**4.2 Activity**

**4.2.1 Поняття Activity**

Ключовим компонентом створення візуального інтерфейсу в додатку є activity (активність). Часто activity асоціюється з окремим екраном або вікном додатку, а перемикання між вікнами буде відбуватися як переміщення від однієї activity до іншої. Додаток може включати одну або декілька activity (рис. 4.1).



Рисунок 4.1 Представлення додатку як набору activity

Всі об’єкти activity є об’єктами класу *android.app.Activity*, який містить базову функціональність для всіх activity. Як правило activity успадковуються від класу *AppCompatActivity* який, успадковує базовий клас *Activity*:

public class MainActivity extends AppCompatActivity {

// вміст класу

}

**4.2.2 Життєвий цикл Activity**

Всі додатки Android мають строго визначений життєвий цикл. При запуску користувачем додатку операційна система надає цьому додатку високий пріоритет.

Кожна програма запускається у вигляді окремого процесу, що дозволяє системі надавати одним процесам вищий пріоритет, на відміну від інших. Завдяки цьому, наприклад, при роботі з одними додатками не блокуються вхідні дзвінки. Після припинення роботи додатку, система звільняє всі пов’язані з ним ресурси, переводить додаток у розряд низькопріоритетного і закриває його.

Всі об’єкти activity, які є в додатку, управляються системою у вигляді стека activity, який називається back stack. При запуску нової activity вона поміщається на вершину стеку і виводиться на екран пристрою, поки не з’явиться нова activity. Коли поточна activity закінчує свою роботу (наприклад, користувач закриває програму), то вона видаляється із стеку, і відновлює роботу та 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 так, як це зроблено на рис 4.2.

Тут **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 відбувається початкове налаштування activity. Зокрема, створюються об’єкти візуального інтерфейсу. Цей метод отримує об’єкт Bundle, який містить попередній стану activity, якщо він булв збережений. Якщо activity заново створюється, то даний об’єкт має значення null. Якщо ж activity вже раніше була створена, але перебувала в призупиненому стані, то bundle містить пов’язану з activity інформацію.

В методі **onStart()** відбувається підготовка до виведення 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.

Після завершення методу onStart() викликається метод **onRestoreInstanceState(),** який призначений для відновлення збереженого стану 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.



Рисунок 4.2 Життєвий цикл activity

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

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., для того, 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().

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

В методі **onStop()** activity переходить в стан Stopped. Тут 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 в стані 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 повністю завершила свою роботу, наприклад, внаслідок закриття програми, то викликається метод onDestroy().

Завершується робота activity викликом методу **onDestroy()**, який відбувається або якщо система вирішить завершити роботу 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 представити наступною схемою

(рис 4.3):



Рисунок 4.3 Діаграма переходів між станами Activity

Розглянемо декілька типових ситуацій. Якщо ми працюємо з Activity і потім перемикається на іншу програму, або натискаємо на кнопку Home, то у Activity викликається наступний ланцюжок методів: *onPause()->onStop()*. Activity переходить в стан Stopped. Якщо користувач вирішить повернутися до Activity, то викликається наступний ланцюжок методів: *onRestart()->onStart()->onResume().*

Інша типова ситуація, коли користувач натискає на кнопку Back (Назад), то викликається наступний ланцюжок *onPause()->onStop()->onDestroy()*. В результаті Activity знищується. Якщо ми раптом захочемо повернутися до Activity через диспетчер задач або заново відкривши додаток, то activity буде заново перестворено шляхом виклику методів *onCreate()->onStart()-> onResume().*

**4.2.3 Управління життєвим циклом Activity**

Ми можемо керувати подіями життєвого циклу activity, перевизначивши відповідні методи. Наприклад:

public class MainActivity extends AppCompatActivity {

private final static String TAG="MainActivity";

@Override

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. Обробка

зводиться до виклику методу *Log.d()*, в який передається TAG – випадкове значення рядка і рядок, який виводиться в консолі logcat внизу середовища Android Studio у вікні Android Monitor. Якщо ця консоль за замовчуванням прихована, то її можна відкрити через пункт меню *View->Tool Windows->Android Monitor*. Під час запуску програми ми

зможемо побачити у вікні logcat відлагоджувальну інформацію (рис. 4.4).



Рисунок 4.4 Відлагоджувальна інормація у вікні logcat середовища Android Studio

**4.2.4 Запуск Activity**

При організації взаємодії між різними об’єктами activity ключовим класом є *android.content.Intent*. Він представляє собою задачу, яку повинен виконати додаток. Нехай ми добавили в проект нову пусту activity з ім’ям *SecondActivity*. Після цього в файлі маніфесту *AndroidManifest.xml* ми зможемо знайти відповідний рядок:

<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>

Тут для MainActivity в елементі intent-filter визначено інтент-фільтр, в якому елемент <action android:name="android.intent.action.MAIN" представляє головну точку входу в додаток. Тобто MainActivity залишається основною і запускається додатком за замовчуванням. Для SecondActivity просто вказано, що вона є в проекті, і ніяких intent- фільтрів для неї не задано.

Для того, щоб з MainActivity запустити SecondActivity, слід викликати метод

*startActivity*():

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

startActivity(intent);

Як параметр в метод *startActivity*() передається об’єкт Intent, який приймає два параметри: action (дію або завдання) і data (передані в задачу дані). В якості параметра action може виступати багато можливих дій. В даному випадку використовується дія ACTION\_MAIN, якf задається константою "android.intent.action.MAIN".

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

<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" />

Для запуску SecondActivity із MainActivity при натисканні на цю кнопку необхідно написати наступний код:

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);

}

}

**4.3 Передача даних між Activity**

Для передавання даних між двома Activity використовується об’єкт Intent. Через його метод *putExtra()* можна передати ключ і пов’язане з ним значення.

Наприклад, передача з поточної activity в SecondActivity стрічки "Hello World" з ключем "hello" може виглядати так:

Intent intent = new Intent(this, SecondActivity.class); intent.putExtra("hello", "Hello World"); startActivity(intent);

Для передавання даних застосовується метод putExtra(), який дозволяє передати дані найпростіших типів – string, int, float, double, long, short, byte, char, масиви 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();

Залежно від типу даних, що передаються, при 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. Відповідно поле отримання дане значення необхідно перетворити до потрібного типу;

 *getString*(): повертає об’єкт типу string;

 *getInt*(): повертає значення типу int;

 *getByte*(): повертає значення типу byte;

 *getChar*(): повертає значення типу char;

 *getShort*(): повертає значення типу short;

 *getLong*(): повертає значення типу long;

 *getFloat*(): повертає значення типу float;

 *getDouble*(): повертає 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.

Для передачі даних складних типів використовується механізм серіалізації. Нехай в проекті є клас *Product*, який реалізує інтерфейс *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;

}

…

}

Щоб передати його в іншу activity слід написати код:

Product product=new Product(name, company, price); Intent intent=new Intent(this, SecondActivity.class); intent.putExtra(Product.class.getSimpleName(), product); startActivity(intent);

В якості ключа тут використовується результат виклику методу *Product.class.getSimpleName()*, який по суті повертає назву класу. Щоб отримати об’єкт типу *Product* слід написати:

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);

}

}