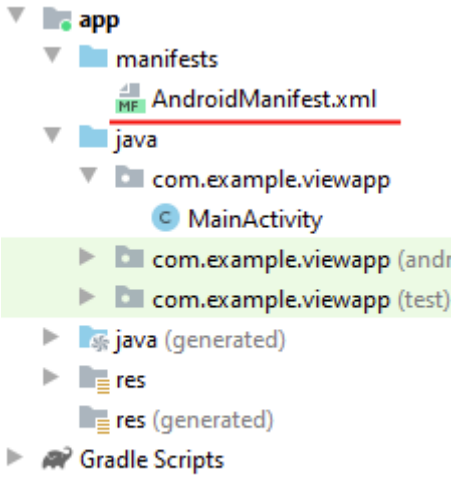
Практическая работа 7

Методические материалы

Файл манифеста AndroidManifest.xml

Каждое приложение содержит файл манифеста AndroidManifest.xml. Данный файл определяет важную информацию о приложении - название, версию, иконки, какие разрешения приложение использует, регистрирует все используемые классы activity, сервисы и т.д. Данный файл можно найти в проекте в папке **manifests**:



AndroidManifest.xml в Android Studio

Файл манифеста может выглядеть так:

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
```

```
        android:supportsRtl="true"
        android:theme="@style/Theme.ViewApp">
        <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>
```

Элементом корневого уровня является узел manifest. В данном случае только определяется пакет приложения - package="com.example.viewapp". Собственно, это определение файла манифеста по умолчанию. В каждом конкретном случае может отличаться пакет приложения, остальное содержимое при создании проекта с пустой activity будет аналогичным.

Большинство настроек уровня приложения определяются элементом application. Ряд настроек задаются с помощью атрибутов. По умолчанию применяются следующие атрибуты:

- **android:allowBackup** указывает, будет ли для приложения создаваться резервная копия. Значение android:allowBackup="true" разрешает создание резервной копии.

- **android:icon** устанавливает иконку приложения. При значении android:icon="@mipmap/ic_launcher" иконка приложения берется из каталога res/mipmap

- **android:roundIcon** устанавливает круглую иконку приложения. Также берется из каталога res/mipmap

- **android:label** задает название приложение, которое будет отображаться на мобильном устройстве в списке приложений и в заголовке. В данном случае оно хранится в строковых ресурсах - android:label="@string/app_name".

- **android:supportsRtl** указывает, могут ли использоваться различные RTL API - специальные API для работы с правосторонней ориентацией текста (например, для таких языков как арабский или фарси).

- **android:theme** устанавливает тему приложения. Тема определяет общий стиль приложения. Значение @style/Theme.ViewApp" берет тему "Theme.ViewApp" из каталога res/values/themes

Вложенные элементы activity определяют все используемые в приложении activity. В данном случае видно, что в приложении есть только одна activity - MainActivity.

```
<activity android:name=".MainActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

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

Элемент intent-filter в MainActivity указывает, как данная activity будет использоваться. В частности, с помощью узла action android:name="android.intent.action.MAIN", что данная activity будет входной точкой в приложение и не должна получать какие-либо данные извне.

Элемент category android:name="android.intent.category.LAUNCHER" указывает, что MainActivity будет представлять стартовый экран, который отображается при запуске приложения.

Файл манифеста может содержать множество элементов, которые имеют множество атрибутов. И все возможные элементы, и их атрибуты можно найти в документации. Здесь же рассмотрим некоторые примеры использования.

Определение версии

С помощью атрибутов элемента manifest можно определить версию приложения и его кода:

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

<!-- остальное содержимое-->

</manifest>
```

Атрибут android:versionName указывает на номер версии, который будет отображаться пользователю и на которую будут ориентироваться пользователи при работе с приложением.

Тогда как атрибут android:versionCode представляет номер версии для внутреннего использования. Этот номер только определяет, что одна версия приложения более новая, чем какая-то другая с меньшим номером номером версии. Этот номер не отображается пользователям.

При желании мы также можем определить версию в ресурсах, а здесь ссылаться на ресурс.

Установка версии SDK

Для управления версией android sdk в файле манифеста определяется элемент <uses-sdk>. Он может использовать следующие атрибуты:

- minSdkVersion: минимальная поддерживаемая версия SDK
- targetSdkVersion: оптимальная версия
- maxSdkVersion: максимальная версия

Версия определяется номером API, например, Jelly Beans 4.1 имеет версию 16, а Android 11 имеет версию 30:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.viewapp"
    android:versionName="1.0"
    android:versionCode="1">
    <uses-sdk android:minSdkVersion="22" android:targetSdkVersion="30" />
<!-- остальное содержимое-->
 </manifest>
```

Установка разрешений

Иногда приложению требуются разрешения на доступ к определенным ресурсам, например, к списку контактов, камере и т.д. Чтобы приложение могло работать с тем же списком контактов, в файле манифесте необходимо установить соответствующие разрешения. Для установки разрешений применяется элемент <uses-permission>:

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

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.viewapp">
    <uses-permission android:name="android.permission.READ_CONTACTS" />
    <uses-permission android:name="android.permission.CAMERA"
android:maxSdkVersion="30" />
<!-- остальное содержимое-->
 </manifest>
```

Атрибут android:name устанавливает название разрешения: в данном случае на чтение списка контактов и использование камеры. Опционально можно установить максимальную версию sdk посредством атрибута android:maxSdkVersion, который принимает номер API.

Поддержка разных разрешений

Мир устройств Android очень сильно фрагментирован, здесь встречаются как гаджеты с небольшим экраном, так и большие широкоэкранные телевизоры. И бывают случаи, когда надо ограничить использование приложения для определенных разрешений экранов. Для этого в файле манифеста определяется элемент <supports-screens>:

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

    <supports-screens
        android:largeScreens="true"
        android:normalScreens="true"
        android:smallScreens="false"
        android:xlargeScreens="true" />
<!-- остальное содержимое-->
 </manifest>
```

Данный элемент принимает четыре атрибута:

- android:largeScreens - экраны с диагональю от 4.5 до 10"
- android:normalScreens - экраны с диагональю от 3 до 4.5"
- android:smallScreens - экраны с диагональю меньше 3"
- android:xlargeScreens - экраны с диагональю больше 10"

Если атрибут имеет значение true, то приложение будет поддерживаться соответствующим размером экрана

Запрет на изменение ориентации

Приложение в зависимости от положения гаджета может находиться в альбомной и портретной ориентации. Не всегда это бывает удобно. Мы можем сделать, чтобы приложение вне зависимости от поворота гаджета использовало только одну ориентацию. Для этого в файле манифеста у требуемой activity надо установить атрибут **android:screenOrientation**. Например, запретим альбомную ориентацию:

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
```

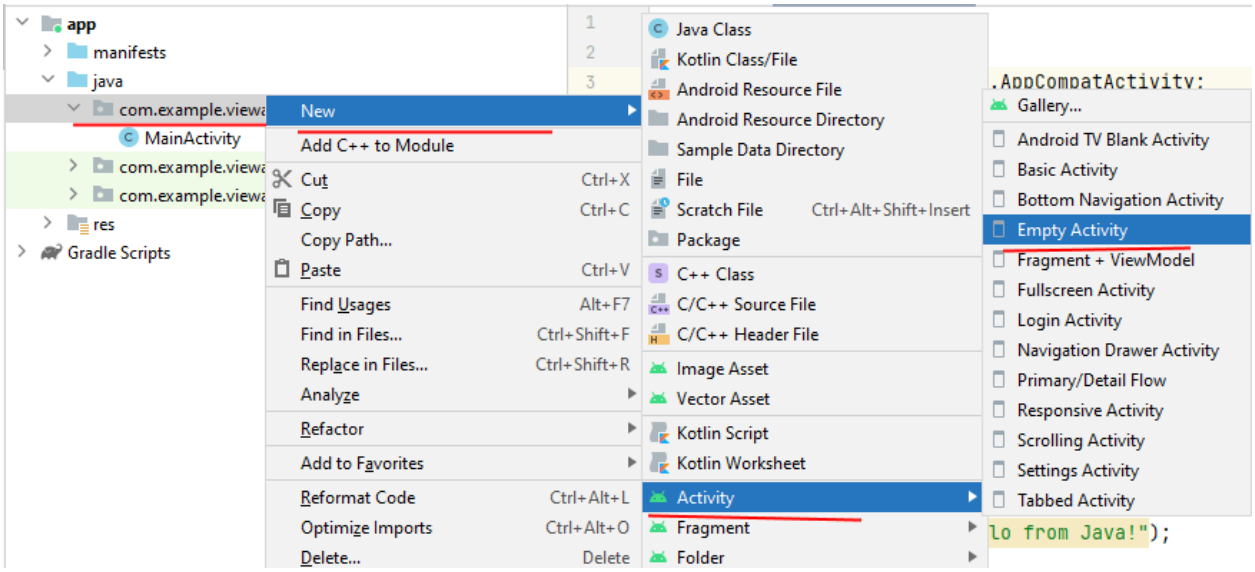
```
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.ViewApp">
        <activity android:name=".MainActivity"

            android:screenOrientation="portrait">

            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

Значение android:screenOrientation="portrait" указывает, что данная activity будет находиться только в портретной ориентации. Если же надо установить только альбомную ориентацию, тогда надо использовать значение android:screenOrientation="landscape"

Введение в Intent. Запуск Activity

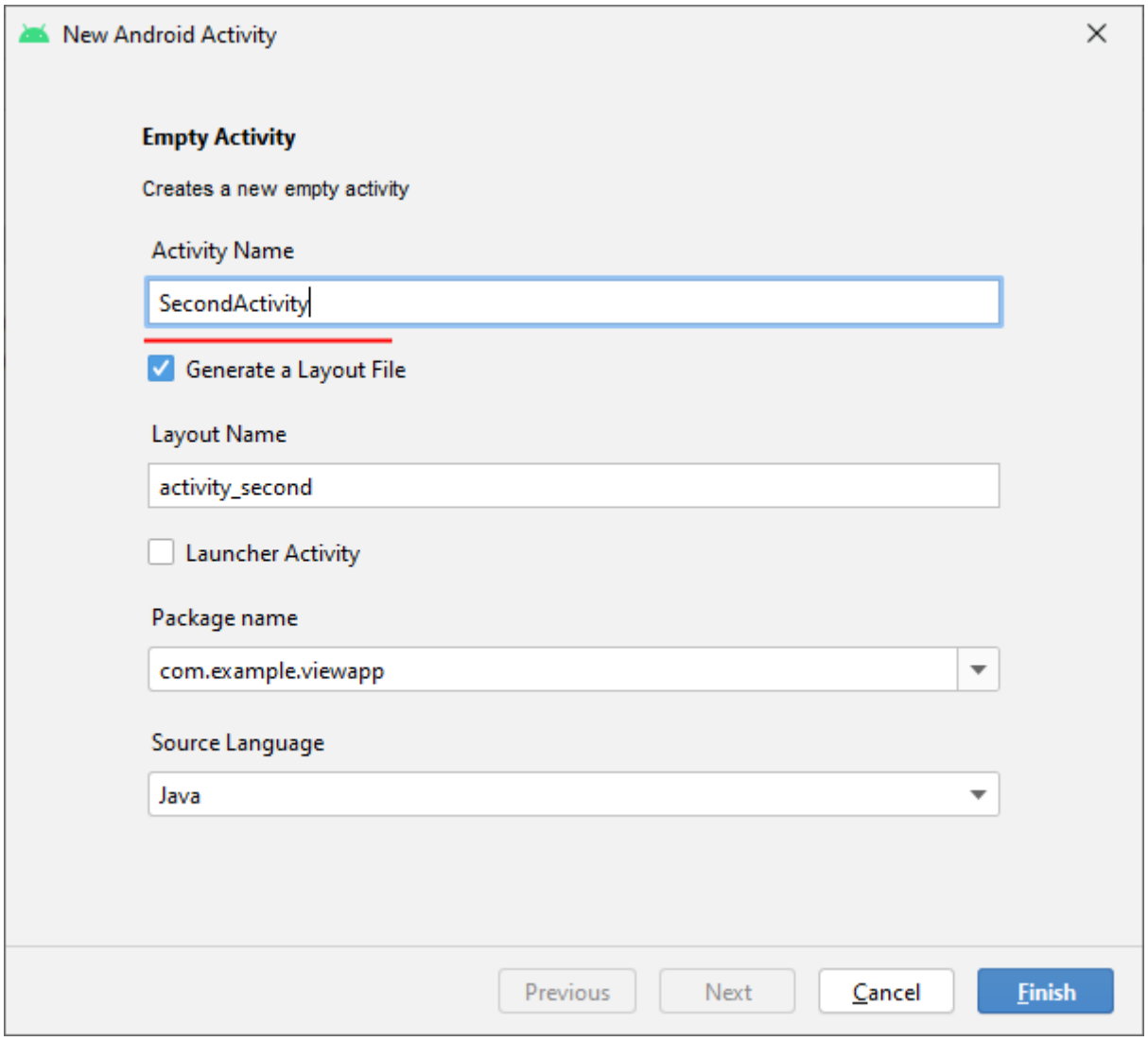
Для взаимодействия между различными объектами activity ключевым классом является **android.content.Intent**. Он представляет собой задачу, которую надо выполнить приложению.

Для работы с Intent добавим новый класс Activity. Для этого нажмем правой кнопкой мыши в папку, в которой находится класс MainActivity, и затем в контекстном меню выберем **New->Activity->Empty Activity**:



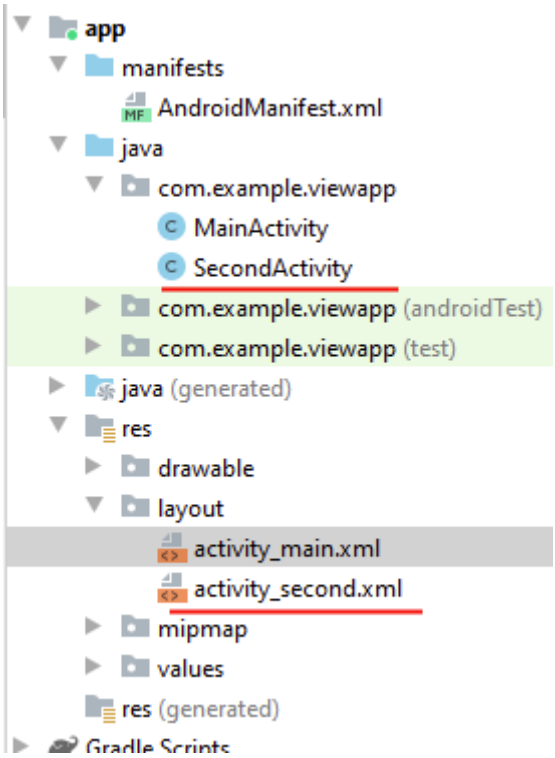
Добавление Empty Activity в Android Studio

Новый класс Activity назовем SecondActivity, а все остальные настройки оставим по умолчанию:



Установка Empty Activity в Android Studio

И после этого в проект будет добавлена новая Activity - SecondActivity:



MainActivity в Android Studio

После этого в файле манифеста AndroidManifest.xml мы сможем найти следующие строки:

```
<activity android:name=".SecondActivity"></activity>
```

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

Все используемые классы activity должны быть описаны в файле AndroidManifest.xml с помощью элемента <activity>. Каждый подобный

элемент содержит как минимум один атрибут android:name, который устанавливает имя класса activity.

Однако, по сути, activity - это стандартные классы java, которые наследуются от класса **Activity** или его наследников. Поэтому вместо встроенных шаблонов в Android Studio можем добавлять обычные классы, и затем их наследовать от класса Activity. Однако в этом случае нужно будет вручную добавлять в файл манифеста данные об activity.

Причем для MainActivity в элементе intent-filter определяется интент-фильтр. В нем элемент action значение "android.intent.action.MAIN" представляет главную точку входа в приложение. То есть MainActivity остается основной и запускается приложением по умолчанию.

Для SecondActivity просто указано, что она в проекте, и никаких intent-фильтров для нее не задано.

Чтобы из MainActivity запустить SecondActivity, надо вызвать метод startActivity():

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

В качестве параметра в метод startActivity передается объект Intent. Для своего создания Intent в конструкторе принимает два параметра: контекст выполнения (в данном случае это текущий объект MainActivity) и класс, который используется объектом Intent и представляет передаваемые в задачу данные (фактически класс activity, которую мы будем запускать).

Теперь рассмотрим реализацию перехода от одной Activity к другой. Для этого в файле activity_main.xml (то есть в интерфейсе для MainActivity) определим кнопку:

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

    <Button
        android:id="@+id/navButton"
        android:textSize="20sp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Перейти к SecondActivity"
        android:onClick="onClick"

        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

И определим для кнопки в классе **MainActivity** обработчик нажатия, по которому будет производиться переход к новой Activity:

```
package com.example.viewapp;
import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends AppCompatActivity {

    @Override
```

```
    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);
    }
}
```

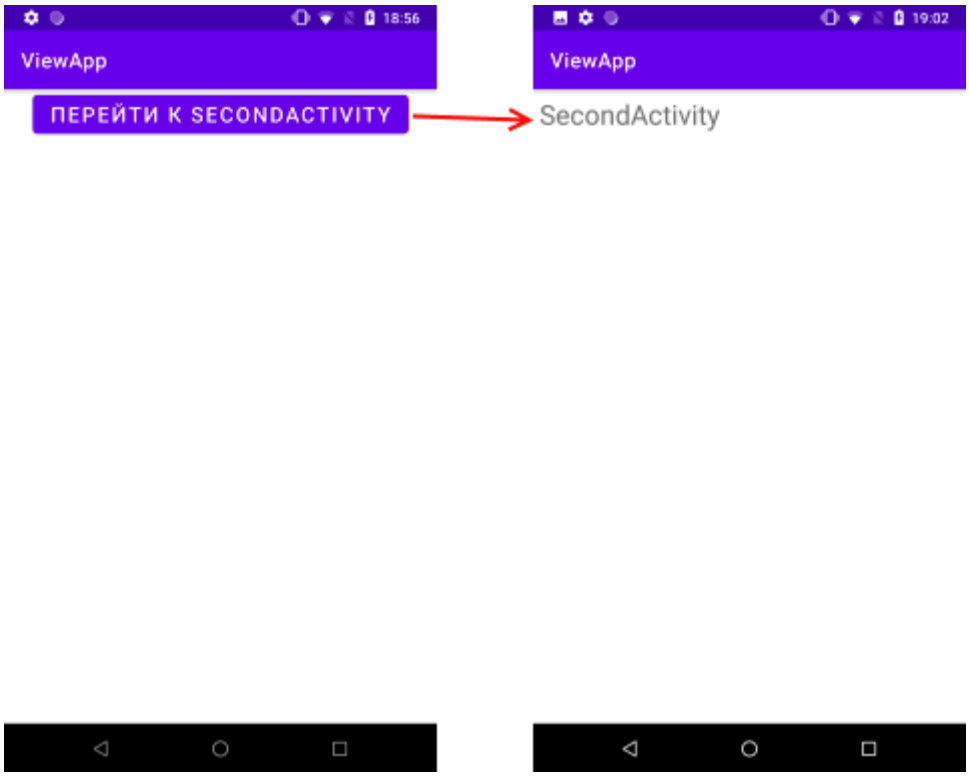
В обработчике нажатия будет запускаться SecondActivity.

Далее изменим код **SecondActivity:**

```
package com.example.viewapp;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;

public class SecondActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //setContentView(R.layout.activity_second);
        TextView textView = new TextView(this);
        textView.setTextSize(20);
        textView.setPadding(16, 16, 16, 16);
        textView.setText("SecondActivity");
        setContentView(textView);
```

```
    }
}
```

Запустим приложение и перейдем от одной Activity к другой:



Intent и startActivity в Android и Java

Передача данных между Activity. Сериализация.

Для передачи данных между двумя Activity используется объект Intent. Через его метод **putExtra()** можно добавить ключ и связанное с ним значение.

Например, передача из текущей activity в SecondActivity строки "Hello World" с ключом "hello":

```
// создание объекта Intent для запуска SecondActivity
Intent intent = new Intent(this, SecondActivity.class);
```

```
// передача объекта с ключом "hello" и значением "Hello World"
intent.putExtra("hello", "Hello World");
// запуск SecondActivity
startActivity(intent);
```

Для передачи данных применяется метод **putExtra()**, который в качестве значения позволяет передать данные простейших типов - String, int, float, double, long, short, byte, char, массивы этих типов, либо объект интерфейса Serializable.

Чтобы получить отправленные данные при загрузке SecondActivity, можно воспользоваться методом get(), в который передается ключ объекта:

```
Bundle arguments = getIntent().getExtras();
String name = arguments.get("hello").toString();    // Hello World
```

В зависимости от типа отправляемых данных при их получении мы можем 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 поле получения данное значение необходимо преобразовать к нужному типу
- **getString()**: возвращает объект типа String
- **getInt():** возвращает значение типа int
- **getByte()**: 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

Пусть у нас в проекте будет определено две activity: MainActivity и SecondActivity.

В коде SecondActivity определим получение данных:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.widget.TextView;
```

```
public class SecondActivity extends AppCompatActivity {

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

        TextView textView = new TextView(this);
        textView.setTextSize(26);
        textView.setPadding(16, 16, 16, 16);

        Bundle arguments = getIntent().getExtras();

        if(arguments!=null){
            String name = arguments.get("name").toString();
            String company = arguments.getString("company");
            int age = arguments.getInt("age");
            textView.setText("Name: " + name + "\nCompany: " + company +
                    "\nAge: " + age);
        }

        setContentView(textView);
    }
}
```

В данном случае в SecondActivity получаем все данных из объекта Bundle и выводим их в текстовое поле TextView. Предполагается, что данной activity будут передаваться три элемента - две строки с ключами name и company и число с ключом price.

Теперь определим передачу в SecondActivity данных. Например, определим для MainActivity следующий интерфейс в файле **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"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/nameLabel"
        android:layout_width="0dp"
        android:layout_height="20dp"
        android:text="Name:"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>
    <EditText
        android:id="@+id/name"
        android:layout_width="0dp"
        android:layout_height="40dp"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/nameLabel"/>
    <TextView
        android:id="@+id/companyLabel"
        android:layout_width="0dp"
        android:layout_height="20dp"
        android:text="Company:"
```

```
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/name"/>
<EditText
    android:id="@+id/company"
    android:layout_width="0dp"
    android:layout_height="40dp"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/companyLabel" />
<TextView
    android:id="@+id/ageLabel"
    android:layout_width="0dp"
    android:layout_height="20dp"
    android:text="Age:"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/company"/>
<EditText
    android:id="@+id/age"
    android:layout_width="0dp"
    android:layout_height="40dp"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/ageLabel"/>
<Button
    android:id="@+id/btn"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

```
        android:onClick="onClick"
        android:text="Save"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/age"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Здесь определены три текстовых поля для ввода данных и кнопка.

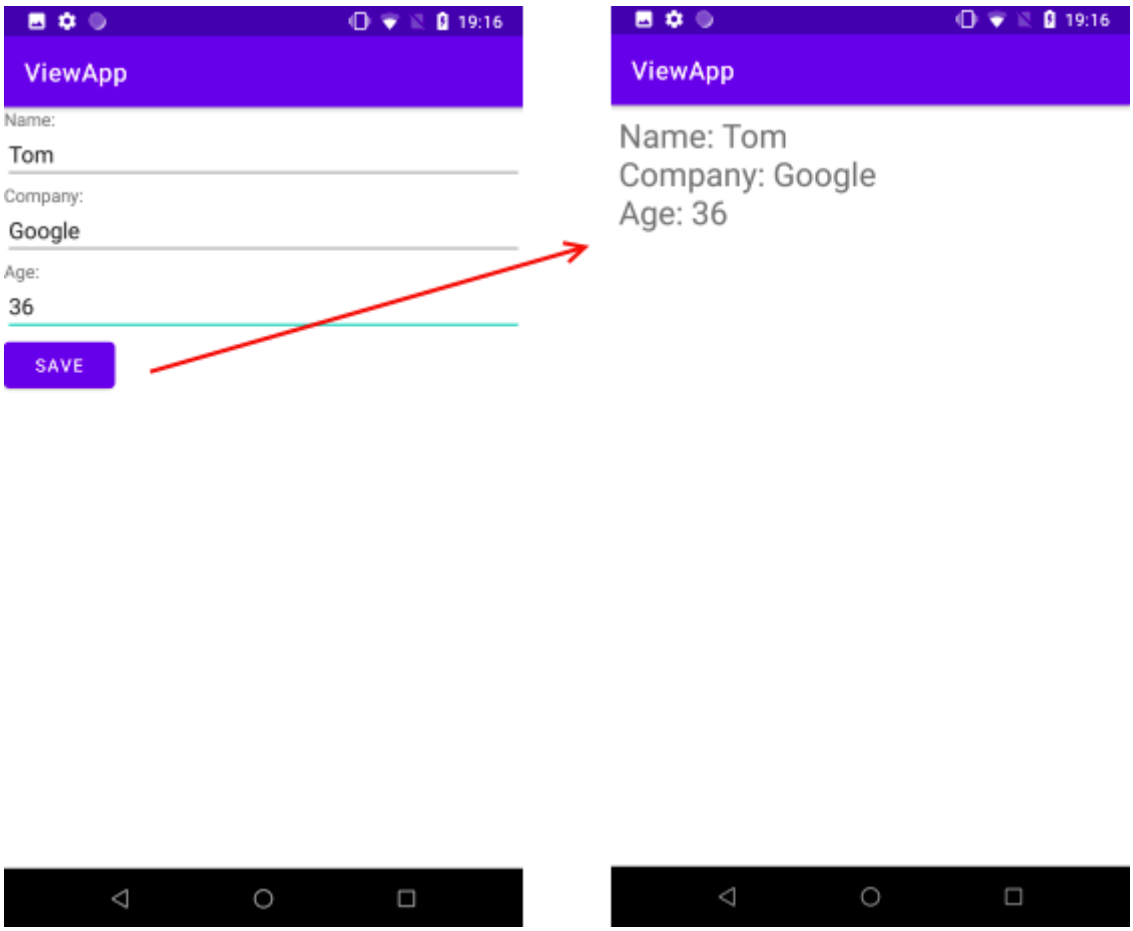
В классе MainActivity определим следующее содержимое:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity {

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

        EditText nameText = findViewById(R.id.name);
```

```
        EditText companyText = findViewById(R.id.company);
        EditText ageText = findViewById(R.id.age);

        String name = nameText.getText().toString();
        String company = companyText.getText().toString();
        int age = Integer.parseInt(ageText.getText().toString());

        Intent intent = new Intent(this, SecondActivity.class);
        intent.putExtra("name", name);
        intent.putExtra("company", company);
        intent.putExtra("age", age);
        startActivity(intent);
    }
}
```

В обработчике нажатия кнопки получаем введенные в текстовые поля EditText данные и передаем их в объект Intent с помощью метода putExtra(). Затем запускаем SecondActivity.

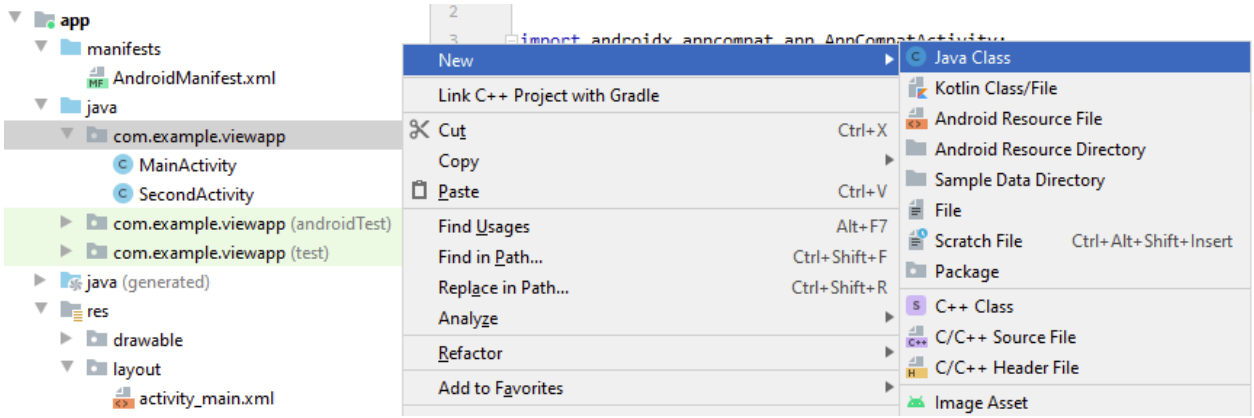
В итоге при нажатии на кнопку запустится SecondActivity, которая получит некоторые введенные в текстовые поля данные:



putExtra in Android и getString in Activity in Android Java

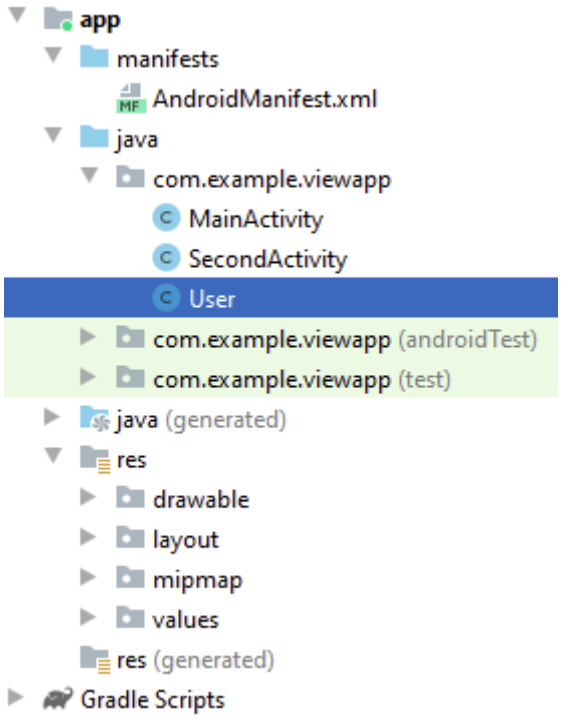
Передача сложных объектов

В примере выше передавались простые данные - числа, строки. Но также мы можем передавать более сложные данные. В этом случае используется механизм сериализации. Для этого нажмем на папку пакета, где находятся классы MainActivity и SecondActivity, правой кнопкой мыши и в контекстном меню выберем **New -> Java Class**:



Добавление класса Java в Android Studio

Назовем новый класс **User** - пусть он будет представлять пользователя.



Добавление класса Java в Android и передача сложных объектов в MainActivity

Пусть класс User имеет следующий код:

```
package com.example.viewapp;
import java.io.Serializable;
public class User implements Serializable {
    private String name;
    private String company;
    private int age;

    public User(String name, String company, int age){
        this.name = name;
        this.company = company;
        this.age = age;
    }
```

```
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getCompany() {
        return company;
    }

    public void setCompany(String company) {
        this.company = company;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

Стоит отметить, что данный класс реализует интерфейс **Serializable**. Теперь изменим код MainActivity:

```
package com.example.viewapp;
```

```
import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity {

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

        EditText nameText = findViewById(R.id.name);
        EditText companyText = findViewById(R.id.company);
        EditText ageText = findViewById(R.id.age);

        String name = nameText.getText().toString();
        String company = companyText.getText().toString();
        int age = Integer.parseInt(ageText.getText().toString());

        User user = new User(name, company, age);

        Intent intent = new Intent(this, SecondActivity.class);
        intent.putExtra(User.class.getSimpleName(), user);
```

```
        startActivity(intent);
    }
}
```

Теперь вместо трех разрозненных данных передается один объект User. В качестве ключа используется результат метода User.class.getSimpleName(), который по сути возвращает название класса.

И изменим класс SecondActivity:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.widget.TextView;

public class SecondActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //setContentView(R.layout.activity_second);
        TextView textView = new TextView(this);
        textView.setTextSize(26);
        textView.setPadding(16, 16, 16, 16);

        Bundle arguments = getIntent().getExtras();

        User user;
```

```
        if(arguments!=null){
            user = (User) arguments.getSerializable(User.class.getSimpleName());

            textView.setText("Name: " + user.getName() + "\nCompany: " +
user.getCompany() +
                    "\nAge: " + String.valueOf(user.getAge()));
        }
        setContentView(textView);
    }
}
```

Для получения данных применяется метод getSerializable(), поскольку класс User реализует интерфейс Serializable. Таким образом, мы можем передать один единственый объект вместо набора разрозненных данных

Parcelable

Возможность сериализации объектов предоставляется напрямую инфраструктурой языка Java. Однако Android также предоставляет интерфейс **Parcelable**, который, по сути, также позволяет сериализовать объекты, как и Serializable, но является более оптимизированным для Android. И подобные объекты Parcelable также можно передавать между двумя activity или использовать каким-то иным образом.

Например, в прошлом материале данные передавались между activity в виде объектов User, которые использовали сериализацию. Теперь пусть класс User применяет интерфейс Parcelable:

```
package com.example.viewapp;

import android.os.Parcel;
import android.os.Parcelable;
```

```
public class User implements Parcelable {

    private String name;
    private String company;
    private int age;

    public static final Creator<User> CREATOR = new Creator<User>() {
        @Override
        public User createFromParcel(Parcel source) {
            String name = source.readString();
            String company = source.readString();
            int age = source.readInt();
            return new User(name, company, age);
        }

        @Override
        public User[] newArray(int size) {
            return new User[size];
        }
    };

    public User(String name, String company, int age){
        this.name = name;
        this.company = company;
        this.age = age;
    }
    public String getName() {
        return name;
    }
```

```
    public void setName(String name) {
        this.name = name;
    }
    public String getCompany() {
        return company;
    }
    public void setCompany(String company) {
        this.company = company;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public int describeContents() {
        return 0;
    }

    @Override
    public void writeToParcel(Parcel dest, int flags) {
        dest.writeString(name);
        dest.writeString(company);
        dest.writeInt(age);
    }
}
```

Интерфейс android.os.Parcelable предполагает реализацию двух методов: describeContents() и writeToParcel(). Первый метод описывает контент и возвращает некторое числовое значение. Второй метод пишет в объект Parcel содержимое объекта User.

Для записи данных объекта в Parcel используется ряд методов, каждый из которых предназначен для определенного типа данных. Основные методы:

- **writeString()**
- **writeInt()**
- **writeFloat()**
- **writeDouble()**
- **writeByte()**
- **writeLong()**
- **writeIntArray()**
- **writeValue()** (записывает объект типа Object)
- **writeParcelable()** (записывает объект типа Parcelable)

Кроме того, объект Parcelable должен содержать статическое поле CREATOR, которое представляет объект Creator<User>. Причем этот объект реализует два метода. Они нужны для создания их ранее сериализованных данных исходных объектов типа User.

Так, метод newArray() создает массив объект User.

Метод **createFromParcel** создает из Parcel новый объект типа User. То есть этот метод противоположен по действию методу writeToParcel. Для получения данных из Parcel применяются методы типа readString(), readInt(), readParcelable() и так далее - для чтения определенных типов данных.

Причем важно, что данные в createFromParcel считываются из объекта Parcel именно в том порядке, в котором они добавляются в этот объект в методе writeToParcel.

Допустим в activity, которая называется SecondActivity мы будем получать объект User:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.widget.TextView;

public class SecondActivity extends AppCompatActivity {

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

        TextView textView = new TextView(this);
        textView.setTextSize(26);
        textView.setPadding(16, 16, 16, 16);
```

```
        Bundle arguments = getIntent().getExtras();


        User user;
        if(arguments!=null){
            user = arguments.getParcelable(User.class.getSimpleName());


            textView.setText("Name: " + user.getName() + "\nCompany: " +
user.getCompany() +
                    "\nAge: " + String.valueOf(user.getAge()));
        }
        setContentView(textView);
    }
}
```

Для получения объекта Parcelable, переданного в activity, применяется метод getParcelable(). Причем никакого приведения типов не требуется.

Для тестирования передачи Parcelable определим в файле **activity_main.xml** простейший интерфейс для MainActivity:

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


    <TextView
        android:id="@+id/nameLabel"
        android:layout_width="0dp"
```

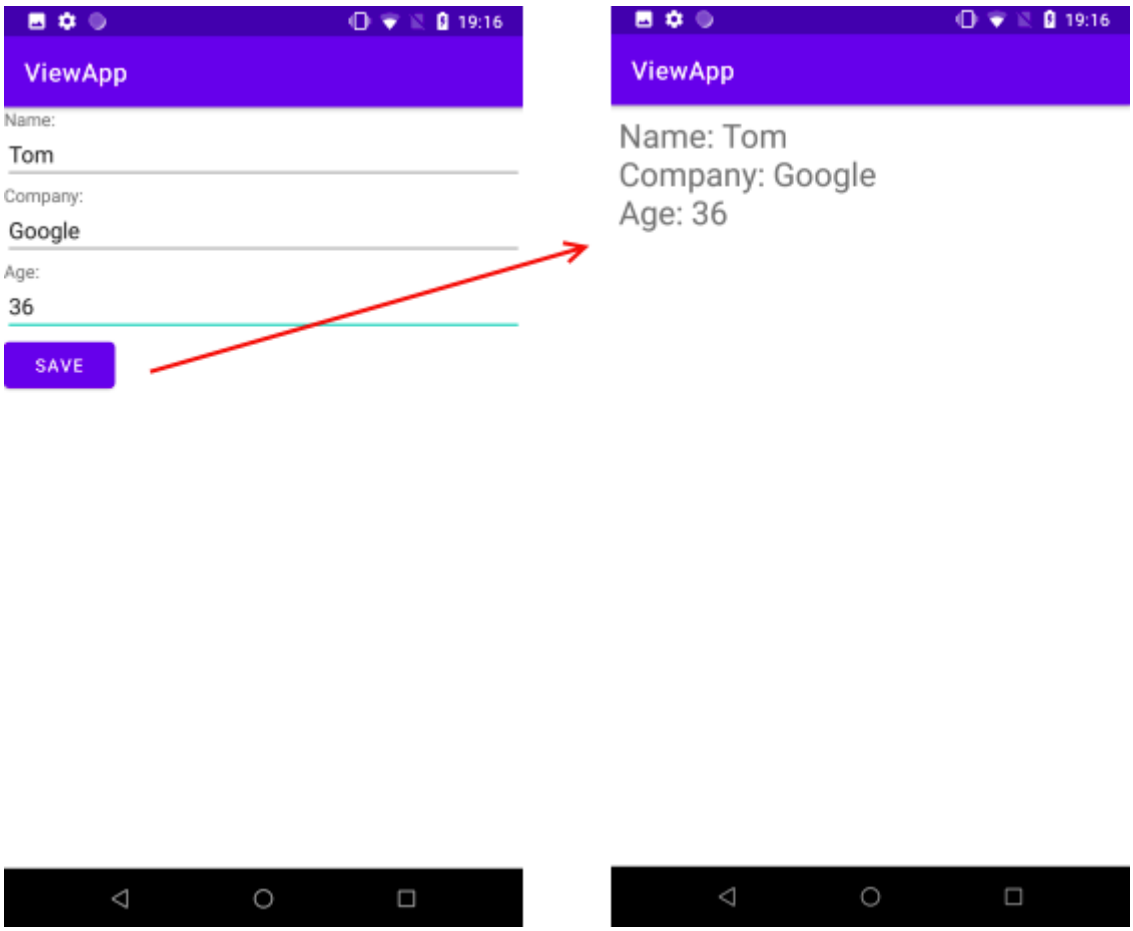
```
    android:layout_height="20dp"
    android:text="Name:"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>
<EditText
    android:id="@+id/name"
    android:layout_width="0dp"
    android:layout_height="40dp"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/nameLabel"/>
<TextView
    android:id="@+id/companyLabel"
    android:layout_width="0dp"
    android:layout_height="20dp"
    android:text="Company:"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/name"/>
<EditText
    android:id="@+id/company"
    android:layout_width="0dp"
    android:layout_height="40dp"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/companyLabel" />
<TextView
    android:id="@+id/ageLabel"
```

```
        android:layout_width="0dp"
        android:layout_height="20dp"
        android:text="Age:"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/company"/>
    <EditText
        android:id="@+id/age"
        android:layout_width="0dp"
        android:layout_height="40dp"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/ageLabel"/>
    <Button
        android:id="@+id/btn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onClick"
        android:text="Save"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/age"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

А в коде MainActivity определим передачу данных в SecondActivity:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;
```

```
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity {

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

        EditText nameText = findViewById(R.id.name);
        EditText companyText = findViewById(R.id.company);
        EditText ageText = findViewById(R.id.age);

        String name = nameText.getText().toString();
        String company = companyText.getText().toString();
        int age = Integer.parseInt(ageText.getText().toString());

        User user = new User(name, company, age);

        Intent intent = new Intent(this, SecondActivity.class);
        intent.putExtra(User.class.getSimpleName(), user);
        startActivity(intent);
    }
}
```

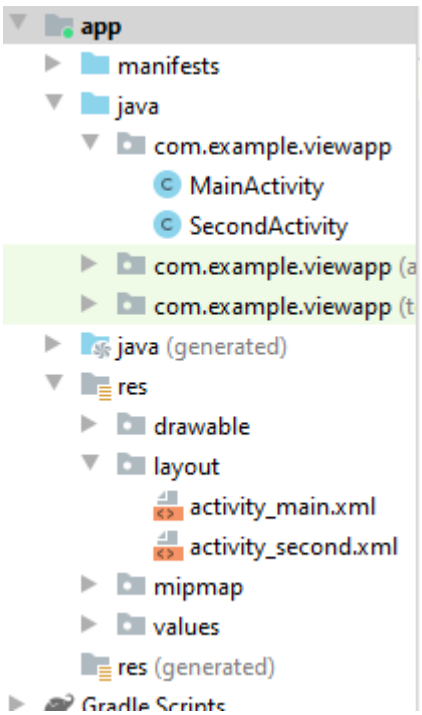


putExtra in Android и getString in Activity in Android Java

Получение результата из Activity

В прошлом материале было рассмотрено как вызывать новую Activity и передавать ей некоторые данные. Но мы можем не только передавать данные запускаемой activity, но и ожидать от нее некоторого результата работы.

К примеру, пусть у нас в проекте будут две activity: **MainActivity** и **SecondActivity.** А для каждой activity есть свой файл интерфейса: **activity_main.xml** и **activity_second.xml** соответственно.



startActivityForResult в Android и Java

В прошлом материале мы вызывали новую activity с помощью метода startActivity(). Для получения же результата работы запускаемой activity необходимо использовать **Activity Result API**.

Activity Result API предоставляет компоненты для регистрации, запуска и обработки результата другой Activity. Одним из преимуществ применения Activity Result API является то, что он отвязывает результат Activity от самой Activity. Это позволяет получить и обработать результат, даже если Activity, которая возвращает результат, в силу ограничений памяти или в силу других причин завершила свою работу. Вкратце рассмотрим основные моменты применения Activity Result API.

Регистрация функции для получения результата

Для регистрации функции, которая будет обрабатывать результат, Activity Result API предоставляет метод **registerForActivityResult().** Этот метод в качестве параметров принимает объекты **ActivityResultContract** и **ActivityResultCallback** и возвращает объект **ActivityResultLauncher,** который применяется для запуска другой activity.

ActivityResultLauncher<I> registerForActivityResult (

```
        ActivityResultContract<I, O> contract,
        ActivityResultCallback<O> callback)
```

ActivityResultContract определяет контракт: данные какого типа будут подаваться на вход и какой тип будет представлять результат.

ActivityResultCallback представляет интерфейс с единственным методом **onActivityResult(),** который определяет обработку полученного результата. Когда вторая activity закончит работу и возвратит результат, то будет как раз вызываться этот метод. Результат передается в метод в качестве параметра. При этом тип параметра должен соответствовать типу результата, определенного в **ActivityResultContract**. Например:

```
ActivityResultLauncher<Intent> mStartForResult = registerForActivityResult(new
ActivityResultContracts.StartActivityForResult(),
   new ActivityResultCallback<ActivityResult>() {
      @Override
      public void onActivityResult(ActivityResult result) {

         // обработка result
      }
});
```

Класс **ActivityResultContracts** предоставляет ряд встроенных типов контрактов. Например, в листинге кода выше применяется встроенный тип **ActivityResultContracts.StartActivityForResult**, который в качестве входного объекта устанавливает объект **Intent,** а в качестве типа результата - тип **ActivityResult.**

Запуск activity для получения результата

Метод registerForActivityResult() регистрирует функцию-колбек и возвращает объект ActivityResultLauncher. С помощью этого мы можем запустить activity. Для этого у объекта ActivityResultLauncher вызывается метод launch():

```
mStartForResult.launch(intent);
```

В метод **lauch()** передается объект того типа, который определен объектом **ActivityResultContracts** в качестве входного.

Практическое применение Activity Result API

Итак, определим в файле activity_main.xml следующий интерфейс:

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

    <TextView
        android:id="@+id/textView"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Укажите возраст"
        android:textSize="22sp"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>
    <EditText
        android:id="@+id/age"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView"/>
    <Button
```

```
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onClick"
        android:text="Отправить"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/age"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Для ввода данных здесь определен элемент EditText, а для отправки - кнопка.

Определим в классе **MainActivity** запуск второй activity:

```
package com.example.viewapp;

import androidx.activity.result.ActivityResult;
import androidx.activity.result.ActivityResultCallback;
import androidx.activity.result.ActivityResultLauncher;
import androidx.activity.result.contract.ActivityResultContracts;
import androidx.appcompat.app.AppCompatActivity;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
```

```
    static final String AGE_KEY = "AGE";
    static final String ACCESS_MESSAGE="ACCESS_MESSAGE";


    ActivityResultLauncher<Intent> mStartForResult =
registerForActivityResult(new ActivityResultContracts.StartActivityForResult(),
            new ActivityResultCallback<ActivityResult>() {
                @Override
                public void onActivityResult(ActivityResult result) {


                    TextView textView = findViewById(R.id.textView);
                    if(result.getResultCode() == Activity.RESULT_OK){
                        Intent intent = result.getData();
                        String accessMessage =
intent.getStringExtra(ACCESS_MESSAGE);
                        textView.setText(accessMessage);
                    }
                    else{
                        textView.setText("Ошибка доступа");
                    }
                }
            });
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    public void onClick(View view) {
        // получаем введенный возраст
```

```
        EditText ageBox = findViewById(R.id.age);
        String age = ageBox.getText().toString();

        Intent intent = new Intent(this, SecondActivity.class);
        intent.putExtra(AGE_KEY, age);

        mStartForResult.launch(intent);
    }
}
```

Вкратце рассмотрим главные моменты этого кода. Прежде всего, мы определяем объект **ActivityResultLauncher**, с помощью которого будем запускать вторую activity и передавать ей данные:

```
ActivityResultLauncher<Intent> mStartForResult = registerForActivityResult(new
ActivityResultContracts.StartActivityForResult(),
            new ActivityResultCallback<ActivityResult>() {
                @Override
                public void onActivityResult(ActivityResult result) {

                    TextView textView = findViewById(R.id.textView);
                    if(result.getResultCode() == Activity.RESULT_OK){
                        Intent intent = result.getData();
                        String accessMessage =
intent.getStringExtra(ACCESS_MESSAGE);
                        textView.setText(accessMessage);
                    }
                    else{
                        textView.setText("Ошибка доступа");
                    }
                }
```

});

Объект **ActivityResultLauncher** типизируется типом **Intent**, так как объект этого типа будет передаваться в метод launch() при запуске второй activity.

Тип контракта определяется типом ActivityResultContracts.StartActivityForResult, который и определяет тип Intent в качестве входного типа и тип ActivityResult в качестве типа результата.

Второй аргумент метода registerForActivityResult() - объект **ActivityResultCallback** типизируется типом результата - типом ActivityResult и определяет функцию-колбек **onActivityResult(),** которая получает результат и обрабатывает его. В данном случае обработка состоит в том, что мы выводим в текстовое поле ответ от второй activity.

При обработке мы проверяем полученный код результата:

```
if(result.getResultCode() == Activity.RESULT_OK)
```

В качестве результата, как правило, применяются встроенные константы Activity.RESULT_OK и Activity.RESULT_CANCELED. На уровне условностей Activity.RESULT_OK означает, что activity успешно обработала запрос, а Activity.RESULT_CANCELED - что activity отклонила обработку запроса.

С помощью метода getData() результата получаем переданные из второй activity данные в виде объекта Intent:

```
Intent intent = result.getData();
```

Далее извлекаем из Intent строку, которая имеют ключ ACCESS_MESSAGE, и выводим ее в текстовое поле.

Таким образом, мы определили объект ActivityResultLauncher. Далее в обработчике нажатия onClick с помощью этого объекта запускаем вторую activity - SecondActivity:

```
public void onClick(View view) {

    // получаем введенный возраст
    EditText ageBox = findViewById(R.id.age);
    String age = ageBox.getText().toString();


    Intent intent = new Intent(this, SecondActivity.class);
    intent.putExtra(AGE_KEY, age);


    mStartForResult.launch(intent);
}
```

В обработчике нажатия кнопки onClick() получаем введенный в текстовое поле возраст, добавляем его в объект Intent с ключем AGE_KEY и запускаем SecondActivity с помощью метода launch()

Теперь перейдем к SecondActivity и определим в файле activity_second.xml набор кнопок:

```
<?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"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <TextView
        android:id="@+id/ageView"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
```

```
    android:textSize="26sp"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>

<Button
    android:id="@+id/button1"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="Открыть доступ"
    android:onClick="onButton1Click"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/ageView"/>

<Button
    android:id="@+id/button2"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="Отклонить доступ"
    android:onClick="onButton2Click"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/button1"/>

<Button
    android:id="@+id/button3"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
```

```
        android:text="Возраст недействителен"
        android:onClick="onButton3Click"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/button2" />

    <Button
        android:id="@+id/cancel"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Отмена"
        android:onClick="onCancelClick"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/button3" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

А в классе **SecondActivity** определим обработчики для этих кнопок:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class SecondActivity extends AppCompatActivity {
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_second);
    Bundle extras = getIntent().getExtras();
    if (extras != null) {
        TextView ageView = findViewById(R.id.ageView);
        String age = extras.getString(MainActivity.AGE_KEY);
        ageView.setText("Возраст: " +  age);
    }
}
public void onCancelClick(View v) {
    setResult(RESULT_CANCELED);
    finish();
}
public void onButton1Click(View v) {
    sendMessage("Доступ разрешен");
}
public void onButton2Click(View v) {
    sendMessage("Доступ запрещен");
}
public void onButton3Click(View v) {
    sendMessage("Недопустимый возраст");
}
private void sendMessage(String message){

    Intent data = new Intent();
    data.putExtra(MainActivity.ACCESS_MESSAGE, message);
```

```
        setResult(RESULT_OK, data);
        finish();
    }
}
```

Три кнопки вызывают метод sendMessage(), в который передают отправляемый ответ. Это и будет то сообщение, которое получить MainActivity в методе onActivityResult.

Для возврата результата необходимо вызвать метод **setResult(),** в который передается два параметра:

- числовой код результата
- отправляемые данные

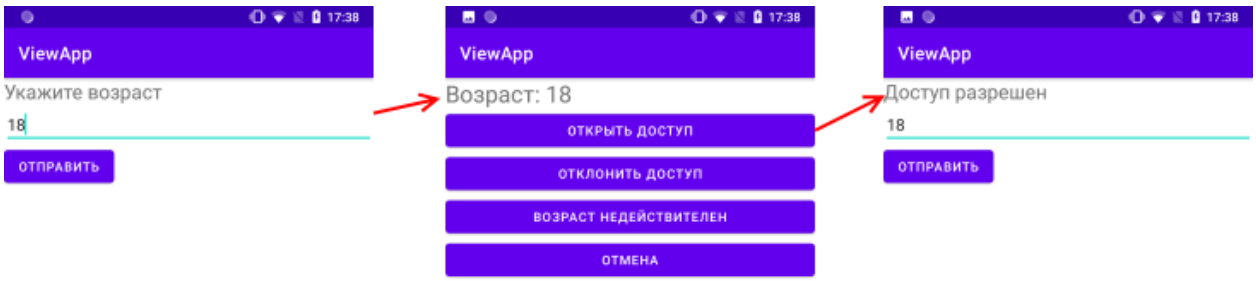
После вызова метода setResult() нужно вызвать метод finish, который уничтожит текущую activity.

Одна кнопка вызывает обработчик onCancelClick(), в котором передается в setResult только код результата - RESULT_CANCELED.

То есть условно говоря, мы получаем в SecondActivity введенный в MainActivity возраст и с помощью нажатия определенной кнопки возвращаем некоторый результат в виде сообщения.

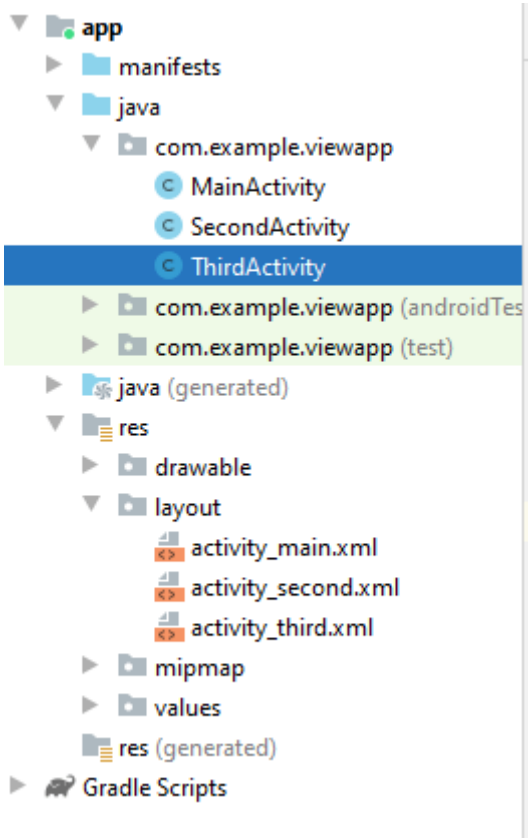
В зависимости от нажатой кнопки на SecondActivity мы будем получать разные результаты в MainActivity:

Получение результата из Activity в Android и Java Метод startActivityForResult в Android и Java



Взаимодействие между Activity

В прошлых темах мы рассмотрели жизненный цикл activity и запуск новых activity с помощью объекта Intent. Теперь рассмотрим некоторые особенности взаимодействия между activity в одном приложении. Допустим, у нас есть три activity: MainActivity, SecondActivity и ThirdActivity.



Back stak и управление стеком из Activity в Android и Java

С помощью Intent, например, по нажатию кнопки MainActivity запускает SecondActivity:

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

startActivity(intent);

На SecondActivity тоже есть кнопка, которая запускает ThirdActivity:

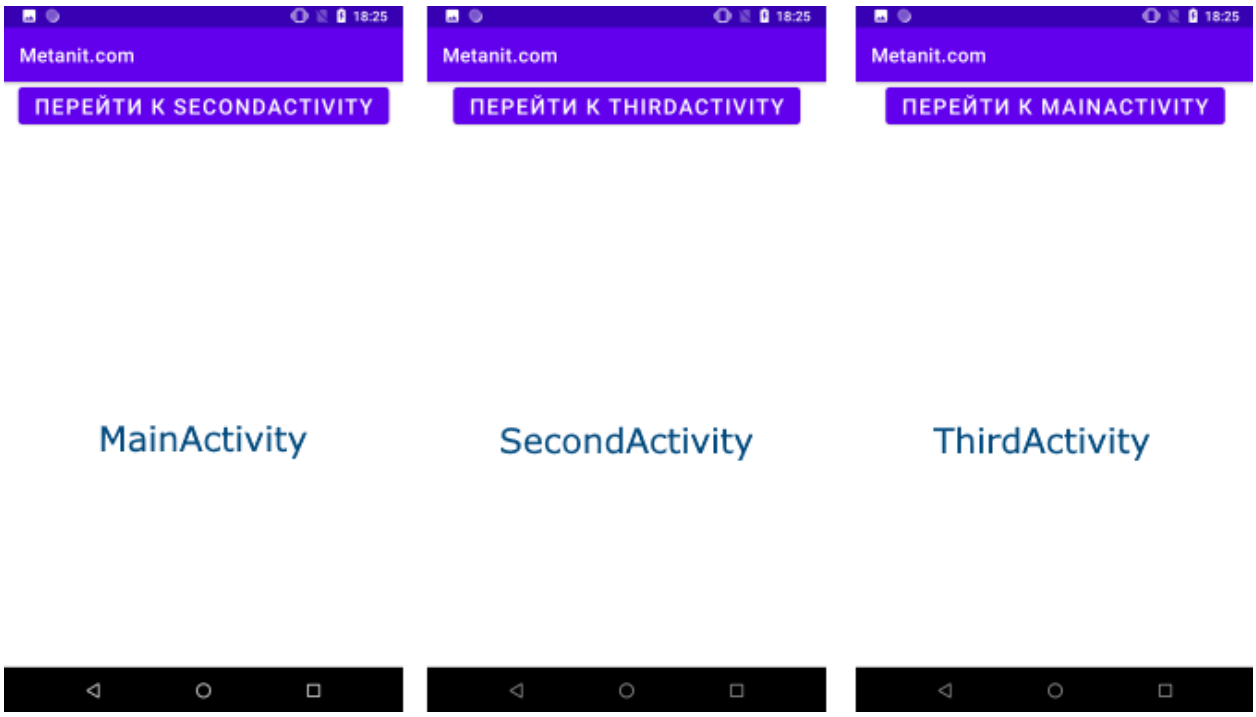
Intent intent = new Intent(this, ThirdActivity.class);

startActivity(intent);

На ThirdActivity также есть кнопка, которая возвращается к первой activity - MainActivity:

Intent intent = new Intent(this, MainActivity.class);

startActivity(intent);



Взаимодействие между Activity в Android и Java

Если мы последовательно запустим все activity: из главной MainActivity запустим SecondActivity, из SecondActivity - ThirdActivity, то в итоге у нас сложится следующий стек activity:

ThirdActivity

SecondActivity

MainActivity

Если после этого из ThirdActivity мы захотим обратиться к MainActivity, то метод startActivity() запустит новый объект MainActivity (а не вернется к уже существующему), и стек уже будет выглядеть следующим образом:

MainActivity

ThirdActivity

SecondActivity

MainActivity

То есть у нас будут две независимые копии MainActivity. Такое положение нежелательно, если мы просто хотим перейти к существующей. И этот момент надо учитывать.

Если мы нажмем на кнопку **Back (Назад)**, то текущая activity, которая находится на вершине стека, удаляется из стека, и предыдущая activity оказывается на вершине стека и возобновляет свою работу. И таким образом с помощью кнопки Back (Назад) мы сможем перейти к предыдущей activity в стеке. Например, в случае выше если мы нажмем на кнопку Назад, то MainActivity на вершине стека завершает свою работу, и на экране начинает отображаться ThirdActivity

ThirdActivity

SecondActivity

MainActivity

Тем не менее иногда возникает необходимость упавлять переходом между activity. Например, в данном случае нам нежелательно при нажатии на кнопку в ThirdActivity запускать новую копию MainActivity вместо того, чтобы просто перейти к MainActivity, которая была запущена первой и находится в самом низу стека. Рассмотрим, какие возможности предоставляет нам Android.

Управление стеком activity

Для управления стеком из activity Android предлагает нам использовать флаги - константы, определенные в классе **Intent**. Применение определенного флага позволит нам определенным образом изменить положение в стеке для определенных activity.

Например, возьмем предыдущую задачу, когда после нажатия на кнопку в ThirdActivity запускается новый экземпляр MainActivity. Но мы хотим не запускать новую, а перейти к уже существующей.

MainActivity

ThirdActivity

SecondActivity

MainActivity

Чтобы выйти из этой ситуации, мы можем использовать флаг **Intent.FLAG_ACTIVITY_REORDER_TO_FRONT**:

```
Intent intent = new Intent(this, MainActivity.class);
intent.addFlags(Intent.FLAG_ACTIVITY_REORDER_TO_FRONT);
startActivity(intent);
```

флаг **Intent.FLAG_ACTIVITY_REORDER_TO_FRONT** перемещает activity, к которой осуществляется переход на вершину стека, если она ужее есть в стеке. И в этом случае после перехода из ThirdActivity к MainActivity стек будет выглядеть следующим образом:

MainActivity

ThirdActivity

SecondActivity

Если же нам просто надо перейти из ThirdActivity к MainActivity, как если бы мы перешли назад с помощью кнопки Back, то мы можем использовать флаги **Intent.FLAG_ACTIVITY_CLEAR_TOP** и **Intent.FLAG_ACTIVITY_SINGLE_TOP:**

```
Intent intent = new Intent(this, MainActivity.class);
intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP |
Intent.FLAG_ACTIVITY_SINGLE_TOP);
startActivity(intent);
```

Флаг **Intent.FLAG_ACTIVITY_CLEAR_TOP** очищает все activity кроме той, которая запускается (если она уже есть в стеке). А флаг **Intent.FLAG_ACTIVITY_SINGLE_TOP** указывает, что если в вершине стеке уже есть activity, которую надо запустить, то она НЕ запускается (то она может существовать в стеке только в единичном виде).

В этом случае после перехода из ThirdActivity к MainActivity стек будет полностью очищен, и там останется одна MainActivity.

Еще один флаг - **Intent.FLAG_ACTIVITY_NO_HISTORY** позволит не сохранять в стеке запускаемую activity. Например, при запуске SecondActivity мы не хотим ее сохранять в стеке:

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

intent.addFlags(Intent.FLAG_ACTIVITY_NO_HISTORY);

startActivity(intent);
```

В этом случае при переходе по цепочке MainActivity -> SecondActivity -> ThirdActivity стек будет выглядеть следующим образом:

MainActivity

ThirdActivity

Задание

1. Создать приложения, которое использует в файле Манифест атрибуты **android:allowBackup, android:icon, android:roundIcon, android:label, android:theme** и реализует активити, которая должна быть входной точкой в приложение и не должна получать какие-либо данные извне и представлять стартовый экран
2. С помощью атрибутов элемента manifest определить версию созданного приложения и его кода.
3. Используя атрибуты произвести установку версии SDK
4. Реализовать установку разрешений к определенным ресурсам.
5. Реализовать поддержку разных разрешений, через ограничения использования приложения для определенных разрешений экранов
6. Реализовать запрет на изменение ориентации экрана
7. Добавить вторую активити, реализовать переход от одной активити к другой
8. Реализовать передачу данных между Activity. В зависимости от типа отправляемых данных при их получении можно использовать ряд методов объекта Bundle: **get(), getString(), getInt(), getByte(),…** и т.д.
9. Реализовать передачу сложных объектов
10. Реализовать интерфейс Parcelable
11. Реализовать получение результата из Activity. Регистрация функции для получения результата. Запуск activity для получения результата. Практическое применение Activity Result API.

12. Реализовать взаимодействие между четырьмя Activity. Реализовать управление стеком activity.