8 Фрагменти

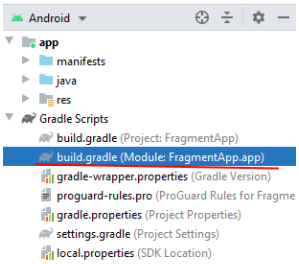
8.1 Що таке фрагмент? Цілі і задачі фрагментів

Організація програми на основі декількох Activity не завжди може бути оптимальною. Світ ОС Android дуже фрагментований і складається з багатьох пристроїв. І якщо для мобільних апаратів з невеликими екранами взаємодія між різними activity виглядає досить непогано, то на великих екранах – планшетах, телевізорах вікна activity виглядали б не дуже через великий розмір екрану. Саме тому і виникла концепція фрагментів.

Фрагмент представляє шматочок візуального інтерфейсу програми, який можна використовувати повторно і багаторазово. У фрагмента може бути власний файл layout, фрагменти мають свій власний життєвий цикл. Фрагмент існує в контексті діяльності і має свій життєвий цикл, поза діяльністю відокремлено він існувати не може. Кожна діяльність може мати кілька фрагментів.

8.2 Створення фрагментів

Для початку роботи з фрагментами створимо новий проект із порожньою MainActivity. І спочатку ми створимо перший фрагмент. Але варто відразу зазначити, що не весь функціонал фрагментів за замовчуванням може бути доступний в проекті, так як він знаходиться в окремій бібліотеці - **AndroidX Fragment library** . Спочатку необхідно підключити до проекту цю бібліотеку у файлі **build.gradle** .



Знайдемо в ньому секцію **dependencies** , яка виглядає за умовчанням приблизно так:

```
1  dependencies {
2
3      implementation 'androidx.appcompat:appcompat:1.3.1'
4      implementation 'com.google.android.material:material:1.4.0'
5      implementation 'androidx.constraintlayout:constraintlayout:2.1.0'
6      testImplementation 'junit:junit:4.+'
```

```
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
}
```

На її початок додамо рядок

```
implementation "androidx.fragment:fragment:1.3.6"
```

Тобто в моєму випадку вийде

```
dependencies {

    implementation "androidx.fragment:fragment:1.3.6"

    implementation 'androidx.appcompat:appcompat:1.3.1'
    implementation 'com.google.android.material:material:1.4.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.0'
    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
}
```

Потім клацніть посилання **«Синхронізувати зараз»** , що з’явиться .

Фактично фрагмент – це звичайний клас Java, який успадковується від класу **Fragment** . Однак, як і клас Activity, фрагмент може використовувати xml-файли layout для визначення графічного інтерфейсу. І таким чином, ми можемо додати окремо клас Java, який представляє фрагмент, і файл xml для зберігання розмітки інтерфейсу, який буде використовувати фрагмент.

Отже, додамо в папку **res/layout** новий файл **fragment_content.xml** і визначимо наступний код:

```
<?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/updateButton"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Обновить"
        app:layout_constraintBottom_toTopOf="@+id/dateTextView"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
```

```
17
18      <TextView
19          android:id="@+id/dateTextView"
20          android:layout_width="0dp"
21          android:layout_height="wrap_content"
22          android:text="Hello from Fragment"
23          android:textSize="28sp"
24          app:layout_constraintLeft_toLeftOf="parent"
25          app:layout_constraintRight_toRightOf="parent"
26          app:layout_constraintTop_toBottomOf="@+id/updateButton" />
27
28  </androidx.constraintlayout.widget.ConstraintLayout>
```

Фрагменти містять самі елементи управління, як і activity. Зокрема, тут визначено кнопку та текстове поле, які становитимуть інтерфейс фрагмента.

Тепер створимо сам клас фрагмента. Для цього додамо в одну папку з **MainActivity** новий клас. Для цього натиснемо на папку правою кнопкою миші та виберемо в меню **New -> Java Class** . Назвемо новий клас **ContentFragment** і визначимо у нього такий зміст:

```
1   package com.example.fragmentapp;
2
3   import androidx.fragment.app.Fragment;
4
5   public class ContentFragment extends Fragment {
6
7       public ContentFragment(){
8           super(R.layout.fragment_content);
9       }
10  }
```

Клас фрагмента повинен успадковуватися від класу **Fragment** .

Щоб вказати, що фрагмент використовувати певний xml-файл layout, ідентифікатор ресурсу layout передається у виклик конструктора батьківського класу (тобто класу Fragment).

8.2.1 Додавання фрагмента до Activity

Для використання фрагмента додамо його в **MainActivity** . Для цього змінимо файл **activity_main.xml** , який визначає інтерфейс для MainActivity:

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <androidx.fragment.app.FragmentContainerView
3       xmlns:android="http://schemas.android.com/apk/res/android"
4       android:id="@+id/fragment_container_view"
5       android:layout_width="match_parent"
6       android:layout_height="match_parent"
```

```
7       android:name="com.example.fragmentapp.ContentFragment" />
```

Для додавання фрагмента застосовується елемент **FragmentContainerView** . По суті, FragmentContainerView представляє об'єкт View, який розширює клас **FrameLayout** і призначений спеціально для роботи з фрагментами. Власне, крім фрагментів, він більше нічого утримувати не може.

Його атрибут **android:name** вказує на назву класу фрагмента, який буде використовуватися. У моєму випадку повне ім'я класу фрагмента з облікових записів **com.example.fragmentapp.ContentFragment** .

Код класу **MainActivity** залишається тим самим, що і при створенні проекту:

```
1   package com.example.fragmentapp;
2
3   import androidx.appcompat.app.AppCompatActivity;
4
5   import android.os.Bundle;
6
7   public class MainActivity extends AppCompatActivity {
8
9       @Override
10      protected void onCreate(Bundle savedInstanceState) {
11          super.onCreate(savedInstanceState);
12          setContentView(R.layout.activity_main);
13      }
14  }
```

Якщо ми запустимо додаток, то ми побачимо фактично той самий інтерфейс, який ми могли б зробити і через activity, тільки в даному випадку інтерфейс буде визначений у фрагменті.

Android Studio представляє готовий шаблон для додавання фрагмента. Власне скористаємось цим способом.

Для цього натиснемо на папку, де знаходиться клас **MainActivity** , правою кнопкою миші і в меню виберемо **New -> Fragment -> Fragment(Blank).**

Даний шаблон запропонувати вказати клас фрагмента та назву файлу пов'язаного з ним класу розмітки інтерфейсу.

8.2.2 Додавання логіки до фрагмента

Фрагмент визначає кнопку. Тепер додамо до цієї кнопки певну дію. Для цього змінимо клас ContentFragment:

```
1   package com.example.fragmentapp;
2
```

```
3   import android.os.Bundle;
4   import android.view.View;
5   import android.widget.Button;
6   import android.widget.TextView;
7
8   import androidx.annotation.NonNull;
9   import androidx.annotation.Nullable;
10  import androidx.fragment.app.Fragment;
11
12  import java.util.Date;
13
14  public class ContentFragment extends Fragment {
15
16      public ContentFragment(){
17          super(R.layout.fragment_content);
18      }
19
20      @Override
21      public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {
22          super.onViewCreated(view, savedInstanceState);
23          Button updateButton = view.findViewById(R.id.updateButton);
24          TextView updateBox = view.findViewById(R.id.dateTextView);
25
26          updateButton.setOnClickListener(new View.OnClickListener() {
27              @Override
28              public void onClick(View v) {
29                  String curDate = new Date().toString();
30                  updateBox.setText(curDate);
31              }
32          });
33      }
34  }
```

Тут перевизначено метод **onViewCreated** класу Fragment, який викликається після створення об'єкта View для візуального інтерфейсу, який представляє даний фрагмент. Створений об'єкт View передається як перший параметр. І далі ми можемо отримати конкретні елементи керування в рамках цього об'єкта View, зокрема TextView та Button, і виконати з ними деякі дії. У цьому випадку в обробнику натискання кнопки у текстовому полі виводиться поточна дата.

8.2.3 Додавання фрагмента у коді

Крім визначення фрагмента в xaml-файлі інтерфейсу, ми можемо додати його динамічно в activity.

Для цього змінимо файл **activity_main.xml** :

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.fragment.app.FragmentContainerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragment_container_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

І також змінимо клас **MainActivity** :

```
package com.example.fragmentapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        if (savedInstanceState == null) {
            getSupportFragmentManager().beginTransaction()
                    .add(R.id.fragment_container_view, ContentFragment.class,
null)
                    .commit();
        }
    }
}
```

Метод `getSupportFragmentManager()` повертає об'єкт `FragmentManager`, який керує фрагментами.

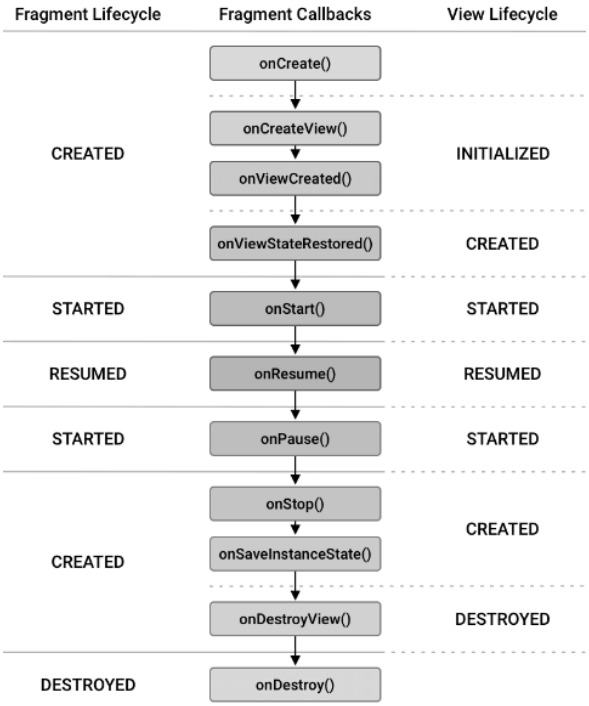
Об'єкт FragmentManager за допомогою методу `beginTransaction()` створює об'єкт **FragmentTransaction .**

FragmentTransaction виконує два методи: add() та commit(). Метод add() додає фрагмент: `add(R.id.fragment_container_view, new ContentFragment())` - першим аргументом передається ресурс розмітки, який треба додати фрагмент (це визначений в **activity_main.xml** елемент `androidx.fragment.app.FragmentContainerView`). І метод `commit()` підтверджує та завершує операцію додавання.

Підсумковий результат такого додавання фрагмента буде тим самим, що і при явному визначенні фрагмента через елемент `FragmentContainerView` розмітки інтерфейсу.

8.3 Життєвий цикл фрагменту. Додавання фрагменту. Видалення фрагменту

Кожен клас фрагмента успадковується від базового класу Fragment і має свій життєвий цикл, що складається з низки етапів:



Кожен етап життєвого циклу описується однією з констант перерахування **Lifecycle.State** :

- **INITIALIZED**
- **CREATED**
- **STARTED**
- **RESUMED**
- **DESTROYED**

Варто зазначити, що представлення фрагмента (його візуальний інтерфейс або View) має окремий цикл життя.

- При створенні фрагмент знаходиться в стані **INITIALIZED** . Щоб фрагмент пройшов решту етапів життєвого циклу, фрагмент необхідно передати в об'єкт **FragmentManager** , який далі визначає стан фрагмента і переводить фрагмент з одного стану до іншого.
- **onCreate()** : відбувається створення фрагмента. У цьому методі ми можемо отримати раніше збережений стан фрагмента через параметр методу `Bundle savedInstanceState`. (Якщо фрагмент створюється вперше, цей об'єкт має значення `null`) Цей метод викликається після виклику відповідного методу onCreate() у activity.

```
1  <font style="vertical-align: inherit;"><font style="vertical-align: inherit;">pu
```

```
savedInstanceState)</font></font>
```

onCreateView() : фрагмент створює уявлення (View або візуальний інтерфейс). У цьому методі ми можемо встановити, який ізуальний інтерфейс використовуватиме фрагмент. При виконанні цього методу подання фрагмента переходить у стан **INITIALIZED** . А сам фрагмент все ще перебуває в стані **CREATED**

```
1 <font style="vertical-align: inherit;"><font style="vertical-align: inherit;">Pu
2 savedInstanceState)
  </font></font>
```

- Перший параметр - об'єкт **LayoutInflater** дозволяє отримати вміст ресурсу layout і передати його у фрагмент.
- Другий параметр - об'єкт **ViewGroup** представляє контейнер, у якій завантажуватиметься фрагмент.
- Третій параметр – об'єкт **Bundle** представляє стан фрагмента. (Якщо фрагмент завантажується вперше, то дорівнює null)
- На виході метод повертає створене за допомогою **LayoutInflater** уявлення у вигляді об'єкта View - власне уявлення фрагмента
- **onViewCreated()** : викликається після створення уявлення фрагмента.

```
1 <font style="vertical-align: inherit;"><font style="vertical-align: inherit;">Pu
2 savedInstanceState)
  </font></font>
```

- Перший параметр - об'єкт **View** - представлення фрагмента, яке було створено за допомогою методу onCreateView.
- Другий параметр – об'єкт **Bundle** представляє стан фрагмента. (Якщо фрагмент завантажується вперше, то дорівнює null)
- **onViewStateRestored()** : отримує стан уявлення фрагмента. Після виконання цього методу подання фрагмента перетворюється на стан **CREATED**

```
1 public void onViewStateRestored (Bundle savedInstanceState)
```

onStart() : викликається, коли фрагмент стає видимим і разом із поданням перетворюється на стан **STARTED**

```
1 public void onStart ()
```

onResume() : викликається, коли фрагмент стає активним, і користувач може взаємодіяти з ним. При цьому фрагмент та його подання переходять у стан **RESUMED**

```
1 public void onResume ()
```

onPause() : фрагмент продовжує залишатися видимим, але вже не активний і разом із поданням переходить у стан **STARTED**

```
1  public void onPause ()
```

onStop() : фрагмент більше не є видимим і разом з поданням переходить у стан **CREATED**

```
1  public void onStop ()
```

- На цьому етапі життєвого циклу ми можемо зберегти стан фрагмента за допомогою методу **onSaveInstanceState()** . Однак, варто враховувати, що виклик цього методу залежить від версії API. До API 28 `onSaveInstanceState()` викликається до `onStop()`, а починаючи API 28 після `onStop()`.
- **onDestroyView()** : знищується представлення фрагмента. Подання переходить у стан **DESTROYED**
- **onDestroy()** : остаточно знищення фрагмента - він також переходить у стан **DESTROYED**

Додатково для фрагмента визначено два методи зворотного виклику, які пов'язані з прикріпленням фрагмента до діяльності:

- Коли фрагмент додається до FragmentManager і прикріплюється до певного класу Activity, у фрагмента викликається метод **onAttach()** . Даний метод викликається до решти методів життєвого циклу. Починаючи з цього моменту фрагмент стає активним, і FragmentManager починає керувати його життєвим циклом.
- Метод **onDetach()** викликається, коли фрагмент видаляється з FragmentManager і відкріплюється від класу Activity. Цей метод викликається після решти методів життєвого циклу.