

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО  
ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное  
образовательное учреждение высшего образования  
«Воронежский государственный технический  
университет»

А. М. Нужный, Н. И. Гребенникова, В. В. Сафронов

РАЗРАБОТКА МОБИЛЬНЫХ ПРИЛОЖЕНИЙ  
НА ЯЗЫКЕ JAVA С ИСПОЛЬЗОВАНИЕМ  
ANDROID STUDIO

Учебное пособие



Воронеж 2020

УДК 681.3.06(07)  
ББК 32.973-018я7  
Н24

Рецензенты:

кафедра вычислительной техники и информационных систем  
Воронежского государственного лесотехнического университета  
им. Г. Ф. Морозова (зав. кафедрой д-р техн. наук, проф. В. К. Зольников);  
канд. техн. наук П. Ю. Гусев

**Нужный, А.М.**

Н24 Разработка мобильных приложений на языке Java с использованием Android Studio: учебное пособие [Электронный ресурс]. - Электрон, текстовые и граф. данные (3,14 Мб) / А. М. Нужный, Н. И. Гребенникова, В. В. Сафронов. - Воронеж: ФГБОУ ВО «Воронежский государственный технический университет», 2020. 1 электрон. опт. диск (CD-ROM): цв. – Систем. требования: ПК 500 и выше; 256 Мб ОЗУ; Windows 7; SVGA с разрешением 1024x768; Adobe Acrobat; CD-ROM дисковод; мышь. - Загл. с экрана.

ISBN 978-5-7731-0906-8

В пособии рассматриваются базовые аспекты разработки приложений в среде Android Studio на языке Java.

Издание предназначено для студентов направления подготовки бакалавров 09.03.01 «Информатика и вычислительная техника» (профили «Вычислительные машины, комплексы, системы и сети», «Системы автоматизированного проектирования», «Системы автоматизированного проектирования в машиностроении») при изучении дисциплин «Программирование на языке Java», «Среды визуального программирования».

Ил. 80. Табл. 2. Библиогр.: 3 назв.

**УДК 681.3.06(07)**  
**ББК 32.973-018я7**

*Издается по решению редакционно-издательского совета  
Воронежского государственного технического университета*

ISBN 978-5-7731-0906-8

© Нужный А. М., Гребенникова Н. И.,  
Сафронов В. В., 2020  
© ФГБОУ ВО «Воронежский  
государственный технический  
университет», 2020

## ВВЕДЕНИЕ

При всем разнообразии направлений использования языка Java, начиная от разработки серверных приложений в таких корпорациях как Citigroup, Deutsche Bank, Barclays, и заканчивая приложениями для встраиваемых систем типа смарт-карт или сенсоров, на сегодняшний день наиболее интересной, динамично развивающейся и востребованной является область разработки мобильных приложений для платформы Android.

Данное учебное пособие содержит теоретические сведения и указания по выполнению практических работ, позволяющие овладеть навыками по установке, настройке и разработке простых приложений для мобильных устройств в среде разработки Android Studio.

В пособии кратко рассмотрены принципы работы операционной системы Android, описаны компоненты Java-приложений для этой системы, принципы их взаимодействия и приемы создания.

Приведено описание основных элементов графического интерфейса и описаны способы работы с ними. Рассмотрены аспекты взаимодействия пользовательских приложений со стандартными приложениями ОС Android, методы работы со всевозможными мультимедийными данными.

Указания по выполнению практических работ содержат детальные указания по практическому использованию возможностей Android Studio, методов управления элементами графического интерфейса, приемов организации взаимодействия с системными приложениями.

# ТЕМА 1. СОЗДАНИЕ ПЕРВОГО ПРИЛОЖЕНИЯ ANDROID

Цель изучения: формирование базовых навыков по установке и настройке среды разработки Android Studio, созданию и отладке простого Java-приложения.

## 1.1. Теоретическая часть

### 1.1.1. Краткие сведения об Android

Android - бесплатная операционная система, основанная на ядре Linux и оригинальной реализации виртуальной машины Java, выполненной компанией Google. Приложения для Android пишутся преимущественно на языках программирования Java и, в последнее время, Kotlin. Инструменты Android SDK (Software Development Kit – комплект разработки программного обеспечения) компилируют код Java, все требуемые файлы данных и ресурсов в файл APK – программный пакет Android, который представляет собой файл архива с расширением .apk.

В файле APK находится все, что требуется для работы Android-приложения. Он позволяет установить приложение на любом устройстве под управлением системы Android.

При разработке на Java могут использоваться различные среды разработки, такие как Android Studio, Eclipse, IntelliJ IDEA, но следует учитывать, что с 2016 официальная поддержка плагинов Eclipse под Android прекратилась, а Android Studio объявлена Google официальной средой разработки под Android.

Android Studio, по сути, — известная Java IDE IntelliJ IDEA с плагинами, ориентированными на разработку под Android. Актуальная версия доступна для скачивания по ссылке: <https://developer.android.com/studio/index.html>.

Android Studio предоставляет возможность работать с компонентами пользовательского интерфейса при помощи перетаскивания, обеспечивает функцию предпросмотра макета, обладает системой автоматической сборки Gradle, позволяет генерировать различные виды сборок и генерацию нескольких .apk файлов.

Также в Android Studio имеются средства рефакторинга кода, утилита для подписывания приложений, шаблоны основных макетов и компонентов Android.

Системные требования Android Studio приведены в табл.1.

Таблица 1

Системные требования Android Studio

	Windows	OS X	Linux
Версия OS	Microsoft Windows 10/8/7/Vista/2003 (32 или 64-bit)	Mac® OS X® 10.8.5 или выше, до 10.13 / 10.14 (High Sierra/ Mojave)	GNOME или KDE
Оперативная память	3 ГБ (минимум), 8 ГБ (рекомендуется); +1 ГБ для Android Emulator		
Свободное место на диске	2 ГБ минимум (500 МБ для IDE + 1.5 ГБ для Android SDK и образа системы эмулятора), 4 ГБ SSD рекомендуемое		
Версия JDK	<a href="#">Java Development Kit 8</a>		
Разрешение экрана	1280 x 800 (минимум)		
Дополнительно	—	<a href="#">Java Runtime Environment</a> (JRE) 6	GNU C Library (glibc) 2.15 или выше

Любое Android-приложение на устройстве работает в изолированной программной среде (в "песочнице"). Это реализовано за счет того, что Android представляет собой

многопользовательскую систему Linux, в которой каждое приложение рассматривается как отдельный пользователь. Каждому приложению присваивается уникальный идентификатор пользователя и полномочия для всех файлов приложения назначаются таким образом, чтобы доступ к ним был разрешен только пользователю с данным идентификатором.

Каждый процесс Linux выполняется на собственной виртуальной машине (VM) изолированно от других приложений, при этом каждое приложение выполняется в собственном процессе Linux. Android запускает процесс, когда требуется выполнить какой-либо компонент приложения, а затем завершает процесс, когда он больше не нужен, либо, когда системе требуется освободить память для других приложений.

Система Android реализует принцип предоставления минимальных прав, т.е., приложение по умолчанию имеет доступ только к тем компонентам, которые ему необходимы для работы. Таким образом формируется исключительно безопасная среда, в которой приложение не имеет доступа к недозванным областям системы.

Однако у приложения есть варианты предоставления своих данных другим приложениям и доступа к системным службам. Приложение может запросить разрешение на доступ к данным устройства, например, к контактам пользователя, SMS-сообщениям, подключаемой карте памяти (SD-карте), камере, Bluetooth и др. Все разрешения должны предоставляться приложению при его установке. Кроме этого, группе приложений, подписанных одним сертификатом, может быть назначен одинаковый идентификатор пользователя, что позволяет им совместно использовать ресурсы и выполняться в рамках одной VM.

### 1.1.2. Компоненты Android-приложения

Компоненты являются строительными блоками для создания приложений Android и представляют собой точки входа, через которые система может взаимодействовать с приложениями.

Существуют следующие типы компонентов:

- операции;
- службы;
- поставщики контента;
- приемники широковещательных сообщений.

**Операция Activity** — основной компонент приложения, используемый для организации взаимодействия с пользователем и формирующий окно для организации пользовательского интерфейса.

Стандартное окно является полноэкранным, однако его размер может быть меньше, и оно может размещаться поверх других окон.

Обычно приложение содержит несколько операций, слабо связанных между собой.

Одна из операций в приложении обозначается как «основная» и предоставляется пользователю при первом запуске приложения. При этом, каждая операция может запустить другую операцию для выполнения различных действий. При запуске новой операции предыдущая операция останавливается системой и сохраняется в так называемом «стеке переходов назад», работающем по принципу «последним вошёл — первым вышел». После завершения пользователем текущей операции и нажатия кнопки «Назад», последняя операция удаляется из стека (и уничтожается), и возобновляется предыдущая операция.

**Служба (Service)** - компонент, работающий в фоновом режиме и предназначенный для выполнения длительных

операции, например, фоновое прослушивание музыки, загрузка данных по сети. Служба не имеет интерфейса и запускается другим компонентом, который в дальнейшем может взаимодействовать с ней, – например операцией.

**Поставщик контента (Content provider)** – компонент для управления набором данных, хранимых в файловой системе, базе данных SQLite, Интернете или любом другом месте, к которому у приложения имеется доступ. Посредством поставщика контента другие приложения могут запрашивать или даже изменять данные. Так, в Android есть поставщик контента, управляющий информацией контактов пользователя. Приложение, получившее соответствующие разрешения, может запросить часть этого поставщика контента (например, ContactsContract.Data) для чтения и записи сведений об определенном контакте.

**Приемник широковещательных сообщений (Broadcast receiver)** представляет собой компонент, который реагирует на объявления, распространяемые по всей системе. Объявления могут быть как системными (разряжен аккумулятор, сделан фотоснимок, выключился экран), так и рассылаться приложениями (данные загружены и готовы к использованию).

Приемники широковещательных сообщений не имеют пользовательского интерфейса, но они могут создавать уведомления в строке состояния, чтобы предупредить пользователя о событии "рассылка объявления". Однако чаще всего они являются просто "шлюзом" для других компонентов и предназначены для выполнения минимального объема работы [1].

### 1.1.3. Запуск и взаимодействие компонентов

Особенностью системы Android является то, что любое приложение может запустить компонент другого приложения.

Так как каждое приложение выполняется системой в отдельном процессе с такими правами доступа к файлам, которые ограничивают доступ в другие приложения, пользовательское приложение не может напрямую вызвать компонент из другого приложения. Это может сделать сама система Android. Поэтому, чтобы вызвать компонент в другом приложении, необходимо сообщить системе о своем намерении (**Intent**) запустить определенный компонент. После этого система активирует требуемый компонент.

Объекты Intent представляют собой асинхронные сообщения, которые связывают друг с другом отдельные компоненты во время выполнения и могут быть представлены для наглядности в виде мессенджеров, посылающих другим компонентам запрос на выполнение действий.

Для операций и служб Intent определяет действие, которое требуется выполнить (например, просмотреть (view) или отправить (send) что-то), а также может указывать **URI (Uniform Resource Identifier** – унифицированный идентификатор ресурса) данных, с которыми это действие нужно выполнить. Например, объект Intent может передавать запрос на выполнение операции "показать изображение" или "открыть веб-страницу".

Для запуска компонентов приложения системе Android необходимо знать, что компонент существует. Информация о них хранится в файле манифеста приложения AndroidManifest.xml. Файл должен находиться в корневой папке приложения и содержит объявления всех компонентов приложения, а также определяет разрешения приложения,



Первый метод, с которого начинается выполнение activity – **onCreate**. Он переводит объект Activity в состояние Created. Метод обычно переопределяется и является обязательным. Он содержит первоначальную настройку, в частности описание элементов визуального интерфейса. Помимо этого, метод получает объект **Bundle**, содержащий прежнее состояние Activity, если оно было сохранено. Это позволяет, например, вернуться к выбранной записи в списке контактов после ее редактирования, а не начинать просмотр всех записей сначала.

**Метод onStart()** выполняет подготовку Activity к выводу на экран устройства и обычно не переопределяется.

**Метод onResume()** переводит Activity в состояние Resumed, когда пользователь может с ней взаимодействовать. В этом состоянии Activity остается, пока не потеряет фокус.

Метод **onPause()** вызывается, если пользователь переходит к другой Activity. Он освобождает используемые ресурсы и приостанавливает процессы. Activity становится невидимой, не отображается на экране, но продолжает быть активной и может быть вновь выведена на экран методом **onResume()**.

Метод **onSaveInstanceState()** вызывается после метода **onPause()**, но до вызова **onStop()**. В **onSaveInstanceState** производится сохранение состояния приложения в передаваемый в качестве параметра объект Bundle.

Метод **onStop()** переводит Activity в состояние Stopped. В методе **onStop** следует освобождать используемые ресурсы, которые не нужны пользователю, когда он не взаимодействует с Activity. Здесь также можно сохранять данные, например, в базу данных. После работы этого метода работа с Activity возможна при помощи методов **onRestart()** или **onCreate()**.

Завершается работа активности вызовом метода **onDestroy**, который возникает в случае, если система решит убить **activity**, либо при вызове метода **finish()**.

Также следует отметить, что при изменении ориентации экрана система завершает **activity** и затем создает ее заново, вызывая метод **onCreate**.

## 1.2. Практическая часть

### 1.2.1. Установка средств разработки. Установка JDK

В первую очередь для создания Android-приложений следует загрузить и установить пакет JDK (Java Development Kit), который необходим для разработки на языке Java.

JDK можно найти на сайте компании Oracle: <http://www.oracle.com/technetwork/java/javase/downloads/index.html>.



Рис. 2. Страница загрузки Java Platform

## 1.2.2. Настройка среды окружения

После установки JDK в ОС Windows следует добавить в переменную среды окружения Path путь к исполняемым файлам JDK (например, «с:\Program Files\Java\jdk1.7.0\_71\bin\»), а также создать или отредактировать системную переменную JAVA\_HOME, значение которой также хранит путь к исполняемым файлам JDK.

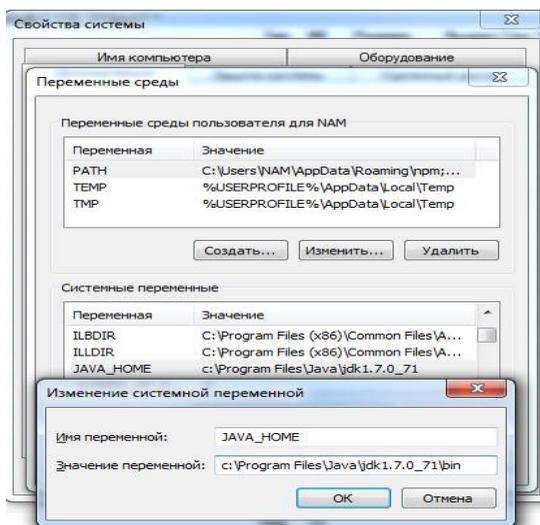


Рис. 3. Настройка переменных среды окружения

## 1.2.3. Установка Android Studio

Следующим этапом является установка IDE Android Studio, доступной для загрузки по адресу:

<https://developer.android.com/studio>

Для загрузки следует принять лицензионное соглашение.

После загрузки установочный файл следует запустить от имени администратора и следовать указаниям установщика.

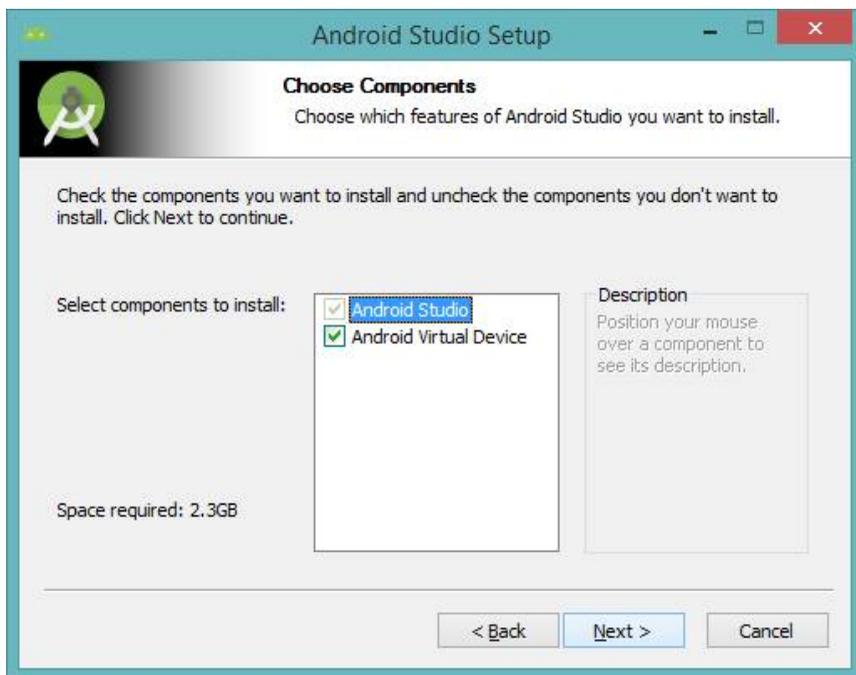


Рис. 4. Установка Android Studio

По окончании установки будет предложено запустить Android Studio, и выполнится установка Android SDK.

**Внимание!** Android SDK по умолчанию устанавливается в каталоги профилей пользователя, что, в случае использования переносимых профилей, приведет к значительному росту их размера и, как следствие, недопустимому увеличению времени загрузки компьютера.

В связи с этим, в лабораториях кафедры АВС запуск Android Studio осуществляется от имени локального администратора и выполняется преподавателем.

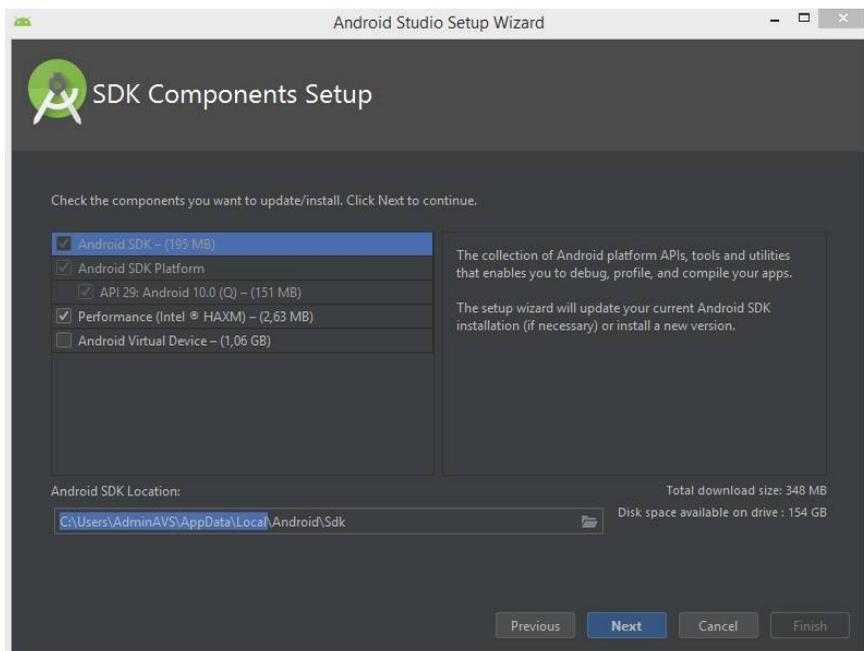


Рис. 5. Установка Android SDK

На этом этапе можно включить и установку эмулятора (виртуального устройства Android).

В процессе установки система попросит определить объем дискового пространства для Hardware accelerated execution manager. Не следует указывать объем памяти меньше рекомендованного.

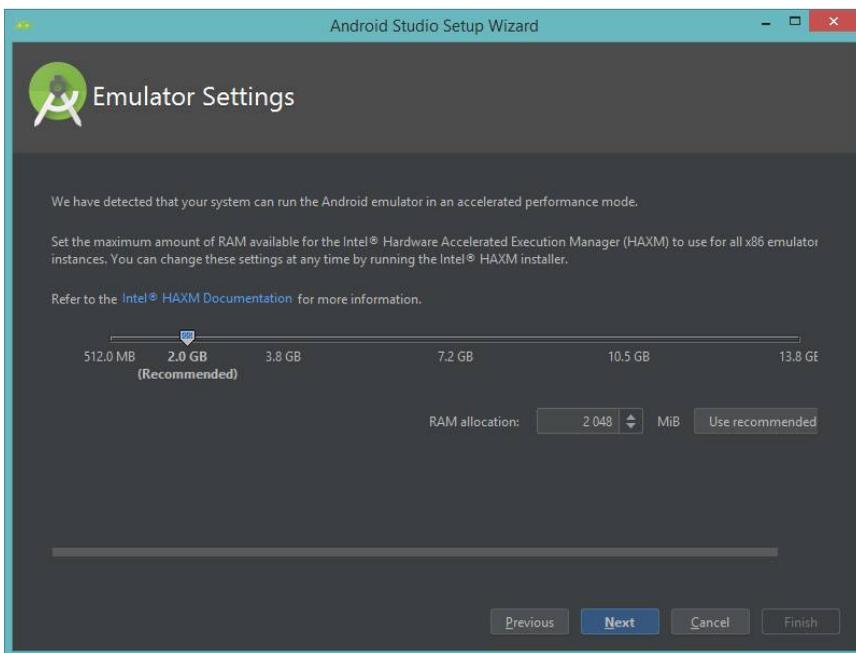


Рис. 6. Конфигурирование

По завершении установки будет предложено создать новое приложение.

## 1.2.4. Создание первого приложения

### 1.2.4.1. Выбор шаблона

В главном меню программы выберите File-New-New Project.

Android Studio предоставляет ряд шаблонов для различных ситуаций, но самыми актуальными для разработки нового приложения являются Basic Activity и Empty Activity. Выберите шаблон Empty Activity на закладке Phone and Tablet и нажмите кнопку **Next**.

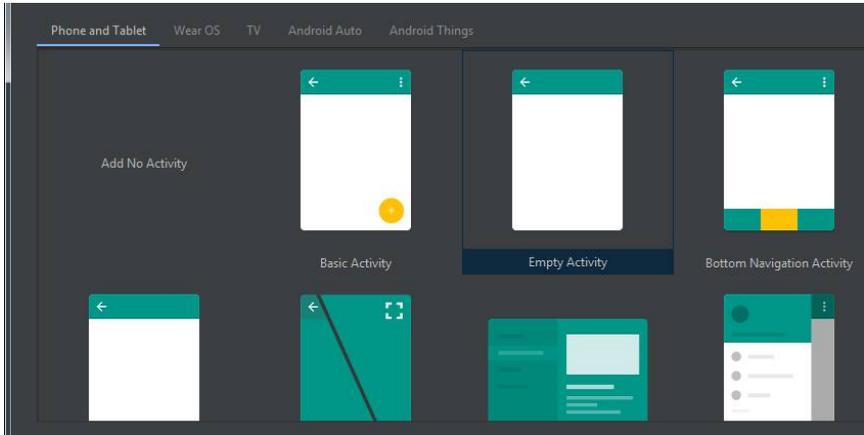


Рис. 7. Выбор шаблона

#### 1.2.4.2. Конфигурирование проекта

В следующем окне осуществляется настройка основных параметров проекта.

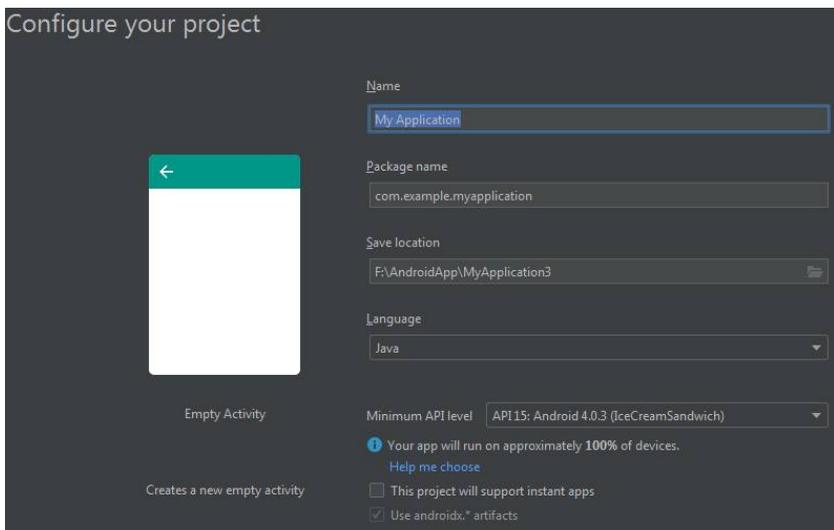


Рис. 8. Окно конфигурирования проекта

В поле **Name** следует указать имя приложения, которое будет в дальнейшем отображаться в заголовке.

Поле **Package Name** по умолчанию содержит имя пакета **com.example.имя приложения** в формате доменных имен, характерном для Java. В дальнейшем это имя будет преобразовано в структуру каталогов по пути, указанному в поле Save location. Имя часто использует обратное доменное имя компании-разработчика. Например, «com.example.myapplication» - пакет с именем mypackage, созданный программистом в компании example.com.

В поле Language следует указать Java, а в поле Minimum API level можно указать API 21: Android 5.0 (Lollipop).

Выбор минимального уровня, поддерживаемого API, определяется данными об использовании различных версий ОС Android [2].

Таблица 2

Процент устройств с поддержкой различных версий Android

Android platform version		API level	Cumulative distribution
10.0	Android 10	29	8.2%
9.5	Pie	28	39.5%
8.1	Oreo	27	53.5%
8.0	Oreo	26	60.8%
7.1	Nougat	25	66.2%
7.0	Nougat	24	73.7%
6.0	Marshmallow	23	84.9%
5.1	Lollipop	22	92.3%
5.0	Lollipop	21	94.1%
4.4	KitKat	19	98.1%
4.3	Jelly Bean	18	98.4%
4.2	Jelly Bean	17	99.2%
4.1	Jelly Bean	16	99.8%

По окончании настройки нажмите кнопку Finish. Вариант настройки проекта приведен на рис. 9.

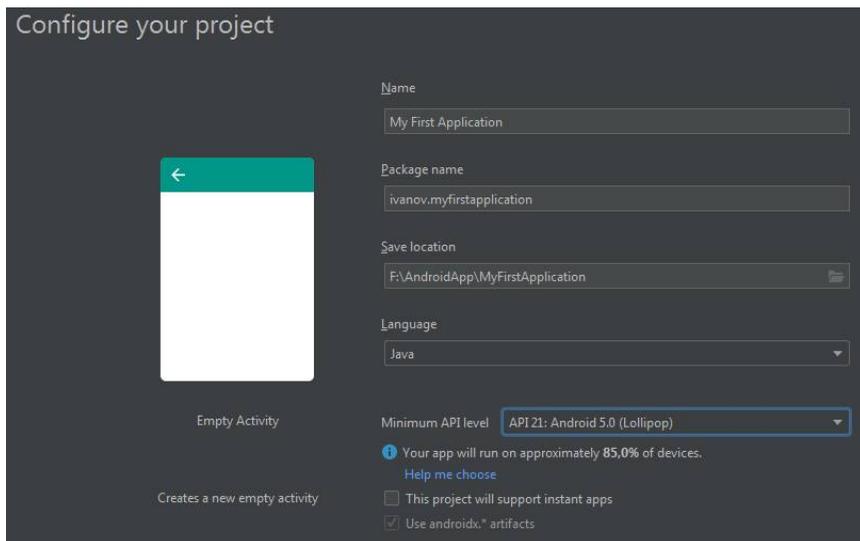


Рис. 9. Вариант настройки проекта

### 1.2.4.3. Работа с проектом

Обратите внимание, что новый проект создается в новом окне Android Studio. Для высвобождения ресурсов исходное окно рекомендуется закрыть.

Через некоторое время, когда будет создана структура проекта, окно Android Studio примет приблизительно следующий вид:

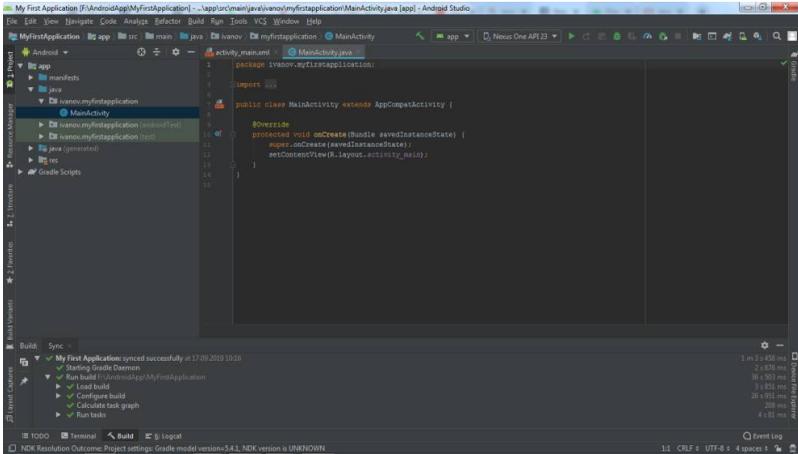


Рис. 10. Первоначальное отображение проекта в Android Studio

В правой части окна отображаются главные файлы проекта:

- MainActivity.java – содержит логику приложения;
- activity\_main.xml - содержит определение графического интерфейса приложения, доступен для редактирования в режимах Design или Text.

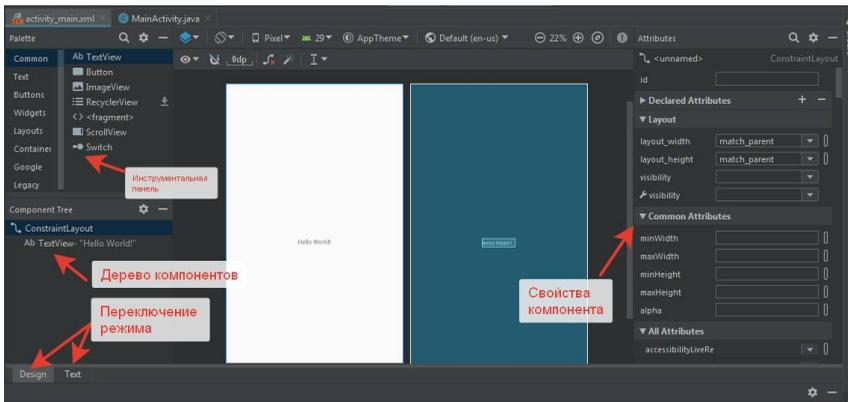


Рис. 11. Элементы редактирования файла activity\_main.xml в режиме Design

Уже в исходном состоянии приложение может быть запущено в эмуляторе. Для этого, если эмулятор уже установлен, следует нажать пиктограмму «Run App»  или Shift+F10. Произойдет запуск эмулятора и выполнение нашей программы.

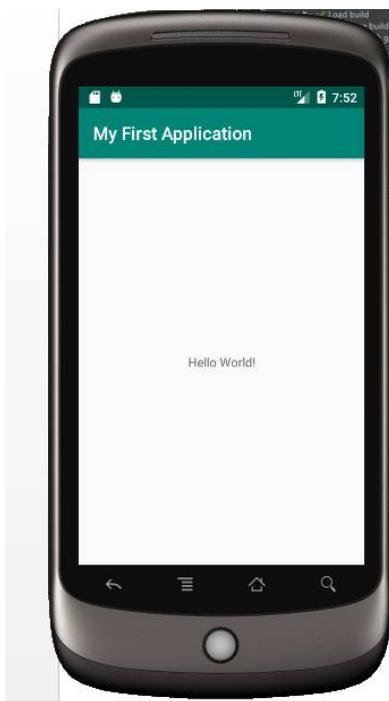


Рис.12. Запущенная программа в эмуляторе

#### 1.2.4.4. Структура проекта

На рисунке 13 слева приведена полная структура проекта (доступна при выборе пункта Project в верхней левой части экрана) и, справа - структура папки **MyFirstApplication\app\**, содержащей наиболее интересные для нас файлы.

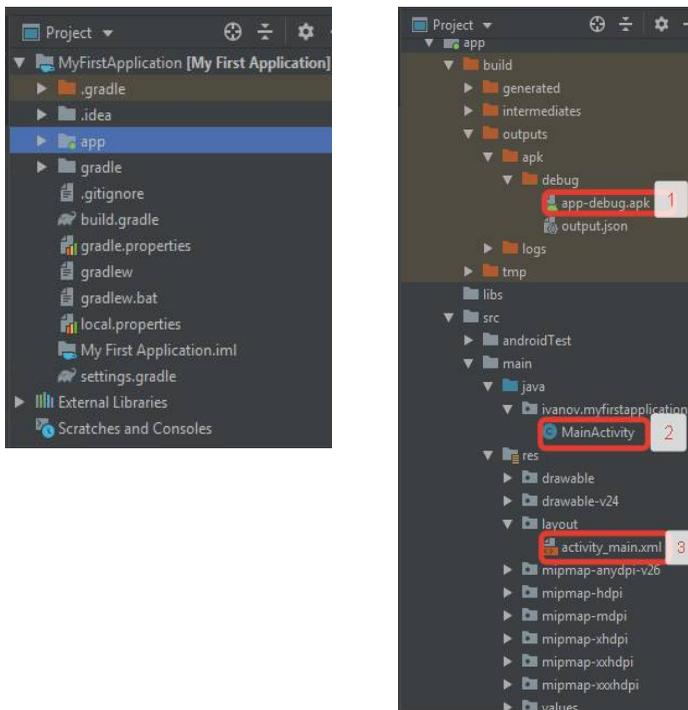


Рис. 13. Структура проекта

Папка **\app** является отдельным модулем для приложения и содержит все необходимые файлы приложения - код, ресурсы картинок и т.д., например:

1. Файл **\app\build\outputs\apk\debug\app-debug.apk** - установочный файл системы Android, который будет создан при выборе пункта меню «**Build - Build Bundle(s)/APK – Build APK (S)**».

2. Файл **\app\src\main\java\ivanov\myfirstapplication\MainActivity.java** (см. выше);

3. Файл **\app\src\main\res\layout\activity\_main.xml** (см. выше);

4. Также в каталоге `\app\src\main\` располагается файл-манифест приложения **AndroidManifest.xml**.

Остальные папки проекта служат для хранения различных настроек, управления проектом и т.д.

#### 1.2.4.5. Анализ файлов проекта

Рассмотрим содержимое файла **MainActivity.java**:

```
package com.nam.myapplication0;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

Рис. 14. Содержимое файла MainActivity.java

Как можно увидеть, имя пакета совпадает с именем приложения, определенным ранее. Далее производится подключение библиотек и определение класса **MainActivity** с переопределенным методом **onCreate**. В дальнейшем в этом файле будет храниться вся логика первой **Activity** нашего приложения. Эта **Activity** связана с файлом **activity\_main.xml**, содержащим описание формы (экрана), что прописано в последней строчке кода определения метода **onCreate**.

Содержимое файла **activity\_main.xml**:

```

<?xml version="1.0" encoding="utf-8" ?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>

```

Рис. 15. Содержимое файла разметки

Файл содержит описание внешнего представления **MainActivity** нашего приложения. Пока что здесь находится описание единственного графического элемента экрана нашего приложения - «TextView», отображающего надпись "Hello World!".

Манифест содержит описания конфигурации приложения и элемента **activity** класса **MainActivity**, определенного в файле **MainActivity.java**. Содержимое файла **AndroidManifest.xml**:

```

<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
package="com.nam.myapplication">

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

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

</manifest>

```

Рис. 16. Содержимое файла манифеста

#### 1.2.4.6. Внесение изменений в приложение

Изменим приложение таким образом, чтобы приветствие было обращено к имени, введенному в поле ввода. Для этого добавим на наш экран в режиме Design три элемента: TextView, PlainText и Button:

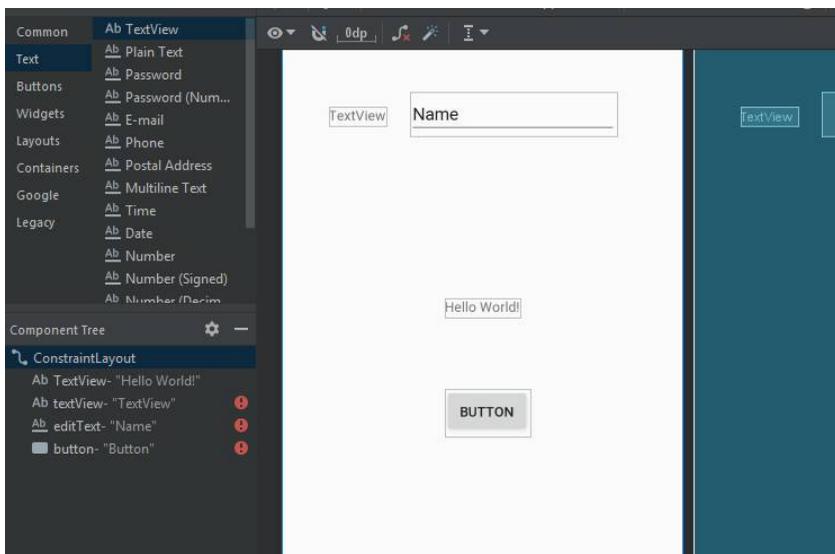


Рис. 17. Новые элементы управления

Отредактируем атрибуты «text» добавленных элементов так, чтобы экран принял вид, как на рис. 17. Попутно изменим атрибуты “id” всех элементов, придав им смысловую нагрузку. Так, для поля ввода имени это может быть «eTName», что обозначает editText Name, для кнопки приветствия – «btnHello», для текстовых полей – «tVName» и «tVHello». Такие имена удобно использовать при разработке логики приложения, поскольку они указывают как на тип элементов управления, так и на их назначение.

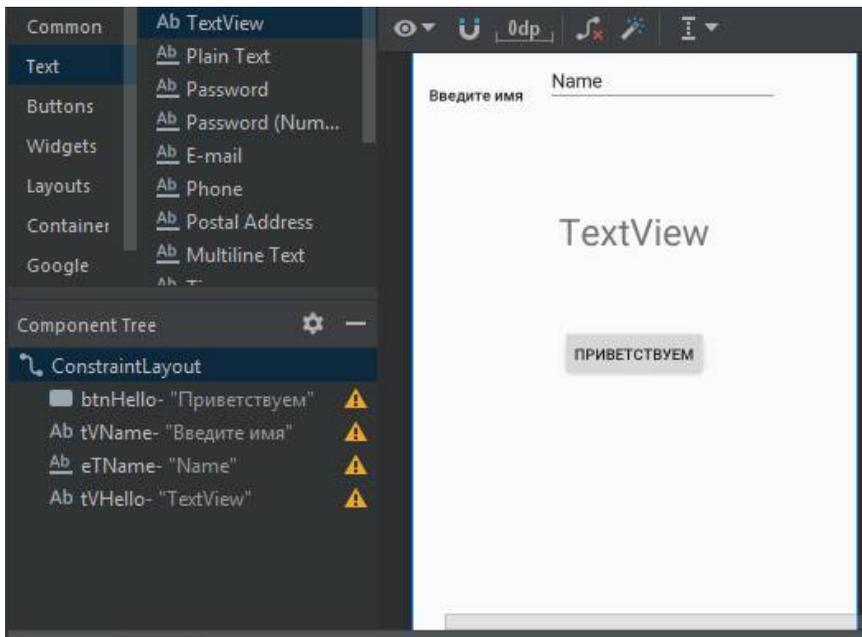


Рис. 18. Настроенные элементы управления

Запустим приложение в эмуляторе. Если результат выполнения напоминает рис. 19 слева, то необходимо нажать пиктограмму Infer Constraints , что позволит автоматически зафиксировать взаимное расположение элементов. После повторного запуска результат должен быть как в правой части рис. 19.

Если возникнет необходимость вновь изменить местоположение элементов, следует воспользоваться пиктограммой Clear all Constraints , а затем вновь зафиксировать их пиктограммой Infer Constraints.

Для добавления обработчика события нажатия кнопки «Приветствуем» выберите ее, и в поле атрибута onClick

напишите имя обработчика события нажатия на кнопку, например, «onClick».

Перейдем в режим редактирования «text» и, установив курсор на имени нашего обработчика событий, нажмем Alt+Enter. В появившемся списке необходимо выбрать «Create 'onClick(View)' in 'MainActivity'», что позволяет автоматически добавить метод onClick в файле **MainActivity.java** и перейти к его редактированию.

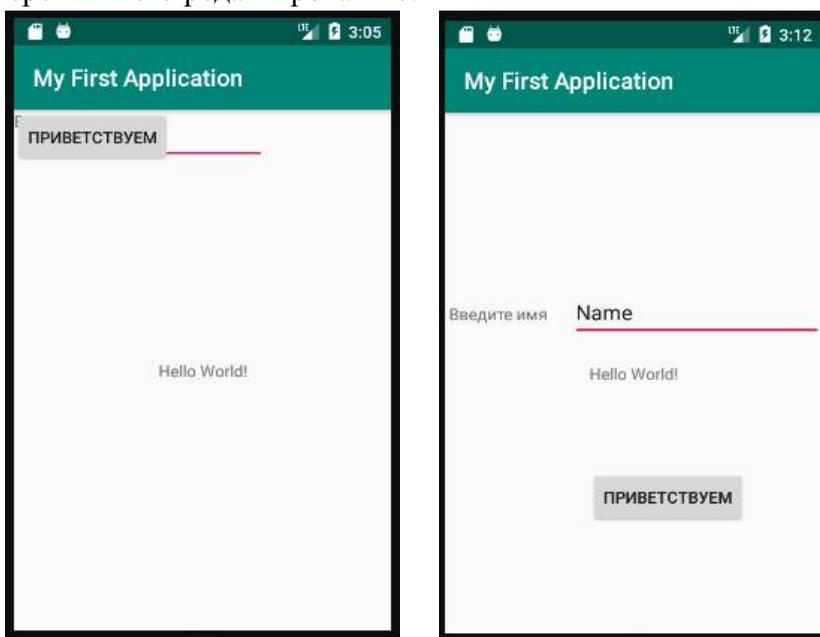


Рис. 19. Результат действия Infer Constrains

Для начала необходимо создать переменные класса для элементов управления, которые будут использоваться в коде (EditText для ввода имени и TextView для отображения приветствия). Для этого перед строкой **@Override** добавьте следующий код:

```
private TextView mHelloTextView;
```

```
private EditText mNameEditText;
```

В коде переопределения процедуры onCreate установим связь между этими параметрами и нашими визуальными элементами:

```
mHelloTextView=findViewById(R.id.tvHello); //получение TextView по ID  
mNameEditText =findViewById(R.id.eTName); // получение EditText по ID
```

Теперь через переменные можно управлять элементами из процедуры onClick:

```
public void onClick(View view) {  
  
    if (mNameEditText.getText().length() == 0) { //проверка, что имя введено  
        mHelloTextView.setText("Здравствуй, Мир!");  
    } else {  
        mHelloTextView.setText("Привет, " +  
mNameEditText.getText());  
    }  
}
```

Результат работы приложения приведен ниже.

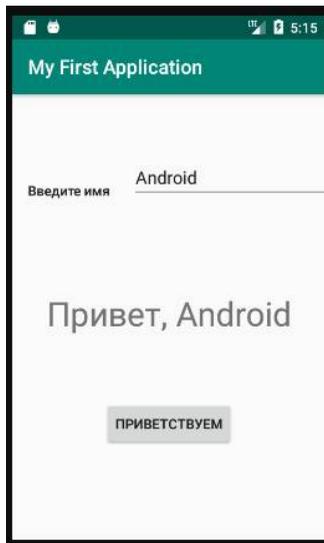


Рис. 20. Результат работы программы

### 1.2.4.7. Исправление недочетов

На рис. 21 можно увидеть, что Android Studio предупреждает нас при помощи желтых восклицательных знаков о некорректном использовании атрибута `text` ряда элементов управления. Если щелкнуть по этому значку, то отобразится следующее предупреждение:

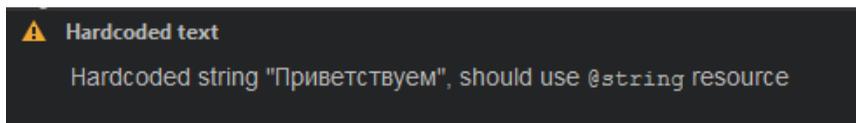


Рис. 21. Предупреждение о неправильном оформлении надписей

По правилам строки нужно хранить в строковых ресурсах. Подобный подход даёт разработчику множество преимуществ, в частности, быструю локализацию приложения.

Для создания строковых ресурсов следует в режиме Design щелкнуть по прямоугольной пиктограмме справа от корректируемого ресурса. В появившемся окне «Pick a Resource» нажмите кликните по надписи “Add new resource” и выберите «New string value». Ниже приведено окно (рис. 22) для определения текстового ресурса кнопки «Приветствуем».

Таким же образом определяются ресурсы цветов и прочие.

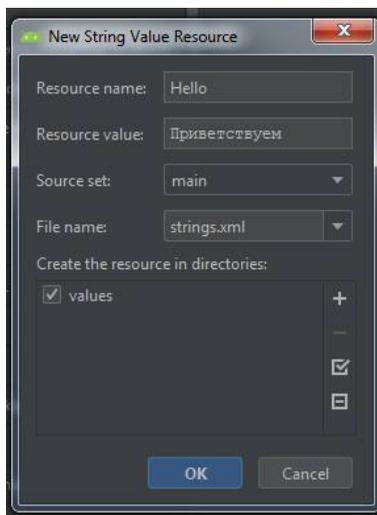


Рис. 22. Определение текстового ресурса

### 1.2.5. Практическое задание

1. Выполнить настройку отображения элементов управления с использованием инструментов центрирования.
2. Определить строковые ресурсы всех элементов управления.
3. Определить ресурс цвета для атрибута `background` базового контейнера `ConstraintLayout`.
4. На базе существующего приложения создать приложение-калькулятор, осуществляющий сложение и вычитание двух целых чисел. Для преобразования типов можно использовать конструкцию:

```
Integer N1=0;  
N1 = Integer.parseInt(mEditText.getText().toString());
```

## ТЕМА 2. ГРАФИЧЕСКИЙ ИНТЕРФЕЙС ANDROID

Цель изучения: формирование навыков проектирования и разработки графического интерфейса Android

### 2.1. Теоретическая часть

#### 2.1.1. Организация GUI Android

Базовым компонентом для создания визуального интерфейса в приложении Android является Activity.

Как отмечалось выше, в новом проекте после выбора шаблона создается главная Activity, которая описывается двумя файлами: MainActivity.java - описывает логику приложения и activity\_main.xml - определяет графический интерфейс активности. Между этими файлами устанавливается взаимное соответствие (см. рис. 23).

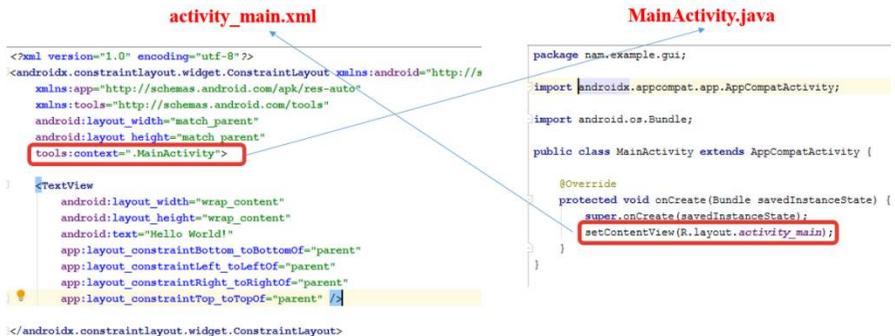


Рис. 23. Файлы описания activity

#### 2.1.2. Activity

Помимо главной Activity, проект может включать в себя набор дополнительных Activity, предназначенных для организации интерфейса слабо связанных между собой функций. Создание новой Activity в Android Studio выполняется через меню File-New-Activity-(Выбор шаблона),

или через контекстное меню New->Activity->Empty Activity, если выбрана папка, в которой находится класс MainActivity.

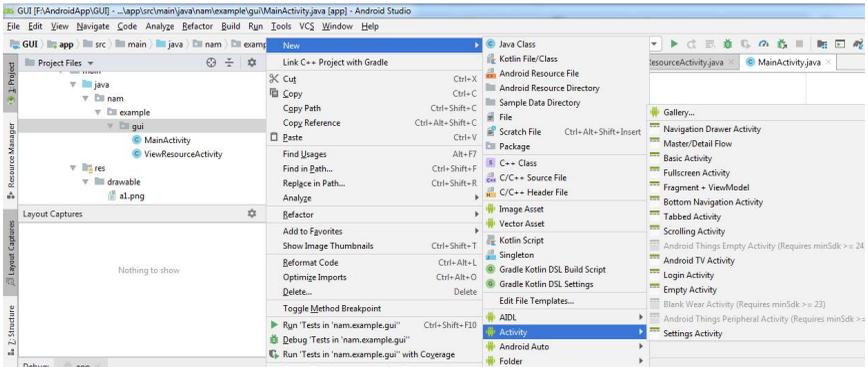


Рис. 24. Создание дополнительной Activity

Далее необходимо выполнить настройку Activity, которая заключается в определении ее имени и имени файла разметки. После этого информация об Activity автоматически добавится в файл манифеста.

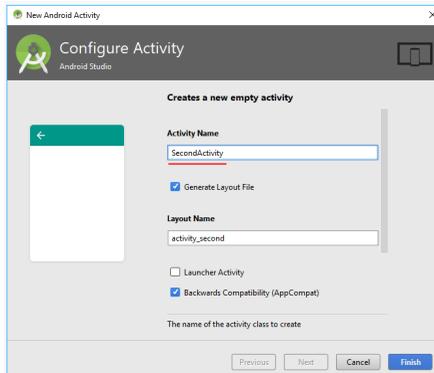


Рис. 25. Конфигурирование Activity

Для вызова одной активности из другой используется метод `startActivity()`:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}
public void onClick(View view) {
    Intent intent = new Intent(this, SecondActivity.class);
    startActivity(intent);
}
```

Рис. 26. Вызов дополнительной activity

Для передачи данных между двумя activity используется объект `Intent`. Через его метод `putExtra()` можно добавить ключ и связанное с ним значение [3].

Ниже приведен пример передачи из текущей activity в `SecondActivity` строки «Piter» с ключом «name».

#### Код MainActivity

```
// создание объекта Intent для запуска SecondActivity
Intent intent = new Intent(this, SecondActivity.class);
// передача объекта с ключом "name" и значением "Piter"
intent.putExtra("name", "Piter");
// запуск SecondActivity
startActivity(intent);
```

Рис. 27. Передача данных через Intent

В зависимости от типа отправляемых данных при их получении можно использовать ряд методов объекта `Bundle`. Все они в качестве параметра принимают ключ объекта. Основные из них:

- `get()`: универсальный метод, который возвращает значение типа `Object`. Соответственно, после получения данное значение необходимо преобразовать к нужному типу;
- `getString()`: возвращает объект типа `String`;
- `getInt()`: возвращает значение типа `int`;
- `getBytes()`: возвращает значение типа `byte`;

- `getChar()`: возвращает значение типа `char`;
- `getShort()`: возвращает значение типа `short`;
- `getLong()`: возвращает значение типа `long`;
- `getFloat()`: возвращает значение типа `float`;
- `getDouble()`: возвращает значение типа `double`;
- `getBoolean()`: возвращает значение типа `boolean`;
- `getCharArray()`: возвращает массив объектов `char`;
- `getIntArray()`: возвращает массив объектов `int`;
- `getFloatArray()`: возвращает массив объектов `float`;
- `getSerializable()`: возвращает объект интерфейса

`Serializable`.

Пример получения данных при загрузке `SecondActivity` с использованием метода `get()` приведен ниже.

#### Код `Secondactivity`

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    Bundle arguments = getIntent().getExtras();

    if(arguments!=null){
        String name = arguments.get("name").toString();
    }
}
```

Рис. 28. Прием данных с помощью метода `get()`

Каждая `Activity` может использовать несколько файлов разметки, что позволяет создавать различные варианты ее визуального отображения. Для добавления нового файла разметки необходимо выбрать: `File-New-Layout XML file`, и выполнить конфигурирование разметки.

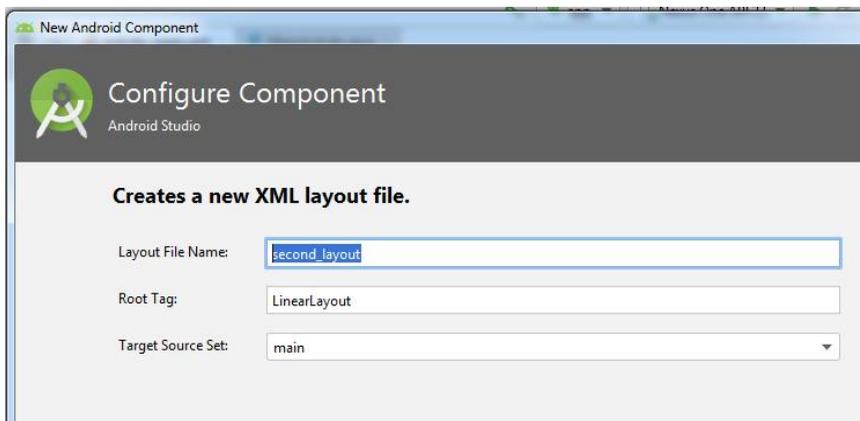


Рис. 29. Добавление разметки Activity

Для смены отображаемого файла разметки следует воспользоваться методом

```
setContentView(R.layout.activity_main);
```

Графический интерфейс пользователя (GUI- graphical user interface) в Android представляет собой иерархию объектов **android.view.ViewGroup** и **android.view.View**.

Каждый объект **ViewGroup** представляет контейнер, который содержит и упорядочивает дочерние объекты **View**. В частности, к контейнерам относят такие элементы, как **LinearLayout**, **GridLayout**, **ConstraintLayout** и ряд других [1]. Простые объекты **View** представляют собой элементы управления и прочие виджеты, например, кнопки, текстовые поля и т.д., через которые пользователь взаимодействует с программой.

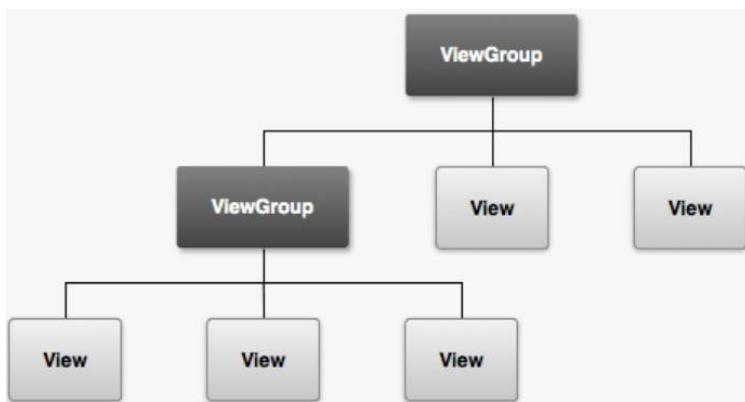


Рис. 30. Иерархическая структура графического интерфейса

Манипуляции с визуальными компонентами Activity удобно выполнять в режиме дизайнера, предоставляющим разнообразный инструментарий для этих целей.

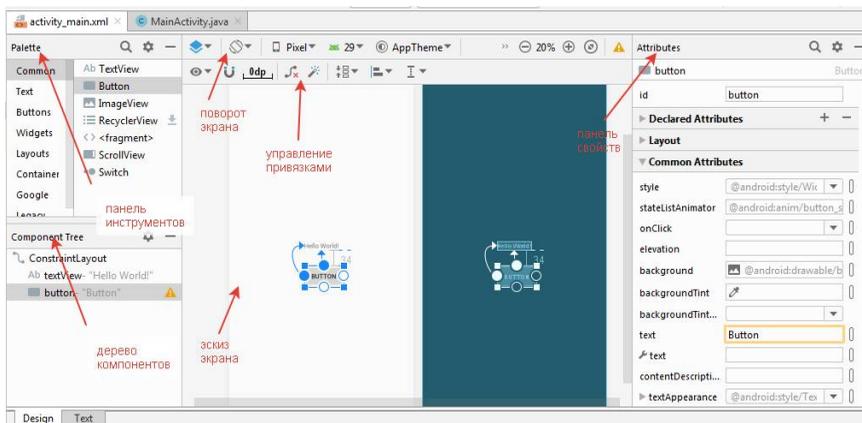


Рис. 31. Средства работы с элементами графического интерфейса в режиме Design

### 2.1.3. Макеты выравнивания

Первым делом при проектировании графического интерфейса следует определиться с типом родительского контейнера, в котором будут размещаться все элементы. Существуют различные типы контейнеров (**LinearLayout**, **RelativeLayout**, **FrameLayout**, **TableLayout**, **ConstraintLayout**), которые позволяют размещать визуальные элементы внутри себя различными способами (имеют различные макеты выравнивания) и доступны для выбора при конфигурировании разметки activity.

По умолчанию для новой activity в Android Studio устанавливается корневой контейнер (Root Tag) типа **ConstraintLayout**, в котором расположение элементов View определяется с помощью привязок (constraints), определяющих взаимное расположение всех элементов интерфейса.

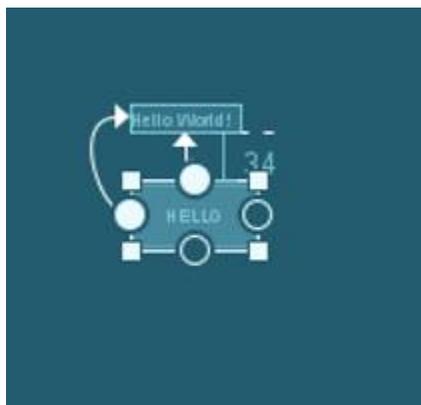


Рис. 32. Определение привязок в контейнере ConstraintLayout

Контейнер **LinearLayout** упорядочивает все дочерние элементы в одном направлении: по горизонтали или по

вертикали. Направление разметки указывается с помощью атрибута `android:orientation`.



Рис. 33. Выравнивание в контейнере `LinearLayout`

`LinearLayout` поддерживает такое свойство, как вес элемента, которое передается атрибутом `android:layout_weight`.

Это свойство принимает значение, указывающее, какую часть оставшегося свободного места контейнера по отношению к другим объектам займет данный элемент.

Например, если один элемент у нас будет иметь для свойства `android:layout_weight` значение 2, а другой - значение 1, то в сумме они дадут 3, поэтому первый элемент будет занимать  $2/3$  оставшегося пространства, а второй -  $1/3$ .

Если все элементы имеют значение `android:layout_weight="1"`, то все эти элементы будут равномерно распределены по всей площади контейнера.

Для создания табличных представлений используются контейнеры **`TableLayout`** и **`GridLayout`**.

```

<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:rowCount="3"
    android:columnCount="3">

    <Button android:text="1" />
    <Button android:text="2" />
    <Button android:text="3" />
    <Button android:text="4" />
    <Button android:text="5" />
    <Button android:text="6" />
    <Button android:text="7" />

    <Button android:text="8" />

    <Button android:text="9" />
</GridLayout>

```

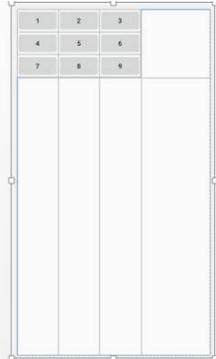


Рис. 34. Пример табличного представления элементов интерфейса с использованием контейнера GridLayout

Контейнер **ScrollView** предназначен для создания прокрутки в том случае, если все элементы одновременно не могут поместиться на экране устройства. **ScrollView** может вмещать только один элемент, поэтому если необходимо разместить несколько элементов, то их надо поместить в какой-нибудь контейнер, например, **LinearLayout**.

#### 2.1.4. Элементы графического интерфейса (элементы управления)

После того, как родительский контейнер выбран, производится добавление визуальных элементов (элементов управления) типа **View**, таких как **Button**, **TextView**, **EditText**, **Checkbox**, **ListView** и пр.

Рассмотрим некоторые из них.

##### 2.1.4.1. TextView

Наиболее простым визуальным элементом является **TextView**, который предназначен для отображения текста без возможности его редактирования.

## Основные атрибуты:

`android:text`: устанавливает текст элемента;

`android:textSize`: устанавливает высоту текста, в качестве единиц измерения для указания высоты используются `sp`;

`android:background`: задает фоновый цвет элемента в виде цвета в шестнадцатеричной записи или в виде цветового ресурса;

`android:textColor`: задает цвет текста;

`android:textAllCaps`: при значении `true` делает все символы в тексте заглавными;

`android:textDirection`: устанавливает направление текста.

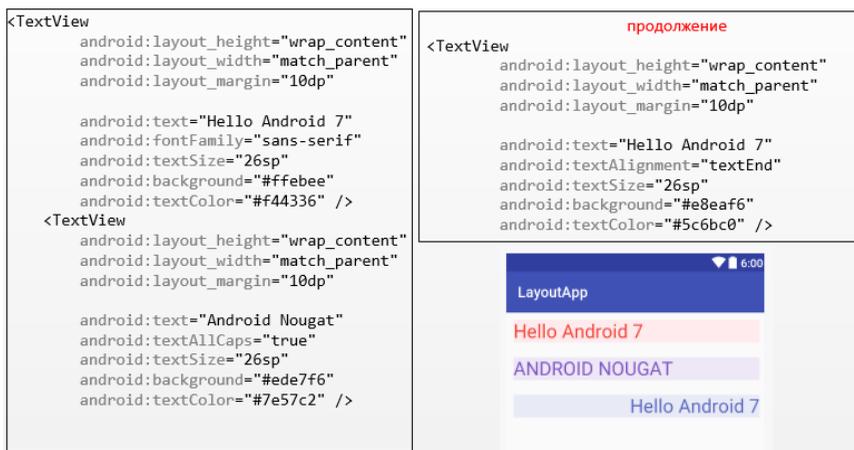


Рис. 35. Элемент `TextView`

### 2.1.4.2. `EditText`

`EditText` - подкласс класса `TextView`, представляет собой текстовое поле с возможностью ввода и редактирования текста. Имеет ряд дополнительных атрибутов:

`android:hint` - задает текст, отображаемый в качестве подсказки;

android:inputType - задает вид клавиатуры для ввода  
text: обычная клавиатура для ввода однострочного  
текста;

textMultiLine: многострочное текстовое поле

TextEmailAddress: обычная клавиатура, на которой  
присутствует символ @, ориентирована на ввод e-mail;

textUri: обычная клавиатура, на которой присутствует  
символ /, ориентирована на ввод интернет-адресов;

textPassword: клавиатура для ввода пароля;

textCapWords: при вводе первый введенный символ  
слова представляет заглавную букву, остальные – строчные;

number: числовая клавиатура;

phone: клавиатура в стиле обычного телефона;

date: клавиатура для ввода даты;

time: клавиатура для ввода времени;

datetime: клавиатура для ввода даты и времени.

EditText предоставляет возможность обрабатывать  
введенные символы по мере ввода пользователя. Ниже  
приведен пример, на котором с помощью метода  
addTextChangedListener() к элементу EditText добавляется  
слушатель ввода текста - объект TextWatcher, метод которого  
onTextChanged вызывается при изменении текста и  
отображает вводимый текст в TextView.

```

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        EditText editText = (EditText) findViewById(R.id.editText);
        editText.addTextChangedListener(new TextWatcher() {
            public void afterTextChanged(Editable s) {}
            public void beforeTextChanged(CharSequence s, int start,
                int count, int after) {}
            public void onTextChanged(CharSequence s, int start,
                int before, int count) {
                TextView textView = (TextView) findViewById(R.id.textView);
                textView.setText(s);
            }
        });
    }
}

```

Рис. 36. Пример обработки ввода в EditText

### 2.1.4.3. Button

Кнопки Button представлены классом android.widget.Button и являются наиболее распространенным элементом организации взаимодействия с пользователем.

Ключевые атрибуты кнопок:

text: задает текст на кнопке;

textColor: задает цвет текста на кнопке;

background: задает фоновый цвет кнопки;

textAllCaps: при значении true устанавливает текст в верхнем регистре. По умолчанию как раз и применяется значение true;

onClick: задает обработчик нажатия кнопки.

Описание процедуры создания обработчика событий для кнопки см. выше, в практической части темы 1.

### 2.1.4.4. Всплывающие окна. Toast

Toast представляет собой всплывающее окно для отображения текста в течение некоторого времени. Toast нельзя создать в xml, можно использовать только в коде java.

Ниже приведен пример создания всплывающего окна из обработчика нажатия кнопки.

```
public class MainActivity extends AppCompatActivity {

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

        setContentView(R.layout.activity_main);
    }

    public void onClick(View view){
        Toast toast = Toast.makeText(this, "Hello Android 7",Toast.LENGTH_LONG);
        toast.show();
    }
}
```

Рис. 37. Создание Toast по нажатию кнопки

#### 2.1.4.5. Checkbox

Checkbox - флажки, которые могут находиться в отмеченном и неотмеченном состоянии. Флажки позволяют производить множественный выбор из нескольких значений.

```
public void onCheckboxClicked(View view) {
    // Получаем флажок
    CheckBox language = (CheckBox) view;
    // Получаем, отмечен ли данный флажок
    boolean checked = language.isChecked();

    TextView selection = (TextView) findViewById(R.id.selection);
    // Смотрим, какой именно из флажков отмечен
    switch(view.getId()) {
        case R.id.java:
            if (checked){
                selection.setText("Java");
            }
            break;
        case R.id.javascript:
            if (checked)
                selection.setText("JavaScript");
            break;
    }
}
```

Рис. 38. Обработчик события Checkbox

Атрибут `android:onClick`, как и в случае с простыми кнопками, позволяет задать обработчик нажатия на флажок.

### 2.1.4.6. ToggleButton

`ToggleButton` – это кнопка, которая подобно элементу `CheckBox` может пребывать в двух состояниях: отмеченном и неотмеченном, причем для каждого состояния можно установить свой текст.

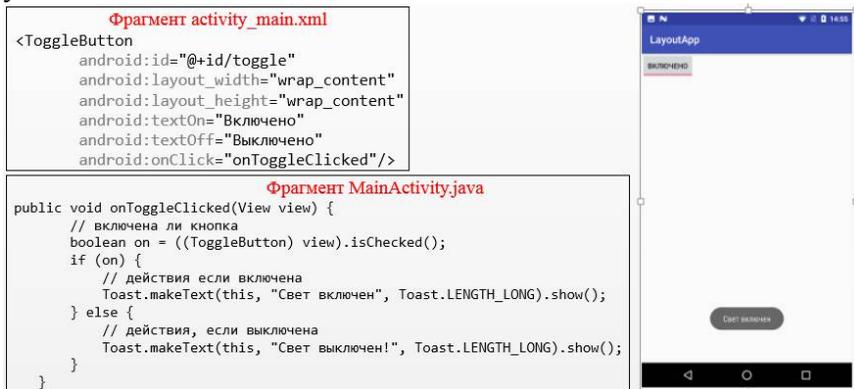


Рис. 39. Разметка, обработчик событий и внешний вид `ToggleButton`

### 2.1.4.7. Списочные элементы

Android представляет широкую палитру элементов, которые представляют списки. Все они являются наследниками класса `android.widget.AdapterView`. Это такие виджеты как **`ListView`**, **`GridView`**, **`Spinner`**. Они могут выступать контейнерами для других элементов управления.

При работе со списками приходится взаимодействовать с тремя компонентами. Во-первых, это сами элементы списков (**`ListView`**, **`GridView`**, **`Spinner`**), которые отображают данные. Во-вторых, это источник данных - массив, объект `ArrayList`, база данных и т.д., в котором находятся сами отображаемые данные. И в-третьих, это адаптеры -

специальные компоненты, которые связывают источник данных с элементом списка.

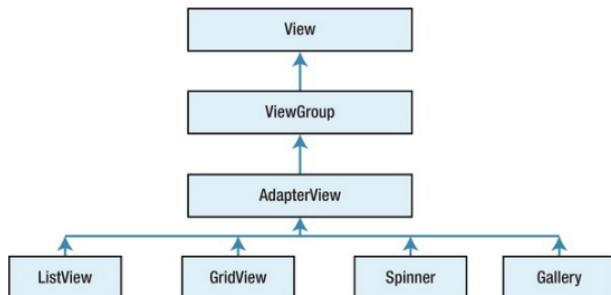


Рис. 40. Компоненты, используемые для организации работы списочных элементов

Рассмотрим связь элемента `ListView` с источником данных с помощью одного из таких адаптеров - класса **`ArrayAdapter`**.

Класс `ArrayAdapter` представляет собой простейший адаптер, который связывает массив данных с набором элементов `TextView`, из которых, к примеру, может состоять `ListView`.

Ниже приведено описание элемента `ListView` в файле разметки. Как можно увидеть, в описании не задается количество строк, а лишь прописывается способ определения размеров (высоты и ширины) элемента - (см. следующий раздел).

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ListView
        android:id="@+id/countriesList"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
    </ListView>

</RelativeLayout>

```

Рис. 41. Описание списочного элемента ListView в файле разметки

Код activity, связывающий ListView через ArrayAdapter с данными (массив), может выглядеть следующим образом:

```

public class MainActivity extends AppCompatActivity {

    // набор данных, которые свяжем со списком
    String[] countries = { "Бразилия", "Аргентина", "Колумбия", "Чили", "Уругвай"};
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // получаем элемент ListView
        ListView countriesList = (ListView) findViewById(R.id.countriesList);

        // создаем адаптер
        ArrayAdapter<String> adapter = new ArrayAdapter(this,
            android.R.layout.simple_list_item_1, countries);

        // устанавливаем для списка адаптер
        countriesList.setAdapter(adapter);
    }
}

```

Рис. 42. Привязка ListView и ArrayAdapter

В приведенном примере создается и заполняется массив стран (**countries**), затем в переопределении процедуры onCreate, по ID получаем ссылку на элемент ListView, создаем новый объект adapter, связываем его с массивом данных и устанавливаем его в качестве адаптера для списочного элемента. Эти действия позволяют отобразить на экране

списочный элемент, содержащий полный перечень элементов массива.

Для организации возможности выбора строки из списка необходимо установить слушатель `OnItemClickListener`, который предоставляет метод `onItemClick`, через параметры которого можно получить выделенный элемент и сопутствующие данные.

```
countriesList.setAdapter(adapter);
// добавляем для списка слушатель (продолжение)
countriesList.setOnItemClickListener(new.OnItemClickListener(){
    @Override
    public void onItemClick(AdapterView<?> parent, View v, int position, long id)
    {
        // выводим выбранный текст во всплывающем окне Toast
        Toast.makeText(getApplicationContext(), ((TextView) itemClicked).getText(),
        Toast.LENGTH_SHORT).show();
        //получаем строку с выбранным текстом
        TextView textView = (TextView) itemClicked;
        String strText = textView.getText().toString();
        ImageView imageView = findViewById(R.id.imageView);
    }
});
}
```

Рис. 43. Установка слушателя для списка и переопределение метода обработки события

### 2.1.5. Настройка элементов интерфейса

После добавления на экран `activity` визуального компонента, первым делом следует определить для него информативный `id`, т.к. `id` является основным атрибутом для доступа к компонентам при программировании логики работы программы.

Изменение атрибута в режиме дизайнера сразу синхронизируется с текстом `activity_main.xml`.

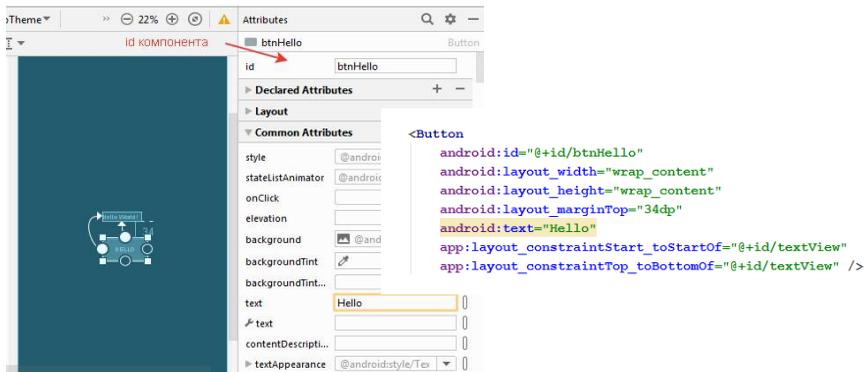


Рис. 44. Определение id элемента графического интерфейса

Следующим шагом в конфигурировании графических элементов является определение их размеров и взаимного расположения.

Все вышеперечисленные контейнеры (**LinearLayout**, **RelativeLayout**, **FrameLayout**, **TableLayout**, **ConstraintLayout**) для определения размеров элементов используют в файле xml атрибуты, которые начинаются с префикса **layout**.

К таким параметрам относятся атрибуты **layout\_height** и **layout\_width**, которые используются для установки размеров и могут принимать одно из следующих значений:

- точные размеры элемента, например, 96 dp;
- значение **wrap\_content**: элемент растягивается до тех границ, которые достаточны, чтобы вместить все его содержимое;
- значение **match\_parent**: элемент заполняет всю область родительского контейнера (тот же эффект будет при установке значения 0dp).

Для указания точных размеров элементов могут использоваться следующие единицы измерения:

- **px**: пиксели текущего экрана (не рекомендуется к использованию);

- **dp**: (device-independent pixels) - абстрактная единица измерения, основанная на физической плотности экрана с разрешением 160 dpi. Является предпочтительной для определения размеров элементов;

- **sp**: (scale-independent pixels) - независимые от масштабирования пиксели, рекомендуются для работы со шрифтами;

- и пр.

После определения размеров элементов следует определить отступы как от внешних границ элемента до границ контейнера, так и внутри самого элемента между его границами и содержимым.

Для установки внутренних отступов применяется атрибут **android:padding**. Он устанавливает отступы контента от всех четырех сторон контейнера. Можно устанавливать отступы только от одной стороны контейнера, применяя следующие атрибуты: **android:paddingLeft**, **android:paddingRight**, **android:paddingTop** и **android:paddingBottom**.

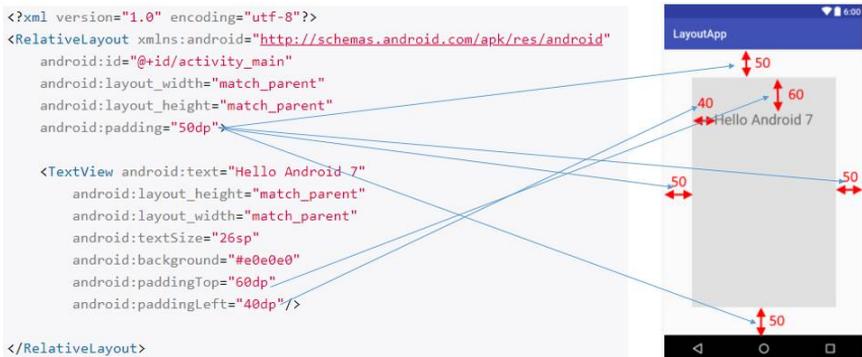


Рис. 45. Определение внутренних отступов

Для установки внешних отступов используются атрибуты **layout\_marginTop**, **layout\_marginBottom**, **layout\_marginLeft** и **layout\_marginRight**.



Рис. 46. Определение внешних отступов

Установка отступов, как и прочие виды управления положением визуальных элементов интерфейса, возможна не только средствами xml-разметки, но и программно.

Для программной установки внутренних отступов элемента вызывается метод **setPadding(left, top, right, bottom)**, в который передаются четыре значения, по одному для каждой из сторон.

Для установки внешних отступов необходимо реализовать объект **LayoutParams** для того контейнера, который применяется. И затем вызвать у этого объекта **LayoutParams** метод **setMargins(left, top, right, bottom)**.

```

RelativeLayout.LayoutParams layoutParams = new RelativeLayout.LayoutParams
    (ViewGroup.LayoutParams.MATCH_PARENT, 200);
// установка внешних отступов
layoutParams.setMargins(30,40,50,60);
// устанавливаем размеры
textView1.setLayoutParams(layoutParams);
// установка внутренних отступов
textView1.setPadding(30,30,30,30);
// добавляем TextView в RelativeLayout
relativeLayout.addView(textView1);
setContentView(relativeLayout);

```

Рис. 47. Программная установка отступов

Помимо вышеперечисленных атрибутов существуют атрибуты для позиционирования содержимого внутри объектов (**gravity**) и для позиционирования внутри контейнера (**layout\_gravity**). Оба этих атрибута могут принимать следующие значения: **top**, **bottom**, **left**, **right**, **center\_vertical**, **center\_horizontal**, **center**, **fill\_vertical**, **fill\_horizontal**, **fill**, **clip\_vertical**, **clip\_horizontal**, **start** и **end**, которые определяют различные варианты положения содержимого (текста).

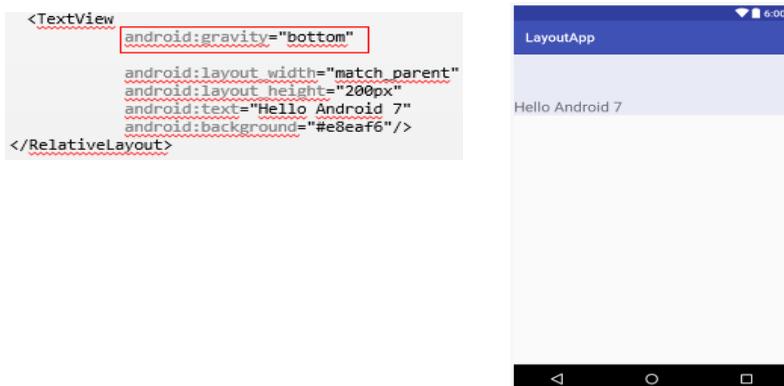


Рис. 48. Пример выравнивания содержимого TextView по нижней границе

## 2.1.6. Темы и стили

Как следует из вышеприведенного материала, настройка элементов интерфейса Android предполагает использование значительного количества параметров, позволяющих установить размер, цвет и тип шрифта, способы выравнивания и отступы и тому подобное. При этом, каждый из устанавливаемых параметров прописывается в виде отдельной строчки для каждого элемента интерфейса в файле разметки.

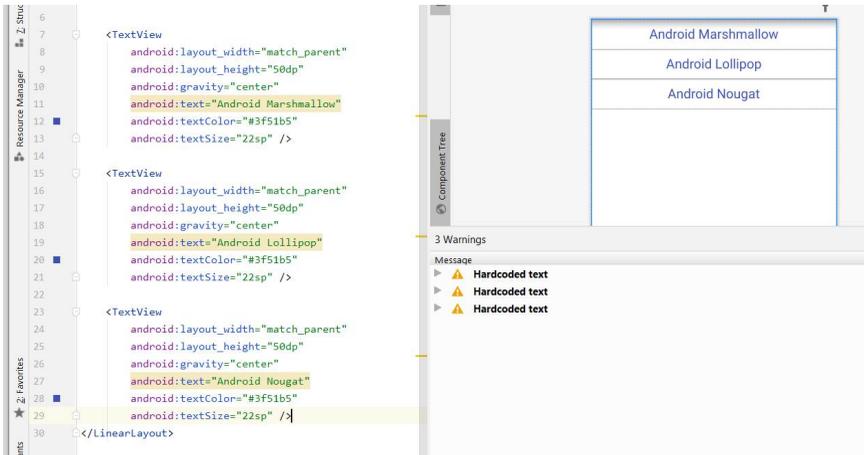


Рис. 49. Однотипные настройки группы элементов

Часто при этом группы однотипных элементов в одном или нескольких файлах разметки имеют совершенно одинаковые настройки, как на рисунке выше. Это приводит к тому, что количество строк файла разметки значительно возрастает за счет включения большого количества повторов одинаковых наборов строк. При этом, если возникнет необходимость изменить стиль приложения (например, размер или тип шрифта компонентов `TextView`), то править придется описание каждого элемента. Для решения этих проблем в Android используются стили. Стиль – единожды

созданное описание набора параметров элементов, которое может использоваться для любого количества элементов интерфейса, поддерживающих эти параметры.

Хранение стилей осуществляется в файле **styles.xml**, расположенном в папке **res/values** проекта. Если файл не существует, его можно создать, кликнув правой кнопкой мыши по папке **values** и выбрав в контекстном меню **New-XML-Values XML File**. Удобнее этот файл создать, воспользовавшись средствами автоматизации **Android Studio**. После того, как в файле разметки для элемента будет прописано определение не существующего стиля, **Android Studio** предложит создать файл **styles.xml** и определит в нем новый стиль.

Ниже приведены примеры описания стиля и его использования.



Рис. 50. Описание стиля и пример его использования

Развитием стилей в **Android** являются темы, представляющие собой укрупненные стили, которые могут применяться не к отдельным элементам, а к **Activity** или к приложению в целом.

Темы можно создавать самостоятельно или использовать стандартные.

При создании приложения автоматически создается его собственная тема, которая сохраняется в файле **themes.xml**. Она наследуется от одной из стандартных тем и может быть изменена следующим образом:

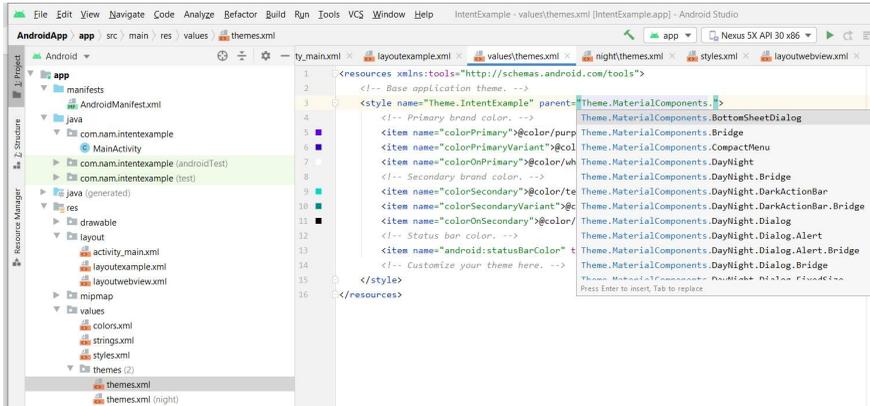


Рис. 51. Изменение родительской темы приложения

### 2.1.7. Ресурсы в Android и их использование в GUI

Ресурс в приложении Android представляет собой файл, например, файл разметки интерфейса, файл изображения, аудио/видео файл или некоторое значение, например, простую строку. То есть, ресурсы представляют собой и файлы разметки, и отдельные строки, и звуковые файлы, файлы изображений и т.д.

Все ресурсы находятся в проекте в каталоге **res**. Для различных типов ресурсов, определенных в проекте, в каталоге **res** создаются подкаталоги. Поддерживаемые подкаталоги:

**animator/**: xml-файлы, определяющие анимацию свойств;

**anim/**: xml-файлы, определяющие tween-анимацию;

**color/**: xml-файлы, определяющие список цветов;

**drawable/**: графические файлы (.png, .jpg, .gif);

**ipmap/:** графические файлы, используемые для иконок приложения под различные разрешения экранов;

**layout/:** xml-файлы, определяющие пользовательский интерфейс приложения;

**menu/:** xml-файлы, определяющие меню приложения

**raw/:** различные файлы, которые сохраняются в исходном виде;

**values/:** xml-файлы, которые содержат различные используемые в приложении значения, например, ресурсы строк;

**xml/:** Произвольные xml-файлы.

В стандартном проекте Android Studio каталог **/res** содержит каталоги не для всех типов ресурсов, которые использоваться в Android, однако при необходимости можно добавить в папку **res** нужный каталог, а в него затем поместить ресурс.

Когда происходит компиляция проекта, сведения обо всех ресурсах добавляются в специальный файл **R.java**, который можно найти в каталоге **Windows** проекта по пути **app\build\generated\source\r\debug\[пакет\_приложения]**.

Существует два способа определения доступа к ресурсам: в файле исходного кода и в файле **xml**.

При необходимости сослаться на ресурс в файле **xml**, например, в файле **activity\_main.xml**, используют следующую формализованную форму доступа:

**@[имя\_пакета:]тип\_ресурса/имя\_ресурса** , где

- **имя\_пакета** - представляет имя пакета, в котором ресурс находится (указывать необязательно, если ресурс находится в том же пакете);

- тип\_ресурса - подкласс, определенный в классе R для типа ресурса, например, **drawable**, **id**, **layout**, **string**, **attr**, **plural**, **array**;

- имя\_ресурса - имя файла ресурса без расширения или значение атрибута android:name в XML-элементе (для простых значений).

Например, мы хотим вывести в элемент TextView строку, содержащую имя приложения (string name="app\_name"), которая определена в виде ресурса в файле strings.xml, подойдет следующий код.

```
<TextView
    android:id="@+id/welcome"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/app_name" />
```

Рис. 52. Работа со строчным ресурсом из файла разметки.

Для решения подобной задачи с использованием кода Activity необходимо вначале использовать метод **getResources()**, который возвращает объект **android.content.res.Resources**, а затем вызвать один из его методов:

**getString()**: получает строку из файла strings.xml по числовому идентификатору;

**getDimension()**: получает числовое значение - ресурс dimen;

**getDrawable()**: получает графический файл;

**getBoolean()**: получает значение boolean;

```

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

    // получение ресурсов из файла values/strings.xml
    String app_name = getResources().getString(R.string.app_name);

    TextView textView = new TextView(this);
    textView.setText(app_name);

    setContentView(textView);
}

```

Рис. 53. Работа со строчным ресурсом из кода activity

### 2.1.7.1. Ресурсы изображений

Одним из наиболее распространенных источников ресурсов являются файлы изображений. Android поддерживает следующие форматы файлов:

- .png (предпочтителен);**
- .jpg (приемлем);**
- .gif (нежелателен).**

Для графических файлов в проекте по умолчанию создана папка **res/drawable**. При добавлении графических файлов в эту папку для каждого из них Android создает ресурс Drawable. После этого к ресурсу можно следующим образом:

в коде Java:

**R.drawable.имя\_файла**

в коде xml:

**@[имя\_пакета:]drawable/имя\_файла**

Стоит учитывать, что файл изображения будет добавляться в приложение, тем самым увеличивая его размер. При этом большие изображения отрицательно влияют на производительность. Поэтому следует использовать небольшие и оптимизированные (сжатые) графические файлы.

Для работы с изображениями в Android можно использовать различные элементы, но непосредственно для вывода изображений предназначен **ImageView**.

На рисунке ниже приведены описания этого элемента в файле разметки и в коде activity.

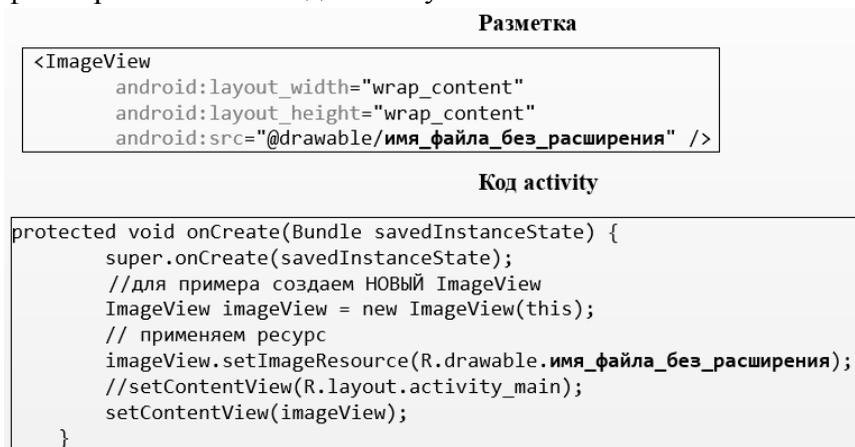


Рис. 54. Описание ImageView

## 2.2. Практическая часть

**Задание:** используя теоретический материал темы создать приложение для визуализации графических ресурсов в различных представлениях, обладающее следующими характеристиками:

1. Наличие трех или более файлов разметки для главной Activity. Разметки должны использовать различные контейнеры: `LinearLayout`, `TableLayout`. Обеспечить возможность переключения разметок.

2. Использование одного или нескольких списочных элементов (`ListView`, `GridView` или `Spinner` – на выбор).

3. Наличие одной или нескольких дополнительных Activity. Обеспечить возможность переключения Activity с передачей данных.

4. Использование в проекте набора файлов графических ресурсов.

**Функциональные требования:** приложение должно обеспечивать функции выбора, предпросмотра и полноэкранного просмотра файлов графических ресурсов.

Пример карты экранов приведен на рисунке ниже.

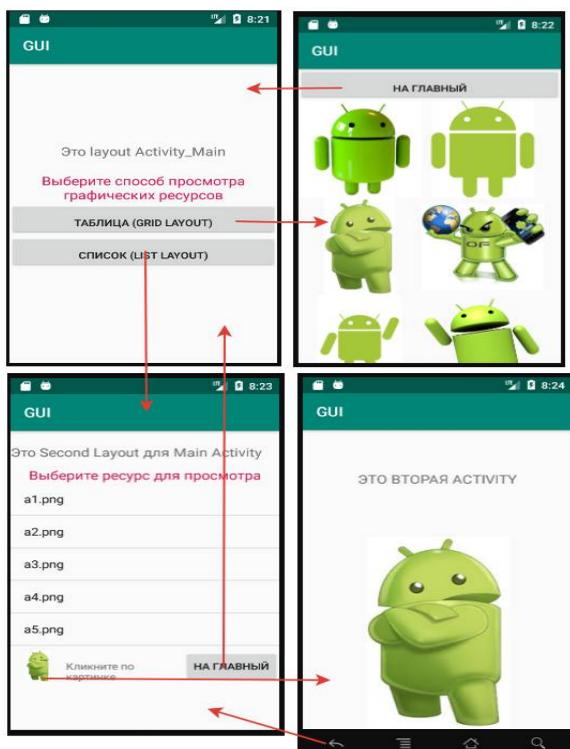


Рис. 55. Пример карты экранов

## Порядок выполнения работы

1. Запустить Android Studio. На кафедре ABC запуск производится от имени администратора. Выполняется преподавателем.
2. Создать новый проект. Использовать шаблон Empty Activity.
3. Добавить в приложение несколько файлов графических ресурсов. Лучше использовать файлы .png, имеющие небольшой размер. Для добавления скопируйте файлы в каталог `app\src\main\res\drawable` (см. рис.).

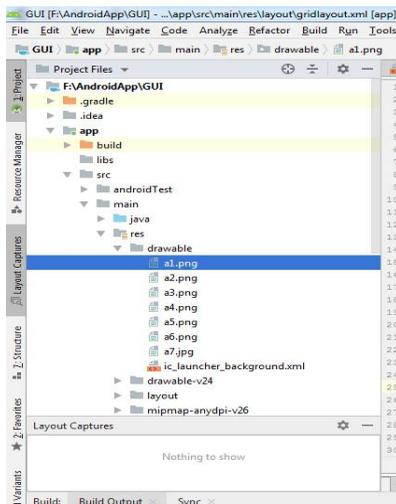


Рис. 56. Каталог графических ресурсов проекта

4. Добавьте необходимые элементы интерфейса и выполните настройку их местоположения, используя атрибуты `android:padding` и `layout_margin`. Для удобства манипулирования атрибутами местоположения рекомендуется добавить контейнер `<ScrollView>`, содержащий в себе контейнер `<LinearLayout>`. Проверьте, чтобы доступ к элементам интерфейса обеспечивался при любой ориентации экрана.

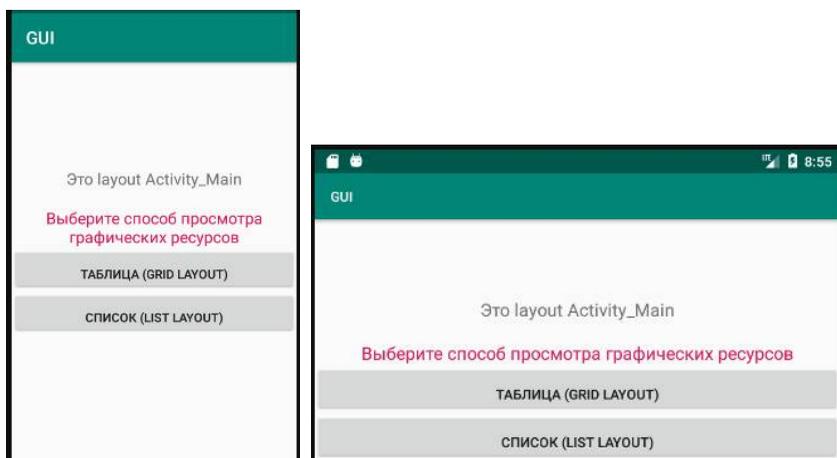


Рис. 57. Главный экран приложения

5. Добавьте дополнительные файлы разметки для главной активности. Для этого следует в каталоге проекта установить курсор на классе `MainActivity` и из контекстного меню `New-XML-Layout XML File` произвести добавление файла разметки (рис. 58).

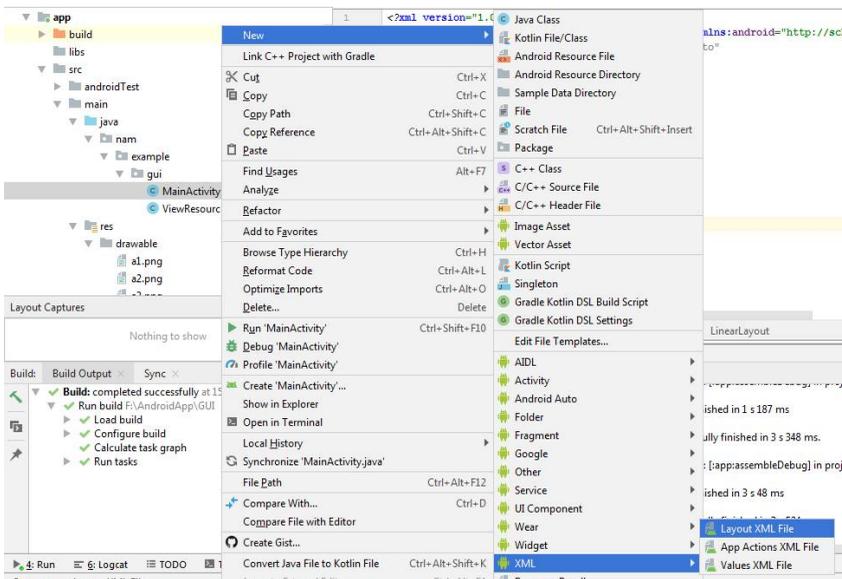


Рис. 58. Добавление файла разметки

В качестве корневого контейнера рекомендуется использовать `LinearLayout`.

6. Добавьте необходимые графические элементы и выполните настройку их местоположения. Вариант разметки и вид экрана просмотра списком приведены на рис. 59.

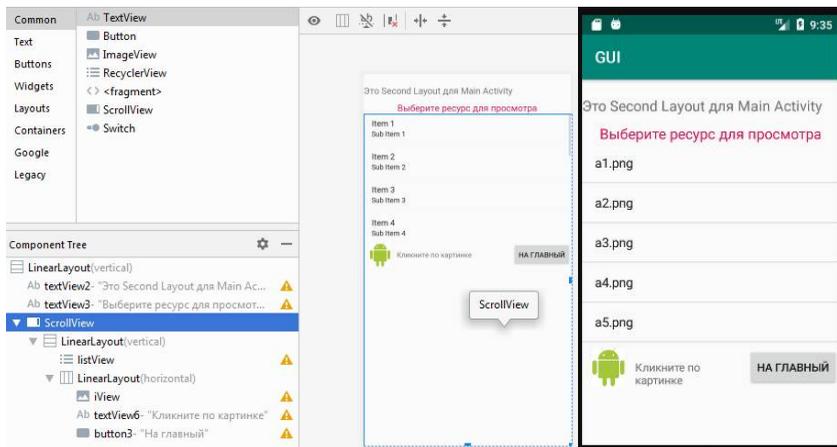


Рис. 59. Макет и вид экрана списочного просмотра ресурсов

Для визуализации файла разметки используйте метод `setContentview()`:

`setContentview(R.layout.имя_файла_разметки);`

Работа со списочным элементом `listView` предполагает выполнение в коде `java` следующих шагов:

1. Получение экземпляра элемента `Listview`.
  2. Формирование массива строк, содержащих имена графических файлов.
  3. Создание и использование адаптера данных, позволяющего связать строчный массив с элементом `listView`.
- Пример кода приведен на рис.60.

```

public void onClickSecond(View view) {
    setContentView(R.layout.second_layout);

    // 1. получаем экземпляр элемента ListView
    ListView listView = findViewById(R.id.listView);
    // 2. определяем строковый массив для имен графических файлов
    final String[] resNames = new String[] {
        "a1.png", "a2.png", "a3.png", "a4.png", "a5.png"
    };
    // 3. используем адаптер данных
    ArrayAdapter<String> adapter = new ArrayAdapter<>( context: this,
        android.R.layout.simple_list_item_1, resNames);
    listView.setAdapter(adapter);
}

```

Рис. 60. Пример кода для работы с ListView

Для обработки события выбора строки ListView необходимо создать «слушатель» `setOnItemClickListener` для элемента ListView и переопределить для него метод `onItemClick`, таким образом, чтобы он выполнял необходимые действия. Вариант переопределения метода для отображения графических ресурсов в элементе «imageView» приведен на рис. 61.

4. Для просмотра графических ресурсов в виде таблицы возможно использование разметок `GridLayout` или `TableLayout` (предпочтительно).

Вариант макета для табличного отображения ресурсов приведен на рис. 62.

5. Для создания дополнительной Activity необходимо воспользоваться меню `File-New-Activity`.

6. Для запуска Activity и передачи ей каких – либо параметров ключевым классом является `android.content.Intent`. Объект этого класса представляет собой задачу, которую надо выполнить приложению.

```

listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View itemClicked, int position,
        long id)
    {
        Toast toast = Toast.makeText(getApplicationContext(), ((TextView) itemClicked).getText(), Toast.LENGTH_SHORT);
        toast.setGravity( gravity: 2, xOffset: 10, yOffset: 400);
        toast.show();
        TextView textView = (TextView) itemClicked;
        strText = textView.getText().toString();
        ImageView imageView = findViewById(R.id.imageView);
        switch (strText) {
            case "a1.png":
                imageView.setImageResource(R.drawable.a1);
                break;
            case "a2.png":
                imageView.setImageResource(R.drawable.a2);
                break;
            case "a3.png":
                imageView.setImageResource(R.drawable.a3);
                break;
            case "a4.png":
                imageView.setImageResource(R.drawable.a4);
                break;
            case "a5.png":
                imageView.setImageResource(R.drawable.a5);
                break;
        }
    }
});

```

Рис. 61. Создание слушателя для listView

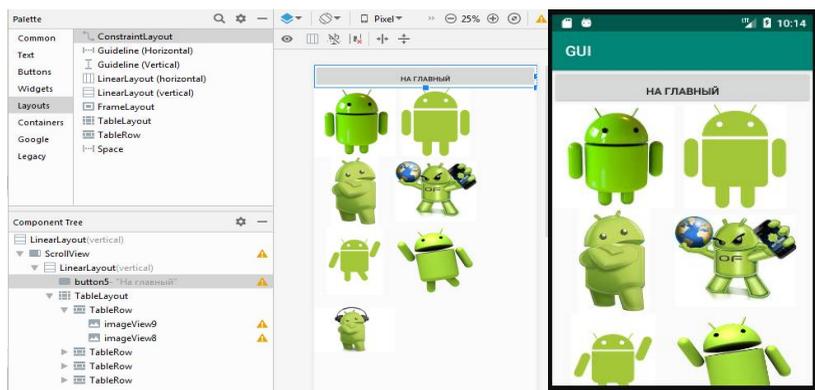


Рис. 62. Макет и вид экрана табличного просмотра ресурсов

На рис. 63 приведен пример обработчика события клика по изображению на экране списочного просмотра, который запускает новую активность **ViewResourceActivity** и передает ей в качестве параметра имя ресурса для просмотра (в

переменной `strText`). Имя параметра определено как «`fileName`».

```
public void onClImageView(View view) {  
    Intent intent = new Intent( packageContext: this, ViewResourceActivity.class);  
    intent.putExtra( name: "fileName", strText );  
    startActivity(intent);  
}
```

Рис. 63. Вызов Activity с передачей параметра

Для получения значения параметра в новой Activity используется объект `Bundle`, который предоставляет набор методов, принимающих в качестве параметра ключ передаваемого параметра.

Пример получения параметра с ключом «`filename`» приведен на рис. 64.

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_view_resource);  
    ImageView imageView1 = findViewById(R.id.imageView2);  
    Bundle arguments = getIntent().getExtras();  
    if(arguments!=null){  
        String fileName = arguments.get("fileName").toString();  
  
        switch (fileName) {  
            case "a1.png":  
                imageView1.setImageResource(R.drawable.a1);  
                break;  
            case "a2.png":  
                imageView1.setImageResource(R.drawable.a2);  
                break;  
            case "a3.png":  
                imageView1.setImageResource(R.drawable.a3);  
                break;  
            case "a4.png":  
                imageView1.setImageResource(R.drawable.a4);  
                break;  
            case "a5.png":  
                imageView1.setImageResource(R.drawable.a5);  
                break;  
        }  
    }  
}
```

Рис. 64. Получение параметра с ключом «`filename`»

## ТЕМА 3. ВЗАИМОДЕЙСТВИЕ С СИСТЕМНЫМИ ПРИЛОЖЕНИЯМИ

Цель изучения: формирование навыков использования объекта Intent.

### 3.1. Теоретическая часть

#### 3.1.1. Явные и неявные намерения

Как отмечалось выше, несмотря на то, что все приложения в Android выполняются в собственной «песочнице» и изолированы друг от друга, имеется механизм, позволяющий приложению использовать другие компоненты системы. Благодаря этому, Android-разработчик не станет писать собственное приложение – камеру, если его приложению необходимо сделать фотографию, или разрабатывать новые мессенджер и почтовый клиент, если из приложения необходимо отправить сообщение о том, что пришло электронное письмо. Вместо этого можно использовать системные приложения, обеспечивающие выполнение всех вышеперечисленных задач.

Для вызова системных приложений, как и для активации прочих компонентов используются объекты намерений «Intent».

Намерения - объекты Intent, могут быть «явными» или «неявными».

Явные намерения уже использовались выше, в практической работе к теме 2, при вызове дополнительной активности.

```
public void onClImageView(View view) {  
    Intent intent = new Intent( packageContext: this, ViewResourceActivity.class);  
    intent.putExtra( name: "fileName", strText );  
    startActivity(intent);  
}
```

Рис. 65. Создание «явного» намерения

Как можно увидеть, при создании «явного» намерения в качестве первого параметра передается объект «this», представляющий собой ссылку на текущую activity, используемую в качестве контекста (Context). Второй параметр явно указывает имя класса вызываемой активности (ViewResourceActivity.class). Поэтому данный Intent является «явным».

В случае использования «неявного» намерения, имя класса активности не указывается, но передается ряд параметров (**action**, **data**, **category**), значения которых указывают, что должна делать вызываемая активность. Так, если в качестве параметра action передать значение «ACTION\_VIEW», а в качестве data - адрес сайта, то будет запущен браузер.

```
public void onClActionView(View view) {  
    Uri address = Uri.parse("http://developer.alexanderklimov.ru");  
    Intent openLinkIntent = new Intent(Intent.ACTION_VIEW, address);  
  
    if (openLinkIntent.resolveActivity(getPackageManager()) != null) {  
        startActivity(openLinkIntent );  
    } else {  
        Log.d( tag: "Intent", msg: "Не получается обработать намерение!");  
    }  
}
```

Рис. 66. Использование неявного намерения для запуска браузера

Впрочем, для получения доступа в интернет можно воспользоваться простейшим элементом для рендеринга html-кода WebView, который базируется на движке WebKit, что гарантирует отображение контента подобно таким браузерам, как Google Chrome и Safari.

Для этого, первым делом следует указать в файле манифеста AndroidManifest.xml разрешение на использование Internet (перед тегом <application>), и определить элемент в файле разметки.

#### Разрешение в файле манифеста

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

#### Описание WebView в файле разметки

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

    <WebView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/webBrowser" />

</LinearLayout>
```

Рис. 67. Использование WebView

Чтобы загрузить определенную страницу в WebView, через метод loadUrl() необходимо передать ее адрес:

```
public void onActivity_Main(View view) {
    setContentView(R.layout.activity_main);
    WebView browser=(WebView) findViewById(R.id.webBrowser);
    browser.loadUrl("http://yandex.ru");
}
```

Рис. 68. Передача адреса страницы в WebView

В версиях Android, начиная с Android 8.0 и старше, для нормального функционирования вышеприведенного кода в файле манифеста, внутри тега <application> необходимо добавить строку:

```
android:usesCleartextTraffic="true"
```

### 3.1.2. Константы действия

Константы действия – это набор стандартных значений параметра `action`, определенных для объекта `Intent`, использование которых приводит к запуску часто употребляемых системных приложений.

Рассмотрим наиболее используемые константы действия:

- `ACTION_VIEW` — наиболее распространённое общее действие. Для данных, передаваемых с помощью пути `URI` в намерении, ищется наиболее подходящий способ вывода. Выбор приложения зависит от схемы (протокола) данных. Стандартные адреса `http:` будут открываться в браузере, адреса `tel:` — в приложении для дозвона, `geo:` — в программе `Google Maps`, а данные о контакте — отобразятся в приложении для управления контактной информацией;

- `ACTION_WEB_SEARCH` — открывает активность, которая ведет поиск в интернете, основываясь на тексте, переданном с помощью пути `URI` (как правило, при этом запускается браузер);

- `ACTION_DIAL` — инициализирует обращение по телефону;

- `ACTION_DELETE` — запускает активность, с помощью которой можно удалить данные, указанные в пути `URI` внутри намерения;

- `ACTION_IMAGE_CAPTURE` – запускает активность, для получения изображения при помощи камеры;

- `ACTION_VIDEO_CAPTURE` – запускает активность, для получения видео при помощи камеры;

- `ACTION_EDIT` — отображает данные для редактирования пользователем;

- `ACTION_PICK` - загружает дочернюю активность, позволяющую выбрать элемент из источника данных,

указанный с помощью пути URI. При закрытии должен возвращаться URI, ссылающийся на выбранный элемент. Активность, которая будет запущена, зависит от типа выбранных данных, например, при передаче пути `content://contacts/people` вызовется системный список контактов;

- `ACTION_SEARCH` — запускает активность для выполнения поиска. Поисковый запрос хранится в виде строки в дополнительном параметре намерения по ключу `SearchManager.QUERY`

- `ACTION_SENDTO` — открывает активность для отправки сообщений контакту, указанному в пути URI, который передаётся через намерение.

Рассмотрим примеры использования Intent с константами действия.

Для простейшего запуска камеры необходимо в `MainActivity.java` перед переопределением метода `onCreate` определить следующую константу:

```
private static final int CAMERA_REQUEST = 0;
```

Если обработка результатов работы камеры не планируется, то для вызова камеры достаточно определить следующий метод обработки нажатия кнопки:

```
public void onClCamera(View view) {  
    Intent cameraIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);  
    startActivityForResult(cameraIntent, CAMERA_REQUEST);  
}
```

Рис. 69. Вызов камеры

Чтобы получить возможность доступа к результатам съемки, например, разместить фотографию на элементе `ImageView`, следует переопределить метод `onActivityResult`, предназначенный для обработки результатов выполнения активности, следующим образом:

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == CAMERA_REQUEST && resultCode == RESULT_OK) {
        // Фотка сделана, извлекаем картинку
        Bitmap thumbnailBitmap = (Bitmap) data.getExtras().get("data");
        imageView=findViewById(R.id.imageView);
        imageView.setImageBitmap(thumbnailBitmap);
    }
}
```

Рис. 70. Переопределение метода для обработки результатов фотосъемки

Разрешение на использование камеры в манифесте прописывать не требуется.

Для вызова активности, осуществляющей телефонный звонок, добавим на экран элемент `editTextNumber` для ввода номера и используем следующий код:

```
public void onCall(View view) {
    number=findViewById(R.id.editTextNumber);
    String toDial="tel:"+number.getText().toString();
    startActivity(new Intent(Intent.ACTION_DIAL, Uri.parse(toDial)));
}
```

Рис. 71. Вызов активности для осуществления телефонного звонка

## 3.2. Практическая часть

**Задание:** используя теоретический материал темы 3 создать приложение для организации взаимодействия со

стандартными приложениями Android, обладающее следующими характеристиками.

1. Наличие одной или нескольких дополнительных разметок и/или Activity. Обеспечить возможность переключения между Activity с передачей данных.

2. Организация взаимодействия с приложением Камера с последующим отображением фото на элементе ImageView.

3. Поиск в списке контактов.

4. Отправка сообщения.

5. Осуществление телефонного вызова.

6. Отображение страницы в Internet с использованием элемента WebView.

7. Запуск браузера.

## ТЕМА 4. РАБОТА С ФАЙЛОВОЙ СИСТЕМОЙ. РАЗРЕШЕНИЯ

Цель изучения: освоение принципов взаимодействия приложения Android с внешним хранилищем

### 4.1. Теоретическая часть

#### 4.1.1. Файлы приложения

При установке приложения Android в системе создается папка `/Android/data/<название_пакета>/`, и, как правило, все данные приложения, хранятся в каталогах этой папки. Так, файлы, которые создаются и редактируются приложением, хранятся в подкаталоге `/files` родительской папки приложения и по умолчанию доступны только самому приложению.

При обращении к файловой системе в Android следует помнить, что здесь, как и во всех Unix-подобных системах используется слэш `/`, а не обратный слэш `\"`и, в отличие от Windows, названия файлов и каталогов являются регистрозависимыми.

Для работы с файлами Android в Java используется абстрактный класс **`android.content.Context`**, предоставляющий следующие методы:

- **`deleteFile(String name)`**: удаляет указанный файл;
- **`fileList()`**: получает имена всех файлов из подкаталога `/files` папки приложения;
- **`getCacheDir()`**: получает ссылку на подкаталог `cache` в каталоге приложения;
- **`getDir(String dirName, int mode)`**: получает ссылку на подкаталог в каталоге приложения, если такого подкаталога нет, то он создается;
- **`getExternalCacheDir()`**: получает ссылку на папку `/cache` внешней файловой системы устройства;

- **getExternalFilesDir()**: получает ссылку на каталог /files внешней файловой системы устройства;
- **getFilePath(String filename)**: возвращает абсолютный путь к файлу в файловой системе;
- **openFileInput(String filename)**: открывает файл для чтения;
- **openFileOutput (String name, int mode)**: открывает файл для записи.

Ниже приведен код метода обработки нажатия кнопки, выполняющий вывод в файл, имя которого указано в EditText. Содержимое для записи в файл считывается из второго EditText.

```
public void onClSaveText(View view) {
    FileOutputStream fos = null; //поток для вывода в файл
    try { //попытка
        String text = textFile.getText().toString(); //чтение содержимого из EditText
        String FILE_NAME = fileName.getText().toString(); //чтение имени из EditText
        fos = openFileOutput(FILE_NAME, MODE_PRIVATE); //сопоставление потока с файлом
        fos.write(text.getBytes()); //побайтный вывод содержимого в файл
        Toast.makeText( context: this, text: "Файл сохранен", Toast.LENGTH_SHORT).show();
    }
    catch(IOException ex) { //обработка исключения
        Toast.makeText( context: this, ex.getMessage(), Toast.LENGTH_SHORT).show();
    }
    finally{
        try{
            if(fos!=null)
                fos.close(); //закрытие файла
        }
        catch(IOException ex){
            Toast.makeText( context: this, ex.getMessage(), Toast.LENGTH_SHORT).show();
        }
    }
}
```

Рис. 72. Вывод текста в файл в папке приложения

На следующем рисунке приведен код обработчика события нажатия кнопки, осуществляющий чтение текста из файла в папке приложения и вывод его в элемент TextView.

```
public void onClReadText(View view) {
    FileInputStream fin = null; //поток для ввода из файла
    String FILE_NAME = fileName.getText().toString(); //чтение имени из EditText
    try {
        fin = openFileInput(FILE_NAME); //открытие файла в поток
        byte[] bytes = new byte[fin.available()]; //массив байт для ввода из файла
        fin.read(bytes); //чтение байтов из файла в массив
        String text = new String (bytes); //байты в строку
        tv.setText(text); //строка в TextView
    }
    catch(IOException ex) {
        Toast.makeText( context: this, ex.getMessage(), Toast.LENGTH_SHORT).show();
    }
    finally{
        try{
            if(fin!=null)
                fin.close(); //закрытие потока
        }
        catch(IOException ex){
            Toast.makeText( context: this, ex.getMessage(), Toast.LENGTH_SHORT).show();
        }
    }
}
```

Рис. 73. Чтение текста из файла в папке приложения

Как можно видеть, все манипуляции с файлами выполняются с использованием конструкции для обработки исключений (try{} catch{} finally).

#### 4.1.2. Работа с файлами во внешнем хранилище

При работе с файлами во внешнем хранилище вышеописанный механизм сохраняется. При этом ключевым отличием является получение и использование пути к внешнему хранилищу через метод **Environment.getExternalStorageDirectory()**.

Но, для того, чтобы использовать в приложении внешнее хранилище, приложение должно получить разрешение на работу с ним.

Для этого пропишем разрешения на чтение и/или запись в файле манифеста.



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.multimedia">
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="MultiMedia">
```

Рис. 74. Разрешения на чтение и запись во внешнем хранилище

Чтобы избежать ошибок, связанных с предоставлением разрешений, перед работой с внешним хранилищем следует произвести проверку разрешений и, при их отсутствии, запросить их у пользователя.

Для этого перед переопределением метода onCreate в MainActivity.java добавим описание переменной и константы:

```
private static final int REQUEST_PERMISSION_WRITE = 1001;
private boolean permissionGranted;
```

В конце этого же файла добавим описание следующего набора методов:

- isExternalStorageWritable() – проверка доступности внешнего хранилища для записи;
- isExternalStorageReadable() - проверка доступности внешнего хранилища для чтения;
- checkPermissions() – вызов проверки разрешений;
- onRequestPermissionsResult – переопределение метода обработки результата проверки наличия разрешений.

```

// проверяем, доступно ли внешнее хранилище для чтения и записи
public boolean isExternalStorageWritable(){
String state = Environment.getExternalStorageState();
return Environment.MEDIA_MOUNTED.equals(state);
}
// проверяем, доступно ли внешнее хранилище хотя бы только для чтения
public boolean isExternalStorageReadable(){
String state = Environment.getExternalStorageState();
return (Environment.MEDIA_MOUNTED.equals(state) ||
Environment.MEDIA_MOUNTED_READ_ONLY.equals(state));
}
private boolean checkPermissions(){

if(!isExternalStorageReadable() || !isExternalStorageWritable()){
Toast.makeText( context: this, text: "Внешнее хранилище не доступно", Toast.LENGTH_LONG).show();
return false;
}
int permissionCheck = ContextCompat.checkSelfPermission( context: this,
Manifest.permission.WRITE_EXTERNAL_STORAGE);
if(permissionCheck!= PackageManager.PERMISSION_GRANTED){
ActivityCompat.requestPermissions( activity: this,
new String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE},
REQUEST_PERMISSION_WRITE);
return false;
}
return true;
}
}

```

Рис. 75. Методы проверки доступности внешнего хранилища

```

@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions,
@NonNull int[] grantResults){
switch (requestCode){
case REQUEST_PERMISSION_WRITE:
if(grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED){
permissionGranted = true;
Toast.makeText( context: this, text: "Разрешения получены", Toast.LENGTH_LONG).show();
}
else{
Toast.makeText( context: this, text: "Необходимо дать разрешения", Toast.LENGTH_LONG).show();
}
break;
}
}
}

```

Рис. 76. Переопределение метода обработки результата проверки наличия разрешений

С использованием методов проверки разрешений, метод записи в файл внешнего хранилища будет изменен следующим образом:

```
public void onClSaveText(View view) {
    //проверим разрешения
    if(!permissionGranted){
        checkPermissions();
        return;
    }
    FileOutputStream fos = null; //поток для вывода в файл
    try { //попытка
        String text = textFile.getText().toString(); //чтение содержимого из EditText
        //String FILE_NAME = fileName.getText().toString(); //чтение имени из EditText
        //fos = openFileOutput(FILE_NAME, MODE_PRIVATE); //сопоставление потока с файлом
        fos = new FileOutputStream(getExternalPath());
        fos.write(text.getBytes()); //побайтный вывод содержимого в файл
        Toast.makeText( context: this, text: "Файл сохранен", Toast.LENGTH_SHORT).show();
    }
    catch(IOException ex) { //обработка исключения
        Toast.makeText( context: this, ex.getMessage(), Toast.LENGTH_SHORT).show();
    }
    finally{
        try{
            if(fos!=null)
                fos.close(); //закрываем файла
        }
        catch(IOException ex){
            Toast.makeText( context: this, ex.getMessage(), Toast.LENGTH_SHORT).show();
        }
    }
}
```

Рис. 77. Модифицированный метод для записи в текстовый файл внешнего хранилища

При этом, если разрешения приложением не были получены, при запуске метода на экран выводится запрос разрешений в следующем виде (для Android 11.0):

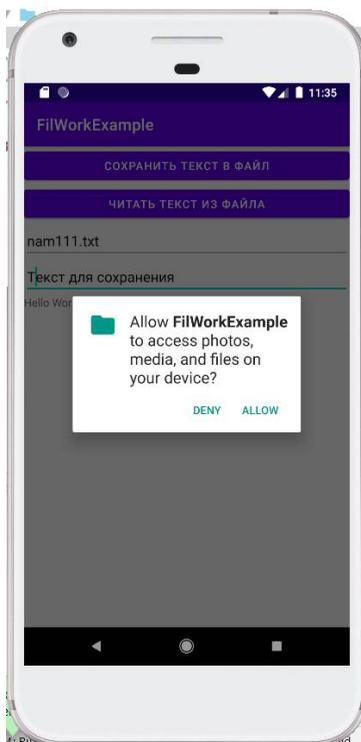


Рис. 78. Запрос пользовательских разрешений на использование внешнего хранилища

## 4.2. Практическая часть

Цель: формирование навыков разработки приложений взаимодействия с файловой системой и мультимедийными приложениями Android

Задание: используя материал темы создать приложение, обеспечивающее выбор файла во внешнем хранилище с

возможностью дальнейшей его обработки в зависимости от расширения:

- графический файл отобразить с использованием элемента. `ImageView` (см. п.1);

- аудиофайл воспроизвести с использованием элемента `MediaPlayer`;

- видеофайл воспроизвести с использованием элемента `VideoView`.

#### **4.2.1. Порядок выполнения**

1. Запустить `Android Studio` и эмулятор устройства.

2. Загрузить (если ранее не загружался) набор медиафайлов (изображения, аудио, видео) для тестирования. Загрузку можно выполнить перетаскиванием файлов на эмулятор или загрузкой с адреса: <https://yadi.sk/i/rZuHm1yGrayFoA>

3. Создать новый проект. Использовать шаблон `Empty Activity`.

4. Добавить необходимые элементы интерфейса для реализации всех функций, перечисленных в задании. Для реализации различных функций можно использовать как дополнительные разметки, так и дополнительные активности. Простейший вариант оформления интерфейса приложения приведен ниже.

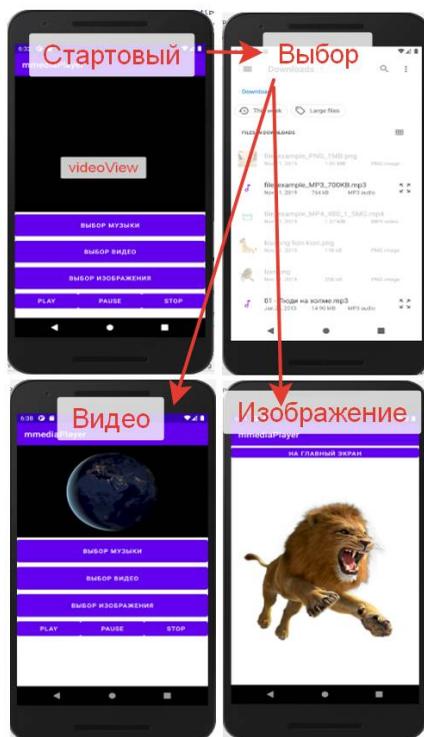


Рис. 79. Карта экранов приложения

Способы реализации UI не регламентируются. Так, функцию выбора файла с его последующей обработкой можно реализовать, например, при помощи набора кнопок, определяющих тип файла, или с использованием элементов RadioGroup/RadioButton.

5. Реализовать логику работы приложения в соответствии с нижеприведенным материалом.

#### 4.2.2. Разрешения

В связи с тем, что приложение будет работать с внешними источниками данных, первым делом необходимо в

манифесте установить разрешения на работу с внешним хранилищем и интернет:

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

Для проверки наличия разрешений и запроса их у пользователя, создадим и переопределим ряд методов:

```
//проверка и установка разрешений для работы с внешним хранилищем
// проверяем, доступно ли внешнее хранилище для чтения и записи
public boolean isExternalStorageWriteable(){
    String state = Environment.getExternalStorageState();
    return Environment.MEDIA_MOUNTED.equals(state);
}
// проверяем, доступно ли внешнее хранилище хотя бы только для чтения
public boolean isExternalStorageReadable(){
    String state = Environment.getExternalStorageState();
    return (Environment.MEDIA_MOUNTED.equals(state) ||
        Environment.MEDIA_MOUNTED_READ_ONLY.equals(state));
}

private boolean checkPermissions(){

    if(!isExternalStorageReadable() || !isExternalStorageWriteable()){
        Toast.makeText(this, "Внешнее хранилище не доступно",
            Toast.LENGTH_LONG).show();
        return false;
    }
    int permissionCheck = ContextCompat.checkSelfPermission(this,
        Manifest.permission.WRITE_EXTERNAL_STORAGE);
    if(permissionCheck!= PackageManager.PERMISSION_GRANTED){
        ActivityCompat.requestPermissions(this, new
            String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE},
            REQUEST_PERMISSION_WRITE);
        return false;
    }
    return true;
}

@Override
public void onRequestPermissionsResult(int requestCode, @NonNull
    String[] permissions, @NonNull int[] grantResults){
    switch (requestCode){
        case REQUEST_PERMISSION_WRITE:
            if(grantResults.length > 0 && grantResults[0] ==
                PackageManager.PERMISSION_GRANTED){
                permissionGranted = true;
                Toast.makeText(this, "Разрешения получены",
                    Toast.LENGTH_LONG).show();
            }
    }
}
```

```

    }
    else{
        Toast.makeText(this, "Необходимо дать разрешения",
Toast.LENGTH_LONG).show();
    }
    break;
}
}
}

```

Вышеперечисленные методы требуют определения булевой переменной `permissionGranted` и константы `REQUEST_PERMISSION_WRITE` :

```

//для установки разрешений
private static final int REQUEST_PERMISSION_WRITE = 1001;
private boolean permissionGranted;
//для элементов интерфейса
private MediaPlayer mPlayer;
private Button startButton, pauseButton, stopButton;
private VideoView videoView;
private String setType;
private ImageView imageView;
//для файлового менеджера
private static final int PICKFILE_RESULT_CODE = 1;

```

Эти описания необходимо расположить в файле `MainActivity.java` до переопределения метода `onCreate`. Здесь же описаны переменные для работы с элементами пользовательского интерфейса.

Вызов метода проверки и запроса разрешений удобно реализовать в переопределении метода `onCreate`:

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    //проверка и установка разрешений
    if(!permissionGranted){
        checkPermissions();
    }
    //получение ссылок на элементы интерфейса
    startButton = (Button) findViewById(R.id.start);
    pauseButton = (Button) findViewById(R.id.pause);
    stopButton = (Button) findViewById(R.id.stop);
    videoView =(VideoView) findViewById(R.id.videoView);
}
}

```

Такое решение приводит к появлению запроса на установку разрешений при первом запуске приложения, если они не были установлены ранее.

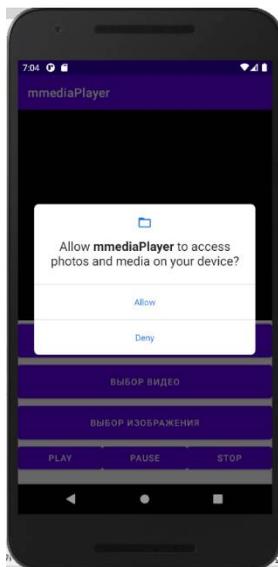


Рис. 80. Запрос на установку разрешений

### 4.2.3. Выбор файлов

Для выбора файлов всех типов с последующей их обработкой реализуем единый метод, реализующий вызов неявного намерения с константой действия **ACTION\_GET\_CONTENT**, что приводит к запуску стандартного проводника. Определение, какая кнопка была нажата, выполняется по ее id с вызовом метода `getId()`:

```
public void onClFile(View viewButton) {  
    //определим, какая кнопка нажата  
    if (viewButton.getId()==R.id.buttonAudio){  
        setType="audio/*";  
    }  
    if (viewButton.getId()==R.id.buttonVideo){  
        setType=setType="video/*";  
    }  
    if (viewButton.getId()==R.id.buttonImage){
```

```

        setType=setType="image/*";
    }
    Intent intent = new Intent(Intent.ACTION_GET_CONTENT);
    intent.setType(setType); //определяем тип
    startActivityForResult(intent, PICKFILE_RESULT_CODE);
}

```

Переменная **setType** типа String хранит описание типа полученного файла и будет использоваться далее в переопределении метода **onActivityResult** (обработчик результата работы неявного намерения):

```

@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if(requestCode == PICKFILE_RESULT_CODE && resultCode == RESULT_OK){
        //если выбран аудиофайл
        if ( setType=="audio/*"){ //если выбран аудиофайл
            mPlayer=MediaPlayer.create(this, data.getData());
            mPlayer.start();
            pauseButton.setEnabled(true);
            stopButton.setEnabled(true);
            mPlayer.setOnCompletionListener(new
MediaPlayer.OnCompletionListener() {
                @Override
                public void onCompletion(MediaPlayer mp) {
                    stopPlay();
                }
            });
        }
        if ( setType=="video/*") { //если выбран видеофайл
            videoView.setVideoURI(data.getData());//
            videoView.start();
        }
        if ( setType=="image/*") { //если выбрано изображение
            setContentView(R.layout.imaageview); //включение разметки с
imageView
            imageView =(ImageView) findViewById(R.id.imageView);
            imageView.setImageURI(data.getData());
        }
    }
}

```

Для воспроизведения музыки и других аудиоматериалов Android предоставляет класс **MediaPlayer**.

Чтобы воспроизводить аудио, **MediaPlayer** должен знать, какой именно ресурс (файл) нужно производить. Установить нужный ресурс для воспроизведения можно тремя способами:

- в метод `create()` объекта `MediaPlayer` передается `id` ресурса, представляющего аудиофайл;
- в метод `create()` объекта `MediaPlayer` передается объект `Uri`, представляющего аудиофайл;
- в метод `setDataSource()` объекта `MediaPlayer` передается полный путь к аудиофайлу.

В нашем случае для обработки всех файлов используется их `URI` - Унифицированный Идентификатор Ресурса, получаемый из результата выполнения намерения через метод **`data.getData()`**.

Для управления воспроизведением в классе `MediaPlayer` определены следующие методы:

- **`start()`**: запускает аудио;
- **`pause()`**: приостанавливает воспроизведение;
- **`stop()`**: полностью останавливает воспроизведение.

Эти методы реализованы в обработчиках события нажатия кнопок `Play`, `Pause` и `Stop`:

```
public void play(View view){ //обработчик кнопки PLAY
    mPlayer.start();
    startButton.setEnabled(false);
    pauseButton.setEnabled(true);
    stopButton.setEnabled(true);
}
public void pause(View view){ //обработчик кнопки PAUSE
    mPlayer.pause();
    startButton.setEnabled(true);
    pauseButton.setEnabled(false);
    stopButton.setEnabled(true);
}

public void stop(View view){ //обработчик кнопки STOP
    stopPlay();
}
private void stopPlay(){ //остановка воспроизведения аудио
    mPlayer.stop();
    pauseButton.setEnabled(false);
    stopButton.setEnabled(false);
    try {
        mPlayer.prepare();
        mPlayer.seekTo(0);
        startButton.setEnabled(true);
    }
    catch (Throwable t) {
        Toast.makeText(this, t.getMessage(), Toast.LENGTH_SHORT).show();
    }
}
```

## ЗАКЛЮЧЕНИЕ

Изучение теоретического материала, приведенного в пособии, рекомендуется совмещать с параллельным выполнением представленных практических заданий. Такой подход гарантирует приобретение базовых навыков по работе с наиболее популярной на сегодняшний день средой разработки для мобильных платформ – Android Studio.

Материал пособия не затрагивает общих вопросов, посвященных разработке на языке Java. Основной акцент сделан на освоении приемов, характерных именно для приложений для платформы Android, таких как создание графического интерфейса пользователя, работа с ресурсами и файловой системой, изучение жизненного цикла и методов класса Activity, взаимодействие с системными приложениями. При этом, для человека, обладающего базовыми знаниями в области программирования, приведенного материала будет достаточно, чтобы на концептуальном уровне освоить принципы построения мобильных приложений на языке Java для мобильной платформы Android.

## БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Общие сведения о платформе Android – Электрон. дан. – Режим доступа: <https://developer.android.com/guide/index.html>.
2. Отчёт о распространении на рынке мобильных устройств и их мировых продажах от компании Gartner – Электрон. дан. – Режим доступа: <https://www.gartner.com/en/newsroom/press-releases/2020-03-03-gartner-says-global-smartphone-sales-fell-slightly-in/>
3. METANIT.COM. Сайт о программировании – Электрон. дан. – Режим доступа: <https://metanit.com/java/android/2.2.php>

## ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ.....	3
ТЕМА 1. СОЗДАНИЕ ПЕРВОГО ПРИЛОЖЕНИЯ ANDROID.....	4
1.1. Теоретическая часть.....	4
1.1.1. Краткие сведения об Android.....	4
1.1.2. Компоненты Android-приложения.....	7
1.1.3. Запуск и взаимодействие компонентов.....	9
1.1.4. Жизненный цикл и методы операции (Activity)...	10
1.2. Практическая часть.....	
1.2.1. Установка средств разработки. Установка JDK...	12
1.2.2. Настройка среды окружения.....	13
1.2.3. Установка Android Studio.....	13
1.2.4. Создание первого приложения.....	
1.2.4.1. Выбор шаблона.....	16
1.2.4.2. Конфигурирование проекта.....	17
1.2.4.3. Работа с проектом.....	19
1.2.4.4. Структура проекта.....	21
1.2.4.5. Анализ файлов проекта.....	23
1.2.4.6. Внесение изменений в приложение.....	25
1.2.4.7. Исправление недочетов.....	30
1.2.5. Практическое задание.....	31
ТЕМА 2. ГРАФИЧЕСКИЙ ИНТЕРФЕЙС ANDROID....	32
2.1. Теоретическая часть.....	32
2.1.2. Activity.....	32
2.1.3. Макеты выравнивания.....	38
2.1.4. Элементы графического интерфейса (элементы управления).....	40
2.1.4.1. TextView.....	40
2.1.4.2. EditText.....	41
2.1.4.3. Button.....	43
2.1.4.4. Всплывающие окна. Toast.....	43
2.1.4.5. Checkbox.....	44
2.1.4.6. ToggleButton.....	45

2.1.4.7. Списочные элементы.....	45
2.1.5. Настройка элементов интерфейса.....	48
2.1.6. Темы и стили.....	53
2.1.7. Ресурсы в Android и их использование в GUI.....	55
2.2. Практическая часть.....	59
<b>ТЕМА 3. ВЗАИМОДЕЙСТВИЕ С СИСТЕМНЫМИ ПРИЛОЖЕНИЯМИ.....</b>	<b>68</b>
3.1. Теоретическая часть.....	68
3.1.1. Явные и неявные намерения.....	68
3.1.2. Константы действия.....	71
3.2. Практическая часть.....	73
<b>ТЕМА 4. РАБОТА С ФАЙЛОВОЙ СИСТЕМОЙ РАЗРЕШЕНИЯ.....</b>	<b>75</b>
4.1. Теоретическая часть.....	75
4.1.1. Файлы приложения.....	75
4.1.2. Работа с файлами во внешнем хранилище.....	77
4.2. Практическая часть.....	81
4.2.1. Порядок выполнения.....	82
4.2.2. Разрешения.....	83
4.2.3. Выбор файлов.....	86
<b>ЗАКЛЮЧЕНИЕ.....</b>	<b>89</b>
<b>БИБЛИОГРАФИЧЕСКИЙ СПИСОК.....</b>	<b>90</b>

**Учебное издание**

**Нужный Александр Михайлович  
Гребенникова Наталия Ивановна  
Сафронов Виталий Владимирович**

**РАЗРАБОТКА МОБИЛЬНЫХ ПРИЛОЖЕНИЙ  
НА ЯЗЫКЕ JAVA С ИСПОЛЬЗОВАНИЕМ ANDROID  
STUDIO**

Учебное пособие

В авторской редакции

Подписано к изданию 17.12.2020.

Объем данных 3,14 Мб.

ФГБОУ ВО «Воронежский государственный технический  
университет»  
394026 Воронеж, Московский просп., 14