ACTIVITY

План:

- Activity і життєвий цикл додатку
- Управління життєвим циклом
- Ресурси рядків Файл маніфесту AndroidManifest.xml
- Intent і Intent-фільтри
- Intent-фільтри і дії
- Передача даних між активністю. Серіалізація
- Передача складних об'єктів
- Parcelable
- Отримання результату з Activity
- Взаємодія між Activity

Activity і життєвий цикл додатку

Ключовим компонентом для створення візуального інтерфейсу в додатку Android є activity (активність). Нерідко activity асоціюється з окремим екраном або вікном додатку, а перемикання між вікнами буде відбуватися як переміщення від однієї activity до іншої. Додаток може мати одну або кілька activity. Наприклад, в минулій темі робота додатку починалася з класу MainActivity:

```
public class MainActivity extends AppCompatActivity {
    // вміст класу
}
```

Всі об'єкти activity є об'єктами класу **android.app.Activity**, який містить базову функціональність для всіх activity. У додатку з минулого теми ми безпосередньо з цим класом не працювали, а MainActivity наслідувалась від класу **AppCompatActivity** . Однак сам клас AppCompatActivity, хоч і не прямо, успадковується від базового класу Activity.

Життєвий цикл додатку

Всі додатки Android мають строго визначений системою життєвий цикл. При запуску користувачем додатку система надає цьому додатку високий пріоритет. Кожна програма запускається у вигляді окремого процесу, що дозволяє системі давати одним процесам вищий пріоритет, на відміну від інших. Завдяки цьому, наприклад, при роботі з одними додатками не блокувати вхідні дзвінки. Після припинення роботи з додатком, система

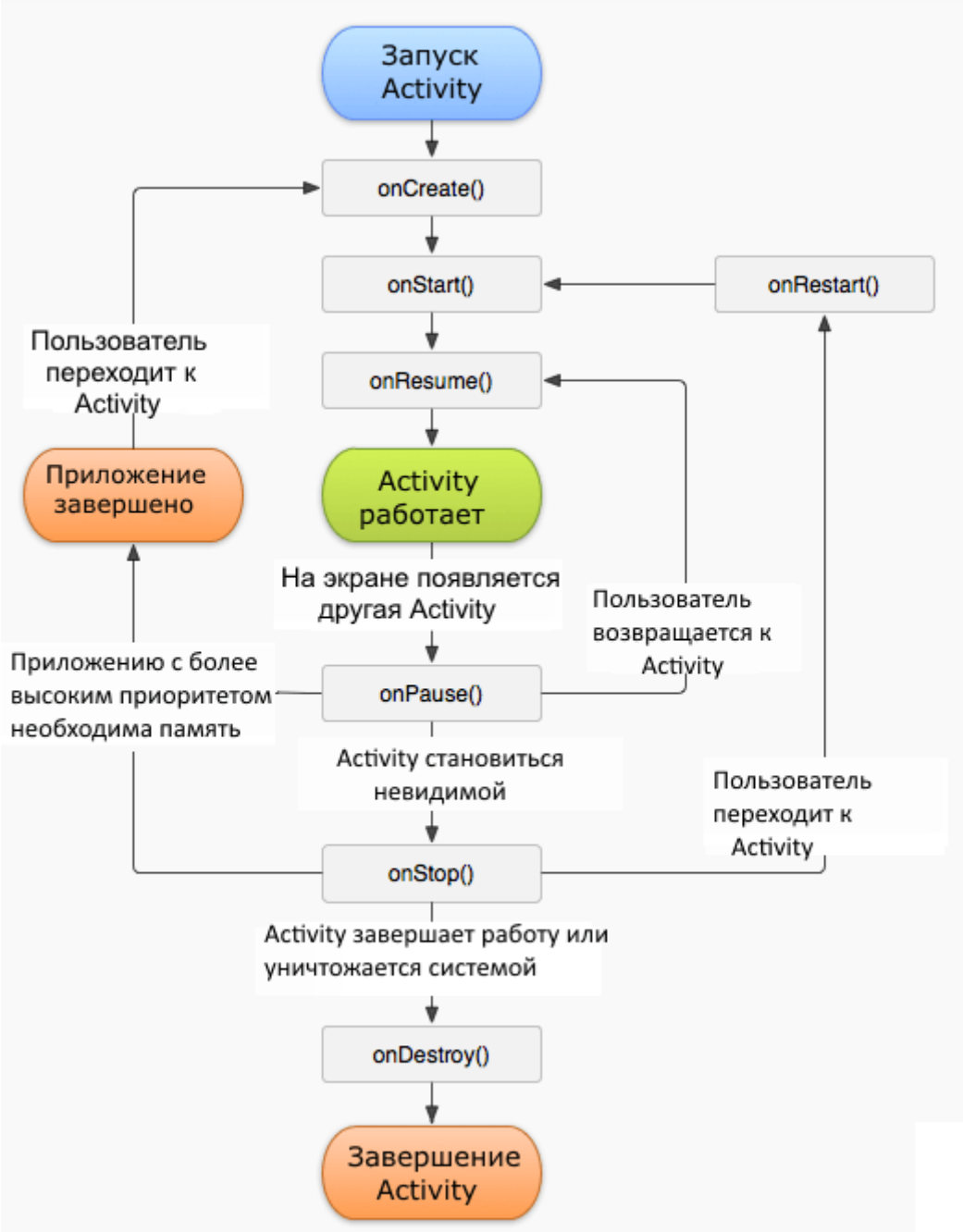
звільняє всі пов'язані ресурси і переводить додаток у розряд низькопріоритетного і закриває його.

Всі об'єкти activity, які є в додатку, управляються системою у вигляді стеку activity, який називається **back stack** . При запуску нової activity вона поміщається наверх стеку і виводиться на екран пристрою, поки не з'явиться нова activity. Коли поточна activity закінчує свою роботу (наприклад, користувач виходить з програми), то вона видаляється з стеку, і відновлює роботу activity, яка раніше була другою в стеці.

Після запуску activity проходить через ряд подій, які обробляються системою і для обробки яких існує ряд зворотних викликів:

```
protected void onCreate(Bundle saveInstanceState);
protected void onStart();
protected void onRestoreInstanceState(Bundle saveInstanceState);
protected void onRestart();
protected void onResume();
protected void onPause();
protected void onSaveInstanceState(Bundle saveInstanceState);
protected void onStop();
protected void onDestroy();
```

Схематично взаємозв'язок між усіма цими зворотними викликами можна представити таким чином:



onCreate()

onCreate - перший метод, з якого починається виконання activity. У цьому методі activity переходить в стан Created. Цей метод обов'язково повинен бути визначений в класі activity. У ньому проводиться початкове налаштування activity. Зокрема, створюються об'єкти візуального інтерфейсу. Цей метод отримує об'єкт Bundle, який містить попередній стану activity, якщо він був збережений. Якщо activity заново створюється, то даний об'єкт має значення null. Якщо ж activity вже раніше була створена, але перебувала в зупиненому стані, то bundle містить пов'язану з activity інформацію.

onStart

У методі **onStart()** здійснюється підготовка до виводу activity на екран пристрою. Як правило, цей метод не вимагає перевизначення, а всю роботу робить вбудований код. Після завершення роботи методу activity відображається на екрані, викликається метод **onResume** , а activity переходить в стан Resumed.

onRestoreInstanceState

Після завершення методу onStart()викликається метод onRestoreInstanceState, який покликаний відновлювати збережений стан з об'єкта Bundle, який передається в якості

параметра. Але слід враховувати, що цей метод викликається тільки тоді, коли Bundle НЕ дорівнює **null** і містить раніше збережений стан. Так, при першому запуску програми цей об'єкт Bundle матиме значення null, тому і метод onRestoreInstanceState не викликатиметься.

onResume

А при виклику методу onResume activity переходить в стан Resumed, а користувач може з нею взаємодіяти. І власне activity залишається в цьому стані, поки вона не втратить фокус, наприклад, внаслідок перемикання на іншу activity або просто через виключення екрану пристрою.

onPause

Якщо користувач вирішить перейти до іншої activity, то система викликає метод **onPause**. У цьому методі можна звільняти використовувані ресурси, припиняти процеси, наприклад, відтворення аудіо, анімацій, зупиняти роботу камери (якщо вона використовується) і т.д., щоб вони менше позначалися на продуктивності системи.

Але треба враховувати, що на роботу даного методу відводиться дуже мало часу, тому не варто тут зберігати якісь дані, особливо якщо при цьому потрібно звернення до мережі, наприклад, відправка даних по інтернету, або звернення до бази даних.

Після виконання цього методу activity стає невидимою, не відображається на екрані, але вона все ще активна. І якщо користувач вирішить повернутися до цієї activity, то система викличе знову метод onResume, і activity знову з'явиться на екрані.

Інший варіант роботи може виникнути, якщо раптом система бачить, що для роботи активних додатків необхідно більше пам'яті. І система може сама завершити повністю роботу activity, яка невидима і знаходиться в фоні. Або користувач може натиснути на кнопку Back (Назад). В цьому випадку у activity викликається метод **onStop** .

onSaveInstanceState

Метод **onSaveInstanceState** викликається після методу **onPause()**, але до виклику **onStop()**. У onSaveInstanceState проводиться збереження стану програми в переданий в якості параметра об'єкт Bundle.

onStop

У цьому методі activity переходить в стан Stopped. У методі onStop слід звільнити використовувані ресурси, які не потрібні користувачеві, коли він не взаємодіє з activity. Тут також можна зберігати дані, наприклад, в базу даних.

При цьому під час стану Stopped activity залишається в пам'яті пристрою, зберігається стан всіх елементів інтерфейсу. Наприклад, якщо в текстове поле EditText був введений якийсь текст, то після відновлення роботи activity і переходу її в стан Resumed ми знову побачимо в текстовому полі раніше введений текст.

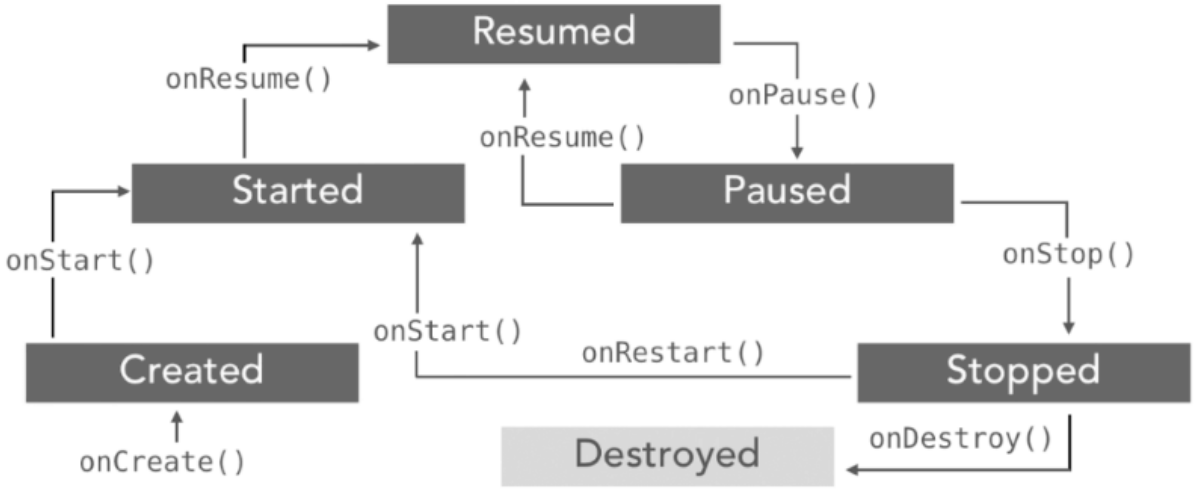
Якщо після виклику методу **onStop** користувач вирішить повернутися до колишньої activity, тоді система викличе метод **onRestart** . Якщо ж activity зовсім завершила свою роботу, наприклад, через закриття програми, то викликається метод **onDestroy()** .

onDestroy

Завершується робота активності викликом методу **onDestroy** , який виникає або, якщо система вирішить вбити activity, або при виклику методу **finish()** .

Також слід зазначити, що при зміні орієнтації екрану система завершує activity і потім створює її заново, викликаючи метод **onCreate** .

В цілому перехід між станами activity можна виразити наступною схемою:



Управління життєвим циклом

Ми можемо управляти цими подіями життєвого циклу, перевизначивши відповідні методи. Для цього візьмемо з минулого розділу клас MainActivity і змінимо його наступним чином:

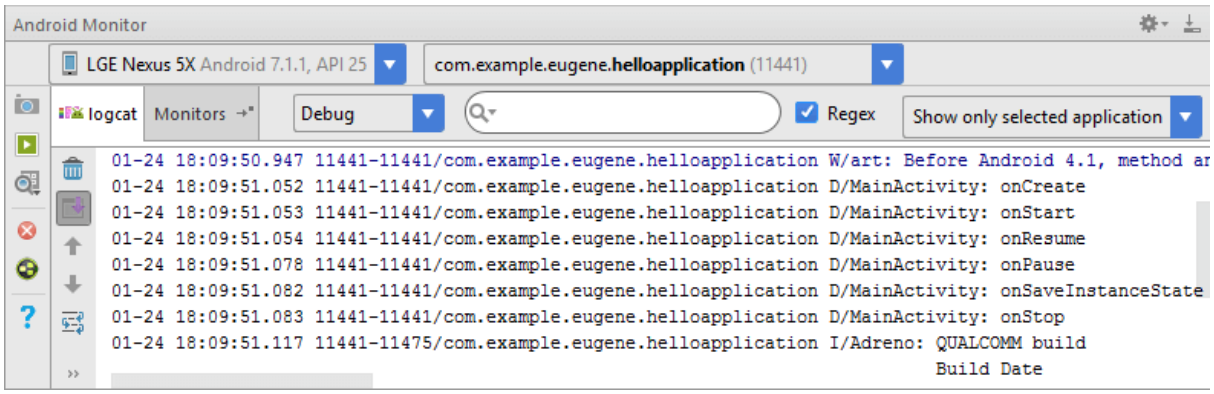
```
package com.example.ozm.viewsapplication;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;

public class MainActivity extends AppCompatActivity {

    private final static String TAG = "MainActivity";

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.d(TAG, "onCreate");
    }

    @Override
    protected void onDestroy(){
        super.onDestroy();
        Log.d(TAG, "onDestroy");
    }
    @Override
    protected void onStop(){
        super.onStop();
        Log.d(TAG, "onStop");
    }
    @Override
    protected void onStart(){
        super.onStart();
        Log.d(TAG, "onStart");
    }
    @Override
    protected void onPause(){
        super.onPause();
        Log.d(TAG, "onPause");
    }
    @Override
    protected void onResume(){
```

```
        super.onResume();
        Log.d(TAG, "onResume");
    }

    @Override
    protected void onRestart(){
        super.onRestart();
        Log.d(TAG, "onRestart");
    }

    @Override
    protected  void onSaveInstanceState(Bundle outState){
        super.onSaveInstanceState(outState);
        Log.d(TAG, "onSaveInstanceState");
    }

    @Override
    protected  void onRestoreInstanceState(Bundle savedInstanceState){
        super.onRestoreInstanceState(savedInstanceState);
        Log.d(TAG, "onRestoreInstanceState");
    }
}
```

Для логування подій тут використовується клас **android.util.Log** .

В даному випадку обробляються всі ключові методи життєвого циклу. Вся обробка зведена до виклику методу Log.d(), в який передається TAG - випадкове значення рядка і рядок, який виводиться в консолі logcat внизу Android Studio в вікні Android Monitor, виконуючи роль налагоджувальної інформації. Якщо ця консоль за замовчуванням прихована, то ми можемо перейти до неї через пункт меню **View -> Tool Windows -> Android Monitor** .

І під час запуску програми ми зможемо побачити у вікні **logcat** зневадження, яка визначається в методах життєвого циклу activity:



Ресурси рядків Файл маніфесту AndroidManifest.xml

Кожна програма містить файл маніфесту **AndroidManifest.xml**. Даний файл визначає важливу інформацію про програму - назва, версію, іконки, які дозволи додаток використовує, реєструє всі використовувані класи activity, сервіси і т.д.

Файл маніфесту може виглядати так:

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

    <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/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>
```

Елементом кореневого рівня є вузол manifest. В даному випадку тільки визначається пакет додатку - **package="com.example.ozm.viewsapplication"**

Більшість налаштувань рівня додатку визначаються елементом application. Наприклад, через атрибут android:icon="@mipmap/ic_launcher" задається іконка програми, яка знаходиться в каталозі *res/mipmap-xxxx*

Також тут задається назва додатку, яка буде відображатися на мобільному пристрої в списку додатків і в заголовку: android:label="@string/app_name". В даному випадку вона зберігається в строкових ресурсах.

Вкладені елементи activityвизначають всі використовувані в додатку activity. В даному випадку видно, що в додатку є тільки одна activity - MainActivity.

Елемент intent-filterв MainActivity вказує, як дана activity буде використовуватися. Зокрема, за допомогою вузла action android:name="android.intent.action.MAIN", що дана activity буде вхідною точкою в додаток і не повинна отримувати будь-які дані ззовні.

Елемент category android:name="android.intent.category.LAUNCHER"вказує, що MainActivity представлятиме стартовий екран, який відображається під час запуску програми.

Визначення версії

За допомогою атрибутів елемента manifestможна визначити версію програми і його коду:

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

<!-- решта вміст -->
</manifest>
```

При бажанні ми також можемо визначити версію в ресурсах, а тут посилатися на ресурс.

Установка версії SDK

Для управління версією android sdk в файлі маніфесту визначається елемент <uses-sdk>. Він може використовувати такі атрибути:

- minSdkVersion: Мінімальна підтримувана версія SDK
- targetSdkVersion: Оптимальна версія

- maxSdkVersion: Максимальна версія
  Версія визначається номером API, наприклад, Jelly Beans 4.1 має версію 16:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.ozm.viewsapplication"
    android:versionName="1.0"
    android:versionCode="1" />
<uses-sdk android:minSdkVersion="16" android:targetSdkVersion="19" />

<!-- решта вміст -->
</manifest>
```

встановлення дозволів

Іноді потрібні додаткові дозволи на доступ до певних ресурсів, наприклад, до списку контактів, камери і т.д. Щоб додаток міг працювати зі списком контактів, в файлі маніфест необхідно встановити відповідні дозволи. Для установки дозволів застосовується елемент <uses-permission>:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.ozm.viewsapplication"/>
<uses-permission android:name="android.permission.READ_CONTACTS" />
<uses-permission                    android:name="android.permission.CAMERA"
android:maxSdkVersion="21" />

<!-- решта вміст -->
</manifest>
```

Атрибут android:name встановлює назву дозволу: в даному випадку на читання списку контактів і використання камери. Опціонально можна встановити максимальну версію sdk за допомогою атрибута android:maxSdkVersion, який приймає номер API.

Підтримка різних роздільних здатностей екрану

Світ пристроїв Android дуже сильно фрагментований, тут зустрічаються як гаджети з невеликим екраном, так і великі широкоформатні телевізори. І бувають випадки, коли треба обмежити використання програми для певних дозволів екранів. Для цього в файлі маніфесту визначається елемент <supports-screens>:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.ozm.viewsapplication"/>
    <supports-screens
    android:largeScreens="true"
    android:normalScreens="true"
    android:smallScreens="false"
    android:xlargeScreens="true" />

<!-- решта вміст -->
</manifest>
```

Даний елемент приймає чотири атрибути:

- android: largeScreens - екрани з діагоналлю від 4.5 до 10 "
- android: normalScreens - екрани з діагоналлю від 3 до 4.5 "
- android: smallScreens - екрани з діагоналлю менше 3 "
- android: xlargeScreens - екрани з діагоналлю більше 10 "

Якщо атрибут має значення true, то додаток буде підтримуватися відповідним розміром екрану

Заборона на зміну орієнтації

Додаток в залежності від положення гаджета може перебувати в портретній та альбомній орієнтації. Не завжди це буває зручно. Ми можемо зробити, щоб додаток незалежно від повороту гаджета використовувало тільки одну орієнтацію. Для цього в файлі маніфесту у необхідній activity треба встановити атрибут **android: screenOrientation**. Наприклад, заборонимо альбомну орієнтацію:

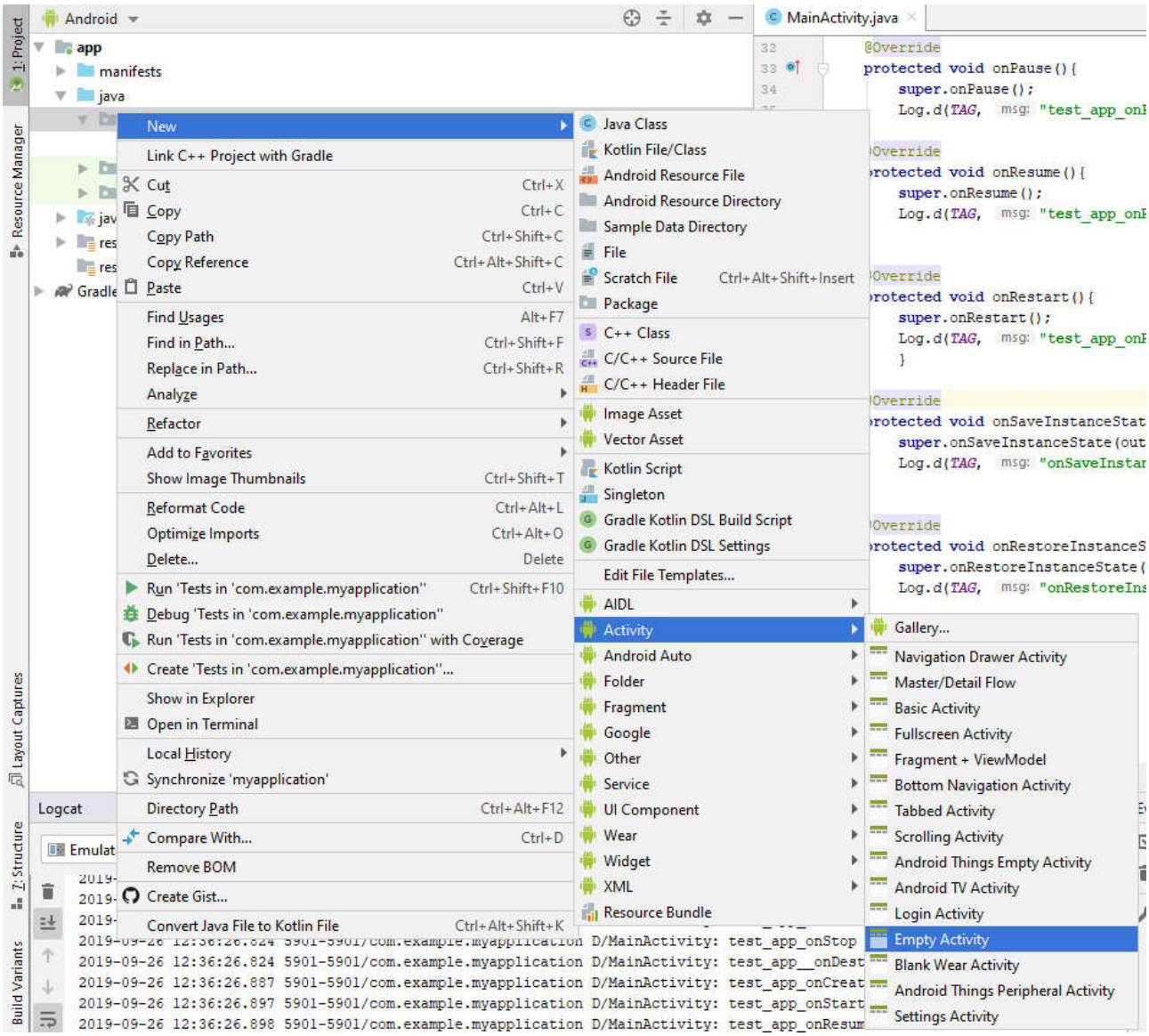
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.ozm.viewsapplication"/>

<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/AppTheme">
        <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>
```

Значення android:screenOrientation="portrait" вказує, що дана activity буде знаходитися тільки в портретній орієнтації. Якщо ж треба встановити тільки альбомну орієнтацію, тоді треба використовувати значенняandroid:screenOrientation="landscape"

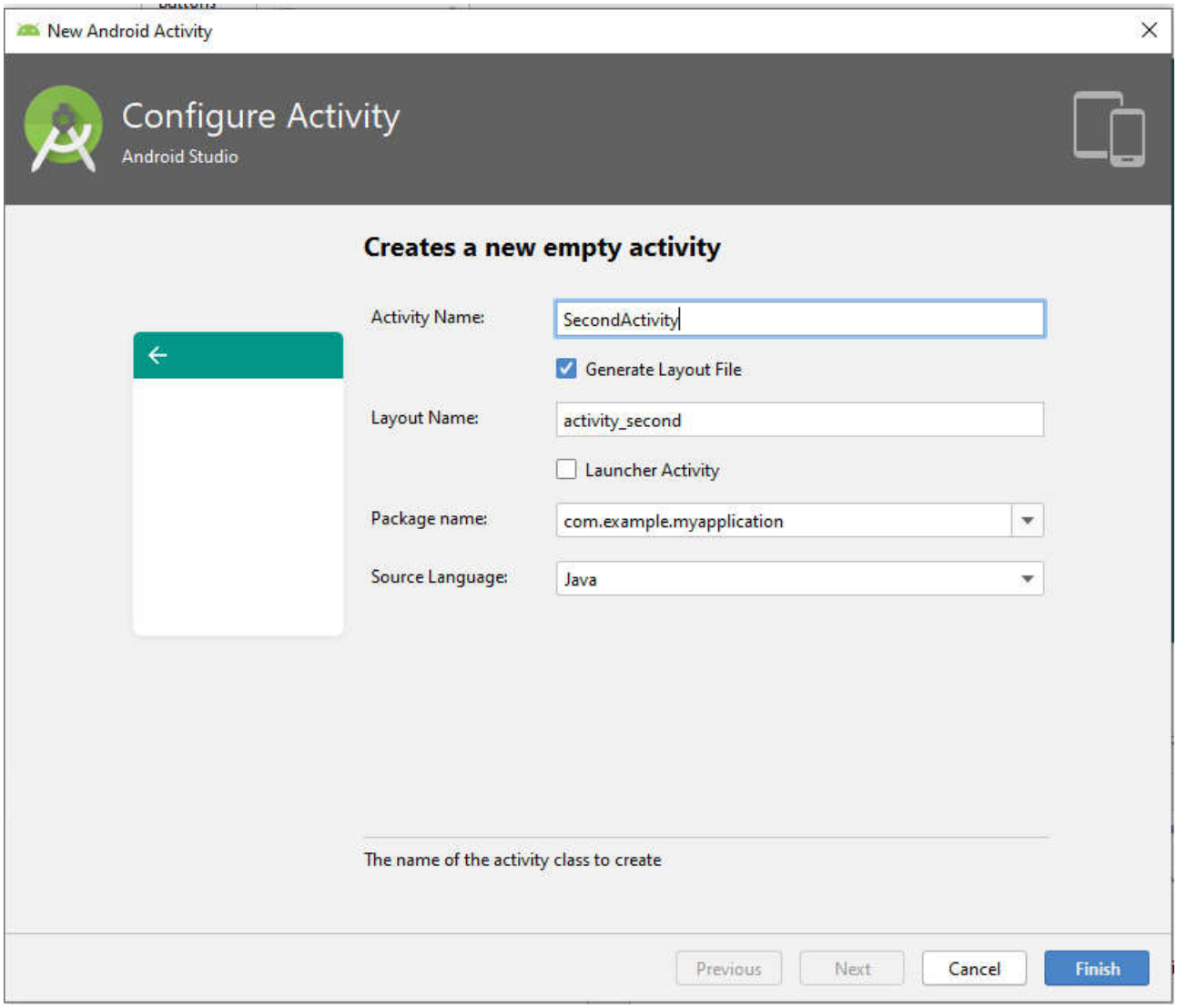
Intent і Intent-фільтри

Для взаємодії між різними об'єктами активності ключовим класом є **android.content.Intent**. Він представляє собою завдання, яке потрібно виконати додатку.

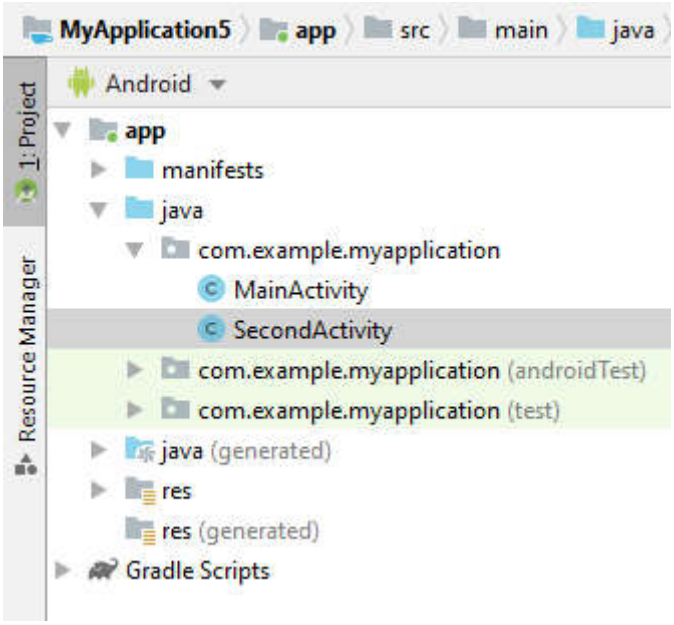
Для роботи с Intent додамо новий клас Activity. Для цього нажмем правою кнопкою мишки на папку, в якій знаходиться класс MainActivity, а потім у контекстному меню виділіть **New-> Activity-> Empty Activity** :



Новий клас Activity назвемо **SecondActivity**, а його файл розмітки інтерфейсу - **activity_second**:



І після цього в проект буде додано нова активність - друга активність:



Після цього в *файлі* маніфесту *AndroidManifest.xml ми* зможемо знайти наступні стрічки:

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

Дії всіх використовуваних класів повинні бути описані в файлі AndroidManifest.xml за допомогою елемента <activity>. Кожен подібний елемент містить як мінімум один атрибут android:name, який встановлює назву класу активності.

Однак по суті активность - це стандартні класи java, які наслідуються від класу **Activity** або його нащадків. Тому замість вбудованих шаблонів в Android Studio ми можемо додати звичайні класи, а потім їх наслідувати від класу Activity. Однак у цьому випадку потрібно буде вручну додати в файл маніфесту дані про активність.

При цьому для MainActivity в елементі **intent-filter** визначається інтент-**фільтр**. В ньому элемент дії значення "android.intent.action.MAIN" представляє головну точку входу в додаток. Тоді MainActivity залишається основною і запускається додатком по замовчуванню.

Для SecondActivity просто вказано, що вона в проекті, і ніяких інтуїтивних фільтрів для неї не задано.

Щоб з MainActivity запустити SecondActivity, необхідно викликати метод startActivity():

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

startActivity(intent);

В якості параметра в метод startActivity передається об'єкт Intent. Для свого створення Намір у конструкторі приймає два параметри: **action** (виконувану **дію** або задачу) і **дані** (передавані в задачу дані). В якості параметра action може виступати множина можливих дій. У даному випадку використовується **дія ACTION_MAIN**, яка задана константою "android.intent.action.MAIN".

Тепер розглянемо реалізацію переходу від однієї діяльності до іншої. Для цього в файлі **activity_main.xml** (тобто в інтерфейсі для MainActivity) визначимо кнопку:

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

</LinearLayout>
```

І визначимо для кнопки в класі **MainActivity** обробник натискання, по якому буде проводиться перехід до нової Activity:

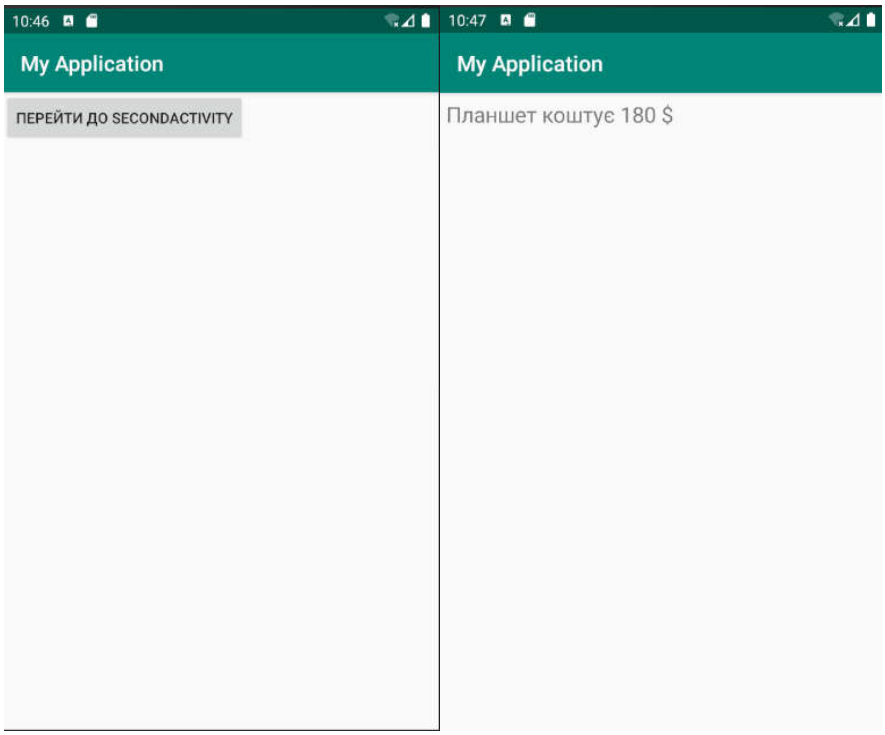
```
package com.example.ozm.viewsapplication;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
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 обробнику натискання буде запускатися SecondActivity.
Далі змінимо код **SecondActivity**:

```
package com.example.ozm.viewsapplication;

import android.support.v7.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(20);
        textView.setPadding(16, 16, 16, 16);
        textView.setText("Планшет коштує 180 $");
        setContentView(textView);
    }
}
```

Запустим додаток і перейдемо від одної Activity до іншої:



Intent-фільтри і дії

Тепер змінимо визначення SecondActivity в файлі AndroidManifest.xml:

```
<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>
<activity android:name=".SecondActivity">
    <intent-filter>
        <action android:name="com.example.ozm.SHOW_SECOND_ACTIVITY" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

Тут до SecondActivity додано намір-фільтр. В елементі дій вказує на дію, яку потрібно виконати для запуску активності. В даному випадку назва дії - "**com.example.ozm.SHOW_SECOND_ACTIVITY**" - може бути будь-якою. І також з допомогою елементу category визначена категорія, в даному випадку це **"android.intent.category.DEFAULT"**.

Тепер для запуску SecondActivity змінимо код MainActivity:

```
package com.example.ozm.viewsapplication;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends AppCompatActivity {
```

```
    public static final String ACTION ="com.example.ozm.
SHOW_SECOND_ACTIVITY";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    public void onClick(View view) {
        Intent intent = new Intent(ACTION);
        startActivity(intent);
    }
}
```

SecondActivity запускається за допомогою методу startActivity, але при створенні об'єкту Intent в його конструкторі передано назву дії, яка визначена в AndroidManifest.xml.

Передача даних між активністю. Серіалізація

Для передачі даних між двома активностями використовується об'єкт **Intent** . Через його метод **putExtra() можна** додати ключ і пов'язане з ним значення.

Наприклад, передача від поточної активності в SecondActivity рядка "Hello World" з ключом "hello":

```
// створення об’єкту Intent для запуску SecondActivity
Intent intent = new Intent(this, SecondActivity.class);
// передача об’єкту з ключем "hello" і значенням "Hello World"
intent.putExtra("hello", "Hello World");
// запуск SecondActivity
startActivity(intent);
```

Для **передачі** даних застосовується метод **putExtra()**, який в якості значення дозволяє передавати дані найпростіших типів - String, int, float, double, long, short, byte, char, масивів цих типів, або об'єкт інтерфейсу Serializable.

Щоб отримати відправлені дані при завантаженні SecondActivity, можна скористатися методом **get()**, в який передається ключ об'єкта:

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

Залежно від типу відправлених даних при їх отриманні ми можемо використовувати ряд методів об'єкта Bundle. Всі вони в якості параметра приймають ключ об'єкта. Основні з них:

- **get()**: універсальний метод, який повертає значення типу Object. Відповідний потік отримання даного значення необхідно перетворити на потрібний тип
- **getString()**: повертає об'єкт типу String
- **getInt()**: повертає значення типу int
- **getByte()**: повертає значення типу байт
- **getChar()**: повертає значення типу char
- **getShort()**: віддає значення типу short
- **getLong()**: віддає значення типу long
- **getFloat()**: повердає значення типу float
- **getDouble()**: віддає значення типу double
- **getBoolean()**: повертає значення типу boolean

- **getCharArray()**: повертає **масив** об'єктів char
- **getIntArray()**: **повертає масив** об'єктів int
- **getFloatArray()**: повертає масив об’єктів float
- **getSerializable()**: повертає об'єкт інтерфейсу. Serializable

Нехай в нас в проекті буде визначено дві активності: MainActivity and SecondActivity. В коді SecondActivity визначимо отримання даних:

```
package com.example.ozm.viewsapplication;

import android.support.v7.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(20);
        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 price = arguments.getInt("price");
            textView.setText("Name: " + name + "\nCompany: " + company +
                    "\nPrice: " + price);
        }
        setContentView(textView);
    }
}
```

У даному випадку в SecondActivity отримуємо всі дані з об'єкта Bundle і виводять їх у текстове поле TextView. Передбачається, що в дану активність буде передаватися три елемента - два рядки з ключем ім'я та компанія і число з ціною.

Тепер визначимо передачу в SecondActivity дані. Наприклад, визначимо для MainActivity наступний інтерфейс в файлі **activity_main.xml** :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="5dp">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="20dp"
        android:text="Name:" />
    <EditText
        android:id="@+id/name"
        android:layout_width="match_parent"
        android:layout_height="40dp"/>
    <TextView
        android:layout_width="match_parent"
```

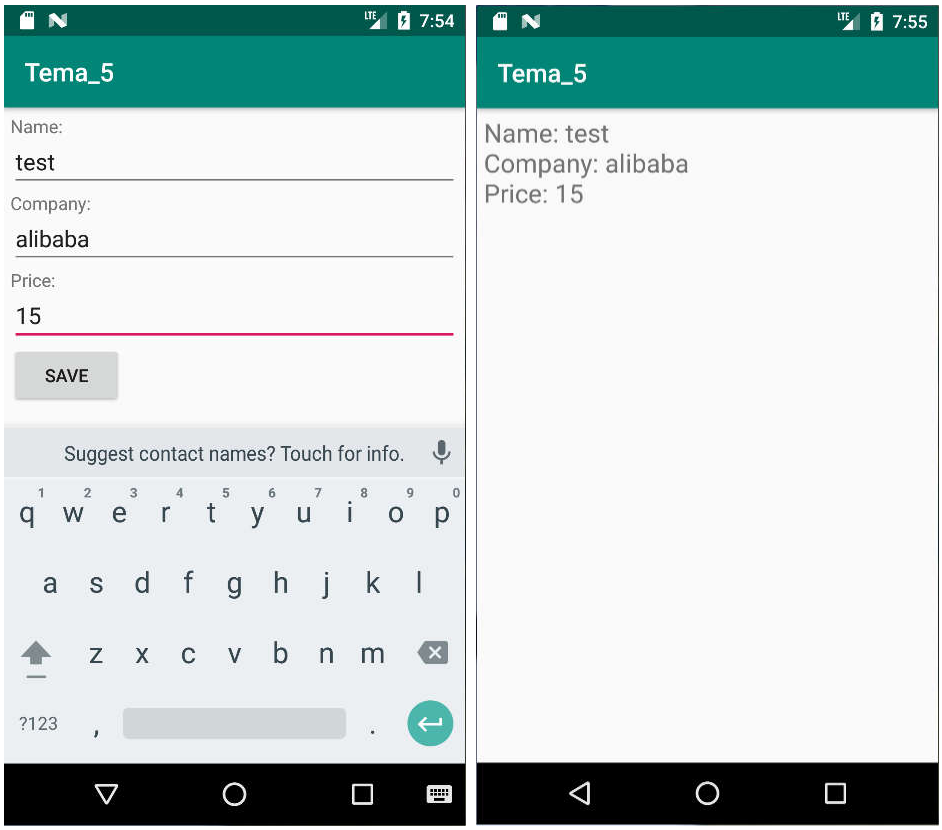
```
        android:layout_height="20dp"
        android:text="Company:" />
    <EditText
        android:id="@+id/company"
        android:layout_width="match_parent"
        android:layout_height="40dp" />
    <TextView
        android:layout_width="match_parent"
        android:layout_height="20dp"
        android:text="Price:" />
    <EditText
        android:id="@+id/price"
        android:layout_width="match_parent"
        android:layout_height="40dp" />
    <Button
        android:id="@+id/btn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onClick"
        android:text="Save"/>

</LinearLayout>
```

Тут визначено три текстових поля для введення даних і кнопка.
В класі MainActivity визначено наступний вміст:

```
package com.example.ozm.viewsapplication;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
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) {

        final EditText nameText = findViewById(R.id.name);
        final EditText companyText = findViewById(R.id.company);
        final EditText priceText = findViewById(R.id.price);

        String name = nameText.getText().toString();
        String company = companyText.getText().toString();
        int price = Integer.parseInt(priceText.getText().toString());

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

У процесі натискання кнопки ми отримуємо введені в текстові поля EditText дані і передаємо їх в об'єкт Intent з допомогою методу putExtra(). Затем запускаємо SecondActivity.

В результаті після натискання кнопки запускається SecondActivity, яка отримує деякі введені в текстові поля дані.



Передача складних об'єктів

У прикладі вище передавались прості дані - числа, стрічки. Але також ми можемо передавати більш складні дані. В цьому випадку використовується механізм серіалізації.

Наприклад, нехай у нас в проекті буде визначений клас Product:

```
package com.example.ozm.viewsapplication;

import java.io.Serializable;

public class Product implements Serializable {

    private String name;
    private String company;
    private int price;

    public Product(String name, String company, int price){
        this.name = name;
        this.company = company;
        this.price = price;
    }

    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 getPrice() {
        return price;
    }

    public void setPrice(int price) {
        this.price = price;
    }
}
```

Слід відмітити, що даний клас реалізує інтерфейс **Serializable**. Тепер змінимо код MainActivity:

```
package com.example.ozm.viewsapplication;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
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) {

        final EditText nameText = findViewById(R.id.name);
        final EditText companyText = findViewById(R.id.company);
        final EditText priceText = findViewById(R.id.price);

        String name = nameText.getText().toString();
        String company = companyText.getText().toString();
        int price = Integer.parseInt(priceText.getText().toString());

        Product product = new Product(name, company, price);

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

Тепер замість трьох різних даних передається один об’єкт Product. В якості ключа використовується результат методу Product.class.getSimpleName(), який по суті повертає назву класу.

І змінимо клас SecondActivity:

```
package com.example.ozm.viewsapplication;

import android.support.v7.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(20);
        textView.setPadding(16, 16, 16, 16);

        Bundle arguments = getIntent().getExtras();
        final Product product;
        if(arguments!=null){
            product                          =                          (Product)
arguments.getSerializable(Product.class.getSimpleName());

            textView.setText("Name: " + product.getName() + "\nCompany: "
+ product.getCompany() +
                    "\nPrice: " + String.valueOf(product.getPrice()));
        }
        setContentView(textView);
    }
}
```

Для отримання даних застосовується метод getSerializable(), оскільки продукт класу реалізує інтерфейс Serializable. Таким чином, ми можемо передавати єдиний об'єкт замість набору різнорідних даних.

Parcelable

Можливість серізації об'єктів надана безпосередньо в інфраструктурі Java. Однак Android також надає інтерфейс **Parcelable**, який по суті також дозволяє серіалізовувати об'єкти, як і Serializable, але більш оптимізований для Android. І подібні об'єкти Parcelable також можна передавати між двома видами діяльності або використовувати якийсь інший спосіб.

Наприклад, в минулому параграфі дані передаються між діяльностями у вигляді об'єктів. Продукт, який використовував серілізацію. Тепер нехай клас Product застосовує інтерфейс Parcelable:

```
package com.example.ozm.viewsapplication;

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

public class Product implements Parcelable {

    private String name;
    private String company;
    private int price;

    public static final Creator<Product> CREATOR = new Creator<Product>() {
        @Override
```

```
    public Product createFromParcel(Parcel source) {
        String name = source.readString();
        String company = source.readString();
        int price = source.readInt();
        return new Product(name, company, price);
    }

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

public Product(String name, String company, int price){
    this.name = name;
    this.company = company;
    this.price = price;
}

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 getPrice() {
    return price;
}

public void setPrice(int price) {
    this.price = price;
}

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

@Override
public void writeToParcel(Parcel parcel, int flags) {
```

```
        parcel.writeString(name);
        parcel.writeString(company);
        parcel.writeInt(price);
    }
}
```

Інтерфейс **android.os.Parcelable** передбачає реалізацію двох методів: describeContents() і writeToParcel(). Перший метод описує контент і повертає деяке числове значення. Другий метод пише в об'єкт Parcel contents of the object Продукт.

Для запису даних об'єкта в Parcel використовується ряд методів, кожен з яких призначений для певного типу даних. Основні методи:

- **writeString ()**
- **writeInt ()**
- **writefloat ()**
- **writeDouble ()**
- **writeByte ()**
- **writeLong ()**
- **writeIntArray ()**
- **writeValue()** (записує об'єкт типу Object)
- **writeParcelable()** (записує об'єкт типу Parcelable)

Крім того, об'єкт Parcelable повинен мати статичне поле CREATOR, який представляє об'єкт Creator<Product>. Причому цей об'єкт реалізує два методи. Вони потрібні для створення їх попередніх серіалізованих даних вихідних об'єктів типу продукту.

Так, метод newArray() створює масивний об'єкт продукту.

Метод **createFromParcel** створює з Parcel новий об'єкт типу продукту. Тож це метод протилежний по дії методу writeToParcel. Для отримання даних з посилок застосовуються методи типу readString(), readInt(), readParcelable() і так далі - для читання певних типів даних.

Примітно, що дані в createFromParcel зчитуються з об'єкта Parcel саме в тому порядку, в якому вони додаються в цей об'єкт у методі writeToParcel.

Припустимо у діяльності, яка називається **SecondActivity** буде нами отримувати об'єкт Product:

```
package com.example.ozm.viewsapplication;

import android.support.v7.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(20);
        textView.setPadding(16, 16, 16, 16);

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

```
        final Product product;
        if(arguments!=null){
          product = arguments.getParcelable(Product.class.getSimpleName());

          textView.setText("Name: " + product.getName() + "\nCompany: " +
product.getCompany() +
              "\nPrice: " + String.valueOf(product.getPrice()));
        }
        setContentView(textView);
      }
    }
```

Для отримання об'єкту Parcelable, переданого в діяльність, застосовується метод getParcelable(). Причому ніякого приведення типів не потрібно.

Для тестування передачі Parcelable визначається в файлі **activity_main.xml** простий інтерфейс для MainActivity:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical"
  android:padding="5dp">

  <TextView
    android:layout_width="match_parent"
    android:layout_height="20dp"
    android:text="Name:" />
  <EditText
    android:id="@+id/name"
    android:layout_width="match_parent"
    android:layout_height="40dp"/>
  <TextView
    android:layout_width="match_parent"
    android:layout_height="20dp"
    android:text="Company:" />
  <EditText
    android:id="@+id/company"
    android:layout_width="match_parent"
    android:layout_height="40dp" />
  <TextView
    android:layout_width="match_parent"
    android:layout_height="20dp"
    android:text="Price:" />
  <EditText
    android:id="@+id/price"
    android:layout_width="match_parent"
    android:layout_height="40dp" />
  <Button
    android:id="@+id/btn"
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="onClick"
    android:text="Save"/>

</LinearLayout>
```

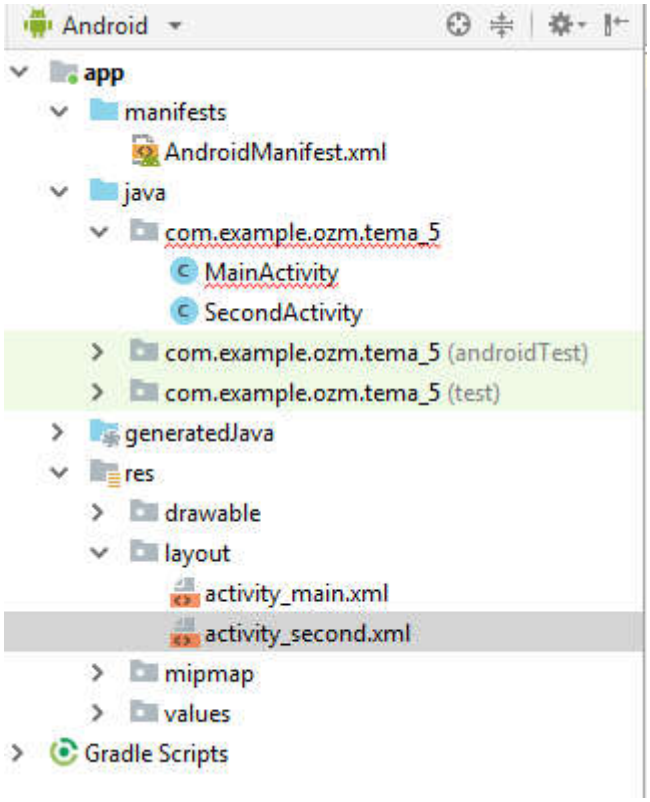
А в коді **MainActivity** визначимо передачу даних в SecondActivity:

```
package com.example.ozm.viewsapplication;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
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) {

            final EditText nameText = findViewById(R.id.name);
            final EditText companyText = findViewById(R.id.company);
            final EditText priceText = findViewById(R.id.price);

            String name = nameText.getText().toString();
            String company = companyText.getText().toString();
            int price = Integer.parseInt(priceText.getText().toString());

            Product product = new Product(name, company, price);

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

Отримання результату з Activity

В минулому параграфі було розглянуто як викликати нову activity і передавати їй деякі дані. Але ми можемо не тільки передавати дані activity, але й очікувати від неї деякого результату роботи.

Наприклад, нехай у нас в проекті будуть дві активності: MainActivity and SecondActivity. А для кожної activity є свій інтерфейс інтерфейсу: activity_main.xml і activity_second.xml.



В минулому параграфі ми викликали нову активність за допомогою методу startActivity(). Для отримання результату activity необхідно використовувати метод **startActivityForResult (Intent Intent, int requestCode)** . Цей метод приймає два параметри: **Intent** передає в activity, що запускається дані, а **другий** параметр **requestCode вказує** на цілочисленний код запиту. Розглянемо його використання на прикладі.

Так, визначимо в файлі **activity_main.xml наступний** інтерфейс:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:orientation="vertical">
    <TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Вкажіть вік"
        android:textSize="22dp"/>
    <EditText
        android:id="@+id/ageBox"
        android:inputType="number"
        android:layout_width="match_parent"
```

```
            android:layout_height="wrap_content" />
        <Button
            android:id="@+id/navButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Надіслати"
            android:onClick="onClick" />

    </LinearLayout>
```

Для введення даних тут визначено елемент EditText, а для відправлення - кнопка. Определим в классе **MainActivity** запуск **другої** activity:

```
package com.example.ozm.tema_5;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
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";
    private static  final int REQUEST_ACCESS_TYPE=1;

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

        textView = (TextView) findViewById(R.id.textView);
    }
    public void onClick(View view) {

        // отримуємо введений вік
        EditText ageBox = (EditText) findViewById(R.id.ageBox);
        String age = ageBox.getText().toString();

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

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent
data){
        if(requestCode==REQUEST_ACCESS_TYPE){
            if(resultCode==RESULT_OK){
                String                    accessMessage                    =
data.getStringExtra(ACCESS_MESSAGE);
                textView.setText(accessMessage);
            }
            else{
                textView.setText("Помилка доступу");
            }
        }
        else{
            super.onActivityResult(requestCode, resultCode, data);
        }
    }
```

```
}
```

У процесі натискання кнопки onClick()отримуємо введений в текстовий вік вік, додаємо його в об'єкт Intent з ключем AGE_KEY і запускаємо SecondActivity за допомогою методу startActivityForResult. Причому числовий код запиту представляє константу REQUEST_ACCESS_TYPE. Тут не так важливо, який номер передавати як результат, але використовуючи це число, ми потім можемо виконати обробку отриманої відповіді від SecondActivity, особливо якщо в різних ситуаціях застосовується кілька кількісних кодів запиту.

Для отримання та обробки результату, отриманого від SecondActivity, необхідно перевизначити метод **onActivityResult** . Цей метод приймає три параметри:

- requestCode: числовий код запиту, який був відправлений у другий параметр у startActivityForResult
- resultCode: числовий код результату. В якості результату, як правило, застосовуються вбудовані константи RESULT_OK та RESULT_CANCELED.
- data: відправлені дані з SecondActivity в MainActivity

У цьому випадку в методі onActivityResult()виводяться отримані дані в елемент TextView.

Далі перейдем до SecondActivity і визначимо в файлі **activity_second.xml** набір кнопок:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/activity_second"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">
    <TextView
        android:id="@+id/ageView"
        android:textSize="22sp"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
    <Button
        android:id="@+id/button1"
        android:text="Відкрити доступ"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="onButton1Click"/>
    <Button
        android:id="@+id/button2"
        android:text="Відмінити доступ"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="onButton2Click" />
    <Button
        android:id="@+id/button3"
        android:text="Вік недійсний"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="onButton3Click" />
    <Button
        android:id="@+id/cancel"
        android:text="Відміна"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="onCancelClick" />
</LinearLayout>
```

А в класі **SecondActivity визначимо** обробники для цих кнопок:

```
package com.example.ozm.tema_5;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
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 = (TextView) findViewById(R.id.ageView);
            String age = extras.getString(MainActivity.AGE_KEY);
            ageView.setText("Вік: " +  age);
        }
    }
    public void onCancelClick(View v) {
        setResult(RESULT_CANCELED);
        finish();
    }
    public void onButton1Click(View v) {
        sendMessage("Доступ дозволено");
    }
    public void onButton2Click(View v) {
        sendMessage("Доступ заборонено");
    }
    public void onButton3Click(View v) {
        sendMessage("Недопустимий вік");
    }
    private void sendMessage(String message){

        Intent data = new Intent();
        data.putExtra(MainActivity.ACCESS_MESSAGE, message);
        setResult(RESULT_OK, data);
        finish();
    }
}
```

Три кнопки викликають метод sendMessage(), в який передаються відправлений ответ. Это и будет то сообщение, которая получит MainActivity в методе onActivityResult.

Для повернення результату необхідно викликати метод **setResult ()** , в який передаються два параметри:

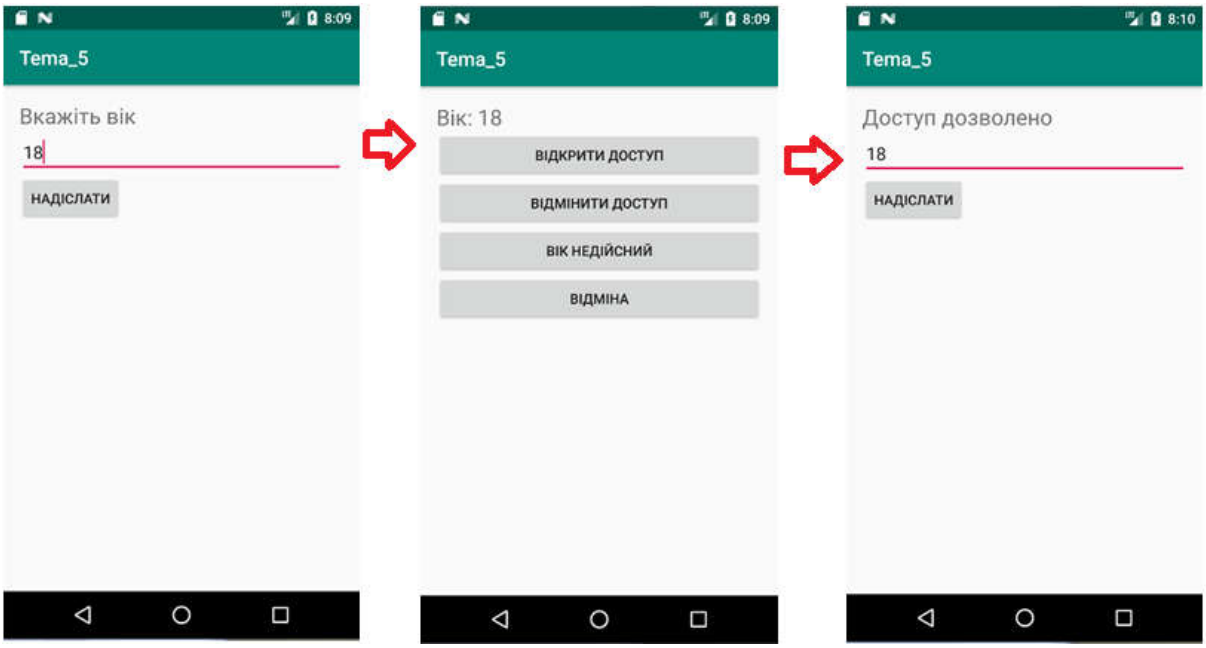
- числовий код результату
- отправляемые дані

Після виклику методу потрібно setResult()викликати метод **finish** , який знищує поточну активність.

Одна кнопка викликає обробника onCancelClick(), в якому передано в setResult лише код результату - RESULT_CANCELED.

Тож, умовно говоря, ми отримуємо в SecondActivity, введенный в MainActivity, і при натисканні певного кнопки повертаємо деякий результат у вигляді повідомлення.

В залежності від натискання кнопки на SecondActivity ми будемо отримувати різні результати в MainActivity:



Взаємодія між Activity

У минулих параграфах ми розглянули життєвий цикл activity і запуск нових activity за допомогою об'єкта Intent. Тепер розглянемо деякі особливості взаємодії між activity в одному додатку. Припустимо, у нас є три activity: MainActivity, SecondActivity і ThirdActivity.

За допомогою Intent, наприклад, після натискання кнопки MainActivity запускає SecondActivity:

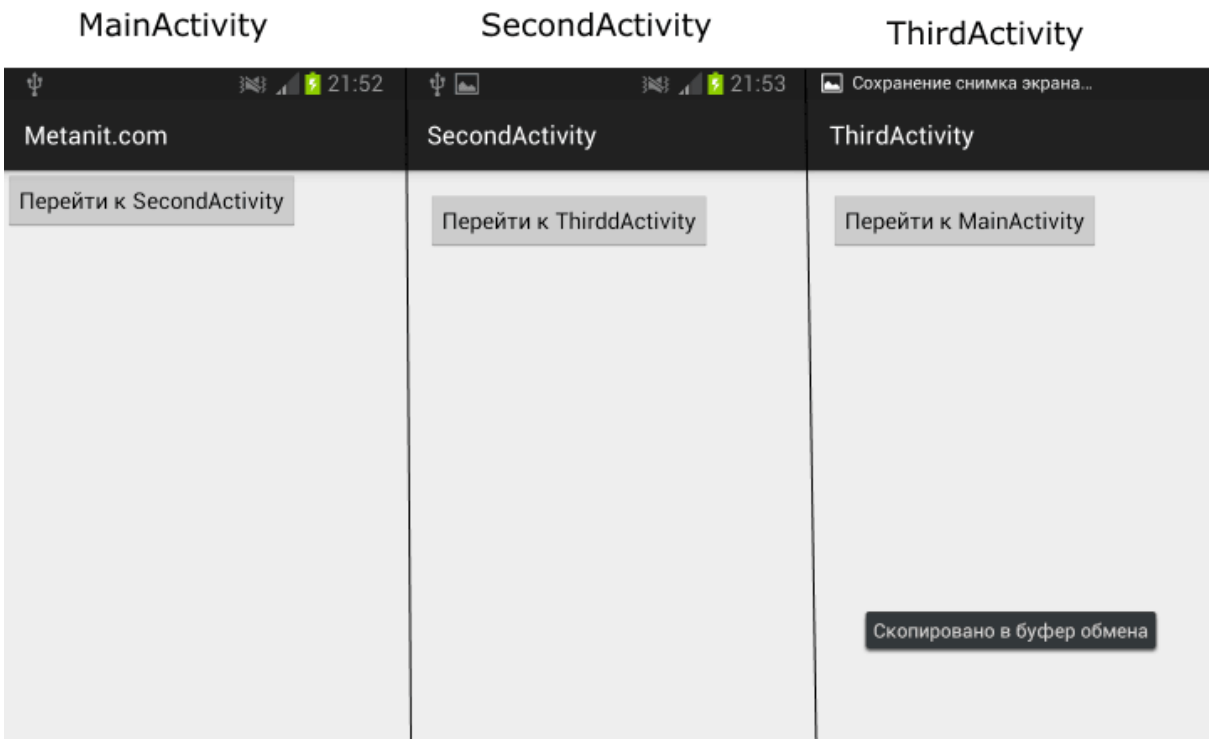
```
Intent intent = new Intent(this, SecondActivity.class);
startActivity(intent);
```

На SecondActivity теж є кнопка, яка запускає ThirdActivity:

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

На ThirdActivity також є кнопка, яка повертається до першої activity - MainActivity:

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



Якщо ми послідовно запустимо всі activity: з головною MainActivity запустимо SecondActivity, з SecondActivity - ThirdActivity, то в результаті у нас складеться наступний стек activity:

ThirdActivity;
SecondActivity;
MainActivity.

Якщо після цього з ThirdActivity ми захочемо повернутися до MainActivity, то метод startActivity () запустить новий об'єкт MainActivity (а не повернеться до вже існуючої), і стек вже буде виглядати наступним чином:

MainActivity;
ThirdActivity;
SecondActivity;
MainActivity.

Тобто у нас будуть дві незалежні копії MainActivity. Такий стан небажаний, якщо ми просто хочемо перейти до існуючої. І цей момент треба враховувати. Якщо ми натиснемо на кнопку Back (Назад), то ми зможемо перейти до попередньої activity в стеці.

Щоб вийти з цієї ситуації, ми можемо використовувати прапорець **Intent.FLAG_ACTIVITY_REORDER_TO_FRONT**:

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

У цьому випадку після переходу з ThirdActivity до MainActivity стек буде виглядати наступним чином:

MainActivity;
ThirdActivity;
SecondActivity.

Якщо ж нам просто треба перейти з ThirdActivity до MainActivity, як якщо б ми перейшли назад за допомогою кнопки Back, то ми можемо використовувати прапорці **Intent.FLAG_ACTIVITY_CLEAR_TOP** і **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);
```

У цьому випадку після переходу з ThirdActivity до MainActivity стек буде повністю очищений, і там залишиться одна MainActivity.