы формирования компетенций, индикаторов
ПК-2. Способен осуществлять концептуальное, функциональное и логическое проектирование систем среднего и крупного масштаба и сложности	ИД-12 _{ПК-2} Разрабатывает программный код на языках программирования высокого уровня	Разрабатывает программный код на языках программирования высокого уровня с использованием платформы Android

Перечень компетенций с указанием этапов их формирования в процессе освоения образовательной программы

Код оцениваемой компетенции, индикатора (ов)	Этап формирования компетенции (№ темы) (в соответствии с рабочей программой дисциплины)	Средства и технологии оценки	Вид аттестация контроля, (текущий/промежуточный)	Тип контроля (устный, письменный или с использованием технических средств)	Наименование оценочного средства
ИД-12 _{ПК-2}	Тема 1-9	Вопросы для собеседования по лабораторным работам	текущий	устный	Вопросы для собеседования по лабораторным работам
ИД-12 _{ПК-2}	Тема 1-9	Вопросы для собеседования	текущий	устный	Вопросы для собеседования

2. ХАРАКТЕРИСТИКА САМОСТОЯТЕЛЬНОЙ РАБОТЫ

В основу самостоятельной работы студентов по дисциплине положено конспектирование лекций и рекомендуемых источников, подготовка к компьютерному тестированию знаний, оформление и подготовка к защите отчетов по лабораторным работам, выполнению разноуровневых индивидуальных заданий в ходе лабораторных работ. По каждому виду самостоятельной работы предусмотрены определённые формы отчетности, которые студенты предъявляют преподавателю в ходе лабораторных занятий или в часы индивидуальных консультаций.

Таблица 1 - Технологическая карта самостоятельной работы.

Коды реализуемых компетенций, индикатор(ов)	Вид деятельности студентов	Средства и технологии оценки	Объем часов, в том числе		
			СРС	Контактная работа с преподавателем	Всего
7 семестр					
ИД-12 _{ПК-2}	Подготовка к лабораторной работе	Собеседование	30	1.0	31
ИД-12 _{ПК-2}	Подготовка к лекции	Собеседование	5	1.0	6
ИД-12 _{ПК-2}	Самостоятельное	Собеседование	15	2.0	17

	изучение литературы				
Итого за 7 семестр			50	4	54
Итого			50	4	54

Для успешного освоения дисциплины, необходимо выполнить указанные в таблице виды самостоятельной работы, используя рекомендуемые источники информации.

1. Перечень основной литературы:

4. Введение в разработку приложений для ОС Android : учебное пособие / Ю. В. Березовская, О. А. Юфрякова, В. Г. Вологодина [и др.]. - Введение в разработку приложений для ОС Android, 2022-12-24. - Электрон. дан. (1 файл). - Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2021. - 427 с.

5. Семакова,, А. Введение в разработку приложений для смартфонов на ОС Android : учебное пособие / А. Семакова. - Введение в разработку приложений для смартфонов на ОС Android, 2022-12-24. - Электрон. дан. (1 файл). - Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2021. - 102 с

Нужный,, А. М. Разработка мобильных приложений на языке Java с использованием Android Studio : учебное пособие / А. М. Нужный, Н. И. Гребенникова, В. В. Сафронов. - Разработка мобильных приложений на языке Java с использованием Android Studio, 2026-09-08. - Электрон. дан. (1 файл). - Воронеж : Воронежский государственный технический университет, ЭБС АСВ, 2020. - 93 с.

2. Перечень дополнительной литературы:

3. Гарибов, А. И. Основы разработки приложений для мобильных устройств на платформе Windows Phone / А.И. Гарибов. - 2-е изд., испр. - Москва : Национальный Открытый Университет «ИНТУИТ», 2016. - 460 с.

4. Вязовик,, Н. А. Программирование на Java: учебное пособие / Н. А. Вязовик. - Программирование на Java, 2022-12-24. - Электрон. дан. (1 файл). - Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2021. - 601 с. - электронный. - Книга находится в премиум-версии ЭБС IPR BOOKS.

3. Перечень учебно-методического обеспечения самостоятельной работы обучающихся по дисциплине (модулю)

1. Программирование мобильных устройств: Методические рекомендации по самостоятельной работе: Направление подготовки Прикладная информатика.

Профиль подготовки - Прикладная информатика в экономике. Квалификация выпускника - бакалавр. Форма обучения - очная. Изучается в 5 семестре / Сев.-Кав. федер. ун-т. - Ставрополь : СКФУ, 2013. - 12 с.

4. Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимых для освоения дисциплины (модуля)

www.biblioclub.ru - Электронная библиотечная система «Университетская библиотека онлайн»

www.eLibrary.ru - Научная электронная библиотека

www.iprbookshop.ru - Электронно-библиотечная система IPRbooks

3. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ИЗУЧЕНИЮ ТЕОРЕТИЧЕСКОГО МАТЕРИАЛА

При самостоятельном изучении тем лекционных занятий и заданной литературы студентам рекомендуется:

- в день проведения занятий проработать, изученный на занятии учебный материал по конспекту. При этом необходимо выделить вопросы, на которые следует обратить большее внимание при дальнейшем изучении текущей темы. После проработки учебного материала необходимо ответить на, рекомендуемые контрольные вопросы;

- с целью качественного закрепления изученного материала в последующем следует более детально проработать учебный материал с использованием рекомендованной литературы. Студентам рекомендуется выделить вопросы, требующие более детальной проработки с помощью преподавателя;

- накануне проведения лекции студенту рекомендуется закрепить ранее изученный теоретический материал, подготовить вопросы, требующие уточнения у преподавателя.

Рекомендации по организации работы с литературой

Для овладения, закрепления и систематизации знаний студентам рекомендуется самостоятельная работа с рекомендованной литературой и конспектом лекций. При самостоятельной работе с рекомендованной литературой студентам рекомендуется проводить конспектирование учебного материала, изложенного в рекомендованных источниках.

Конспектирование источников проводится при детальном изучении темы, при этом студентам рекомендуется:

- дополнять конспект лекционного занятия путем его расширения по наиболее важным вопросам, рассмотренным на занятии.

Контроль этого вида самостоятельной работы осуществляется на учебных занятиях путем проверки преподавателем конспекта лекций и собеседования.

При конспектировании источников студенты должны исходить из рационального объема, конспектируемого материала. Конспект должен быть кратким и понятным при его использовании после изучения текущей темы.

Для успешного освоения дисциплины, необходимо самостоятельно детально изучить представленные темы по рекомендуемым источникам информации.

Работа по конспектированию источников оценивается удовлетворительно если:

- конспект имеет достаточный объем детализации по, изучаемой теме, обеспечивающий заданный уровень освоения теоретического материала дисциплины;
- конспект оформлен аккуратно, изучаемый материал изложен последовательно в соответствии с порядком изучения дисциплины, содержит обобщающие выводы по каждой теме.

4. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

При выполнении практикума по дисциплине (лабораторные работы) для достижения базового уровня сформированности компетенций студент должен уметь применять базовые знания по разделам дисциплины, владеть навыками самостоятельного решения учебных задач базового уровня.

При выполнении практикума по дисциплине (лабораторные работы) для достижения повышенного уровня сформированности компетенций студент должен уметь применять повышенные знания по разделам дисциплины, владеть навыками самостоятельного решения учебных задач повышенного уровня.

Методические указания при подготовке к лабораторной работе

Подготовку к лабораторной работе рекомендуется проводить в следующей последовательности:

- уяснить тему и цель, предстоящей лабораторной работы;
- изучить теоретический материал в соответствии с темой лабораторной работы (рекомендуется использовать рекомендованную литературу, конспект лекций, учебное пособие (практикум по лабораторным работам);
- ознакомиться с оборудованием и материалами, используемыми на лабораторной работе (при использовании специализированного оборудования

или программного обеспечения необходимо изучить порядок и правила его использования).

Методические указания при выполнении лабораторной работы

При выполнении лабораторной работы студенты должны строго соблюдать, установленные требования безопасности.

При выполнении лабораторной работы студентам рекомендуется:

- уяснить цель, выполняемых заданий и способы их решения;
- задания, указанные в лабораторной работе выполнять в той последовательности, в которой они указаны в лабораторном практикуме;
- при выполнении практического задания и изучении теоретического материала использовать помощь преподавателя;
- оформить отчет по лабораторной работе;
- ответить на контрольные вопросы.

Методические указания при подготовке к защите лабораторной работы

При подготовке к защите лабораторной работы студентам рекомендуется:

- подготовить отчет по лабораторной работе;
- подготовить обоснование, сделанных выводов;
- закрепить знания теоретического материала по теме лабораторной работы (рекомендуется использовать контрольные вопросы);
- знать порядок проведения расчетов (проводимых исследований);
- уметь показать и пояснить порядок исследований при использовании специализированного оборудования или программного обеспечения.

Защита лабораторной работы осуществляется путем собеседования студента с преподавателем. При собеседовании студент представляет на проверку отчет по лабораторной работе. Собеседование со студентом, претендующим на оценку «отлично» проводится по вопросам и практическим заданиям повышенного уровня обучения. Собеседование со студентом, не претендующим на оценку «отлично» проводится по вопросам и практическим заданиям базового уровня обучения.

5. ВОПРОСЫ ДЛЯ СОБЕСЕДОВАНИЯ ПО САМОСТОЯТЕЛЬНОМУ ИЗУЧЕНИЮ ЛИТЕРАТУРЫ

Вопросы (задача, задание) для проверки уровня обученности

1. Устройство платформы Android
2. Обзор сред программирования
3. Эмуляторы

4. Примеры приложений
5. Основные виды Android-приложений
6. Архитектура приложения, основные компоненты
7. Манифест приложения
8. Визуальный дизайн интерфейсов
9. Строительные блоки визуального дизайна интерфейсов
10. Элементы управления и дизайн навигации
11. Рекомендации по проектированию GUI под Android
12. Многооконные приложения
13. Работа с диалоговыми окнами
14. Особенности разработки приложения, содержащего несколько активностей
15. Перелистывание (Swipe)
16. Отличительные особенности смартфонов
17. Сенсорное (touch) управление
18. Работа с мультимедиа
19. Использование встроенной камеры
20. Взаимодействие с системами позиционирования
21. Другие сенсоры и датчики
22. Библиотеки
23. Обзор популярных библиотек
24. Безопасность использования подключаемых библиотек
25. Анимация
26. 2D и 3D графика
27. Основные принципы разработки игровых приложений для смартфонов
28. Обзор среды Intel® XDK
29. Используемые технологии
30. Общие вопросы тестирования и отладки
31. Особенности тестирования и отладки мобильных приложений
32. Основы тестирования приложений в Android IDE
33. Обзор дополнительных инструментов тестирования

1. Критерии оценивания компетенций (в соответствии с результатами освоения дисциплины)

Оценка «**отлично**» выставляется студенту, если студент дал полный, развернутый ответ на поставленные вопросы, при этом показана совокупность осознанных знаний по дисциплине, доказательно раскрыты основные положения вопросов; в ответе прослеживается четкая структура, логическая

последовательность, отражающая сущность раскрываемых понятий. Знание демонстрируется на фоне понимания его в системе дисциплины. Ответ изложен литературным языком с использованием технической терминологии. Могут быть допущены недочеты в определении понятий, исправленные студентом самостоятельно в процессе ответа, продемонстрировал высокое умение применять полученные знания на практике через решение конкретной задачи, свободно без затруднений справился с поставленной задачей, показав владение разносторонними приемами и навыками ее выполнения, не допустил ошибок и неточностей.

Оценка **«хорошо»** выставляется студенту, если студент дал полный, развернутый ответ на поставленные вопросы, при этом показано умение выделить существенные и несущественные признаки, причинно-следственные связи. Ответ четко структурирован, логичен, изложен литературным языком с использованием технической терминологии. Могут быть допущены 2-3 неточности или незначительные ошибки, исправленные студентом с помощью преподавателя, продемонстрировал умение применять полученные знания на практике через решение конкретной задачи, справился с поставленной задачей, показав владение необходимыми приемами и навыками ее выполнения, при этом допустил не более одной ошибки.

Оценка **«удовлетворительно»** выставляется студенту, если студент дал недостаточно полные и недостаточно развернутые ответы по вопросам билета. Логика и последовательность изложения имеют нарушения. Допущены ошибки в раскрытии понятий, употреблении терминов. Студент не способен самостоятельно выделить существенные и несущественные признаки и причинно-следственные связи. В ответе отсутствуют выводы. Умение раскрыть значение обобщенных знаний не показано. Речевое оформление требует поправок, коррекции, продемонстрировал посредственное умение применять полученные знания на практике через решение конкретной задачи, с трудом справился с поставленной задачей, при этом допустил не более двух ошибок.

Оценка **«неудовлетворительно»** выставляется студенту, если ответ представляет собой разрозненные знания с существенными ошибками по вопросу. Присутствуют фрагментарность, нелогичность изложения. Студент не осознает связь обсуждаемых вопросов с другими объектами дисциплины. Отсутствуют выводы, конкретизация и доказательность изложения. Речь неграмотная, гистологическая терминология не используется. Дополнительные и уточняющие вопросы преподавателя не приводят к коррекции ответа студента, ответ на вопросы полностью отсутствует или студент отказался от ответа, не продемонстрировал умение применять полученные знания на практике через решение конкретной задачи, не справился с поставленной задачей или допустил

при ее решении три и более серьезные ошибки.

2. Описание шкалы оценивания

Успеваемость студентов по каждой дисциплине оценивается в ходе текущего контроля и промежуточной аттестации.

3. Методические материалы, определяющие процедуры оценивания знаний, умений, навыков и (или) опыта деятельности, характеризующих этапы формирования компетенций

Процедура проведения данного оценочного мероприятия включает в себя:

- отдельное собеседование по каждой из указанных тем;
- составление конспекта.

Предлагаемые задания позволяют проверить компетенции **ПК-2**

Для подготовки к каждой теме следует изучить и законспектировать рекомендованные источники и научную литературу и на их основании подготовиться к указанным в методических материалах вопросам для обсуждения.

При проверке задания, оцениваются

- уверенное владение материалом;
- умение четко и логично излагать свои мысли;
- умение творчески подходить к решению основных вопросов темы;
- самостоятельность мышления,
- полное соответствие конспекта установленным требованиям;
- самостоятельная устная речь, полностью раскрывающая суть суть проблематики собеседования.

7. ВОПРОСЫ ДЛЯ СОБЕСЕДОВАНИЯ ПРИ ПРОВЕРКЕ ГОТОВНОСТИ К ЛАБОРАТОРНЫМ РАБОТАМ

Лабораторная работа №1. Знакомство с Android Studio

1. Как создать View-элемент из содержимого layout-файла? В каких случаях это необходимо?
2. Как создать и обеспечить работу списка ListView?
3. Как реализовать собственный кастомизированный список?
4. Что собой представляет и для чего нужен адаптер в Android-приложениях?
5. Какие методы класса BaseAdapter необходимо переопределить при создании кастомизированных списков?
6. Для чего нужен метод getView в адаптере?
7. Что собой представляет и как реализовать Header в списках?

8. Что собой представляет и как реализовать Footer в списках?
9. Какие Вы знаете механизмы обратного вызова для обработки событий в Android-приложениях?
10. Как передать динамический массив объектов из одного Activity в другое с помощью Intent?

Лабораторная работа 2. Разработка дизайна страницы с помощью языка XML. Менеджеры размещения.

1. Язык XML
2. Понятие Тег
3. Что такое Менеджер размещения
4. Элемент Grid
5. Элемент LinearLayout

Лабораторная работа 3. Использование механизма событий Android.

15. Как с помощью класса Toast создать всплывающее сообщение?
16. В каких случаях необходимо логирование?
17. Что представляет собой окно LogCat? Какие существуют уровни логирования?
18. Как программно реализовать логирование?
19. Как создать новое Activity (опишите работу с java-классом, layout-файлом, файлом конфигурации AndroidManifest.xml)?
20. Для чего используется контекст приложения Context?
21. Для чего используются объекты класса Intent?
22. Как выполнить явный вызов Activity?
23. Как выполнить неявный вызов Activity?
24. Какие состояния предусмотрены жизненным циклом Activity?
25. Какие методы автоматически срабатывают при смене состояния Activity? Как можно использовать эти методы?
26. Как выполняется передача данных с помощью Intent?
27. В каком виде хранятся данных с помощью класса SharedPreferences?
28. В каких случаях целесообразно хранить данные с помощью класса SharedPreferences?

Лабораторная работа № 4. Использование базы данных.

1. Библиотека SQLite
1. Виджет TextView
2. Переопределение метода
3. Наследование

Лабораторная работа 6. Работа с потоками.

1. Что такое Поток.
2. Понятие Task.

Лабораторная работа 7. Работа с несколькими окнами.

1. Intent.
2. Перегрузка методов.

Лабораторная работа 8. Создание меню.

1. Меню.
2. Обработка выбора пункта меню.
3. Перегрузка.

Лабораторная работа №9. Разрешения

1. Понятие виджета
2. Виджет TextView
3. Переопределение метода
4. Наследование

Лабораторная работа № 10. Тестирование и оптимизация мобильного приложения

11. Как создать View-элемент из содержимого layout-файла? В каких случаях это необходимо?
12. Как создать и обеспечить работу списка ListView?
13. Как реализовать собственный кастомизированный список?
14. Что собой представляет и для чего нужен адаптер в Android-приложениях?
15. Какие методы класса BaseAdapter необходимо переопределить при создании кастомизированных списков?
16. Для чего нужен метод getView в адаптере?
17. Что собой представляет и как реализовать Header в списках?
18. Что собой представляет и как реализовать Footer в списках?

19. Какие Вы знаете механизмы обратного вызова для обработки событий в Android-приложениях?

20. Как передать динамический массив объектов из одного Activity в другое с помощью Intent?

1. Критерии оценивания компетенций

оценка «отлично» выставляется, если студент продемонстрировал высокое умение применять полученные знания на практике через решение конкретной задачи с применением программирования мобильных устройств, свободно без затруднений справился с поставленной задачей, показав владение разносторонними приемами и навыками ее выполнения, не допустил ошибок и неточностей;

оценка «хорошо» выставляется, если студент продемонстрировал умение применять полученные знания на практике через решение конкретной задачи с применением программирования мобильных устройств, справился с поставленной задачей, показав владение необходимыми приемами и навыками ее выполнения, при этом допустил не более одной ошибки;

оценка «удовлетворительно» выставляется, если студент продемонстрировал посредственное умение применять полученные знания на практике через решение конкретной задачи с применением программирования мобильных устройств, с трудом справился с поставленной задачей, при этом допустил не более двух ошибок;

оценка «неудовлетворительно» выставляется, если студент не продемонстрировал умение применять полученные знания на практике через решение конкретной задачи с применением программирования мобильных устройств, не справился с поставленной задачей или допустил при ее решении три и более серьезные ошибки.

2. Описание шкалы оценивания

Успеваемость студентов по каждой дисциплине оценивается в ходе текущего контроля и промежуточной аттестации.

3. Методические материалы, определяющие процедуры оценивания знаний, умений, навыков и (или) опыта деятельности, характеризующих этапы формирования компетенций

Процедура проведения данного оценочного мероприятия включает в себя:

- собеседование;
- защита лабораторных работ.

Предлагаемые задания позволяют проверить компетенции **ПК-2**.

К лабораторному занятию студент должен подготовить ответы на вопросы, проработать дополнительные научные источники, сделать конспект

теоретического материала (если это предусмотрено).

Максимальное количество баллов студент получает, если он выполнит лабораторное задание, владеет материалом, умеет логично и четко излагать мысли, показывает самостоятельность выполнения задания.

Основанием для снижением оценки являются:

- слабое знание темы и неполностью выполненное задание;
- невыполненное задание к лабораторной работе;
- отсутствие умения применить теоретические знания для решения лабораторных задач.

При проверке задания, оцениваются

- активное участие в работе;
- уверенное владение материалом;
- умение четко и логично излагать свои мысли;
- умение творчески подходить к решению заданий;
- самостоятельность мышления,
- самостоятельная устная речь, полностью раскрывающая суть сути проблематики собеседования.