Практическая работа 8

Методические материалы

Фрагменты

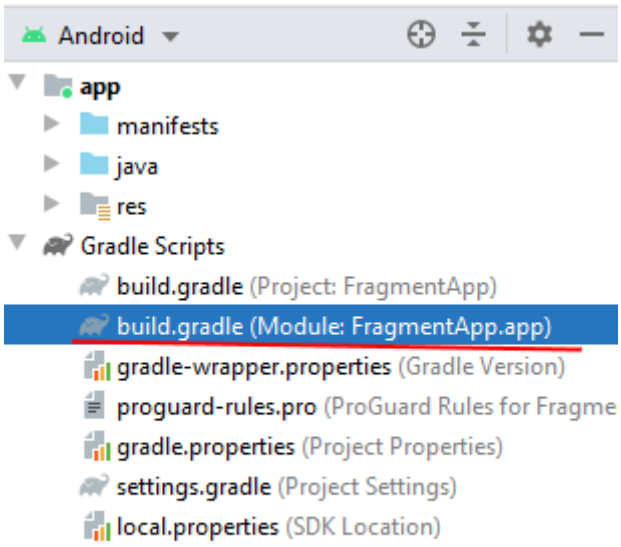
Введение во фрагменты

Организация приложения на основе нескольких activity не всегда может быть оптимальной. Мир ОС Android довольно сильно фрагментирован и состоит из многих устройств. И если для мобильных аппаратов с небольшими экранами взаимодействие между разными activity выглядит довольно неплохо, то на больших экранах - планшетах, телевизорах окна activity смотрелись бы не очень в силу большого размера экрана. Собственно поэтому и появилась концепция фрагментов.

Фрагмент представляет кусочек визуального интерфейса приложения, который может использоваться повторно и многократно. У фрагмента может быть собственный файл layout, у фрагментов есть свой собственный жизненный цикл. Фрагмент существует в контексте activity и имеет свой жизненный цикл, вне activity обособлено он существовать не может. Каждая activity может иметь несколько фрагментов.

Фрагменты в Android

Для начала работы с фрагментами создадим новый проект с пустой MainActivity. И вначале создадим первый фрагмент. Но сразу стоит отметить, что не вся функциональность фрагментов по умолчанию может быть доступна в проекте, поскольку располагается в отдельной библиотеке - **AndroidX Fragment library**. И вначале необходимо подключить к проекту эту библиотеку в файле build.gradle.



Подключение фрагментов и AndroidX Fragment library в Android и Java

Найдем в нем секцию **dependencies**, которая выглядит по умолчанию примерно так:

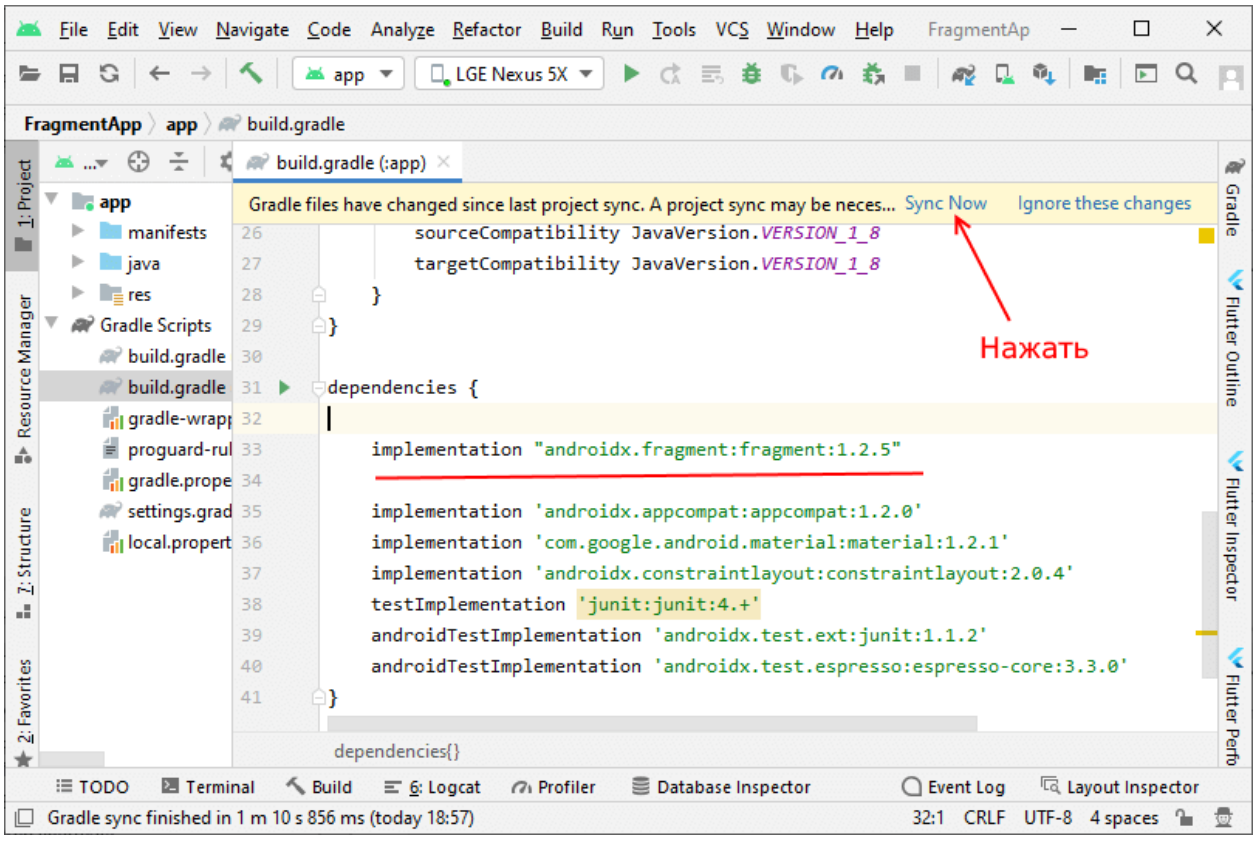
```
dependencies {

    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'
}
```

В ее начало добавим строку

implementation "androidx.fragment:fragment:1.3.6"

То есть в моем случае получится

```
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'
}
```



Подключение фрагментов и AndroidX Fragment library в Android и Java и Gradle

И затем нажмем на появившуюся ссылку **Sync Now.**

Фактически фрагмент - это обычный класс Java, который наследуется от класса **Fragment**. Однако как и класс Activity, фрагмент может использовать xml-файлы layout для определения графического интерфейса. И таким образом, мы можем добавить по отдельности класс Java, который представляет фрагмент, и файл xml для хранения в нем разметки интерфейса, который будет использовать фрагмент.

Итак, добавим в папку **res/layout** новый файл **fragment_content.xml** и определим в нем следующий 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" />

    <TextView
```

```
        android:id="@+id/dateTextView"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Hello from Fragment"
        android:textSize="28sp"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/updateButton" />

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

Фрагменты содержат те же элементы управления, что и activity. В частности, здесь определены кнопка и текстовое поле, которые будут составлять интерфейс фрагмента.

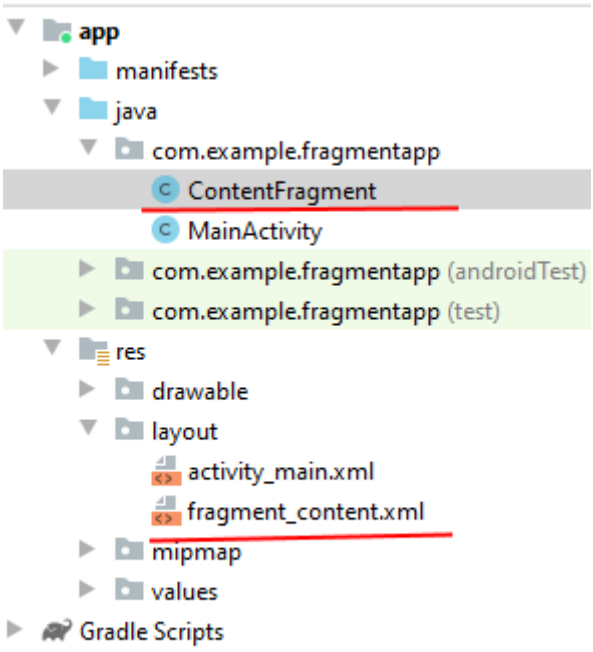
Теперь создадим сам класс фрагмента. Для этого добавим в одну папку с **MainActivity** новый класс. Для этого нажмем на папку правой кнопкой мыши и выберем в меню **New -> Java Class**. Назовем новый класс **ContentFragment** и определим у него следующее содержание:

```
package com.example.fragmentapp;

import androidx.fragment.app.Fragment;

public class ContentFragment extends Fragment {


    public ContentFragment(){
        super(R.layout.fragment_content);
    }
}
```

Класс фрагмента должен наследоваться от класса Fragment.

Чтобы указать, что фрагмент будет использовать определенный xml-файл layout, идентификатор ресурса layout передается в вызов конструктора родительского класса (то есть класса Fragment).

Весь проект будет выглядеть следующим образом:



Фрагменты в Android Studio и Java

Добавление фрагмента в Activity

Для использования фрагмента добавим его в **MainActivity.** Для этого изменим файл **activity_main.xml**, которая определяет интерфейс для MainActivity:

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

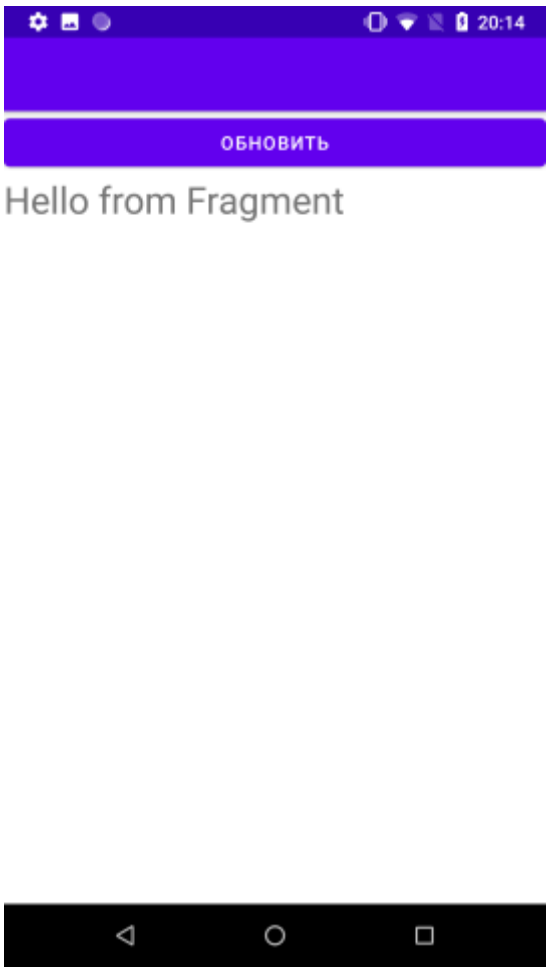
Для добавления фрамента применяется элемент **FragmentContainerView**. По сути, FragmentContainerView представляет объект View, который расширяет класс **FrameLayout** и предназначен специально для работы с фрагментами. Собственно кроме фрагментов он больше ничего содержать не может.

Его атрибут **android:name** указывает на имя класса фрагмента, который будет использоваться. В моем случае полное имя класса фрагмента с учетов пакета **com.example.fragmentapp.ContentFragment**.

Код класса **MainActivity** остается тем же, что и при создании проекта:

```
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);
    }
}
```

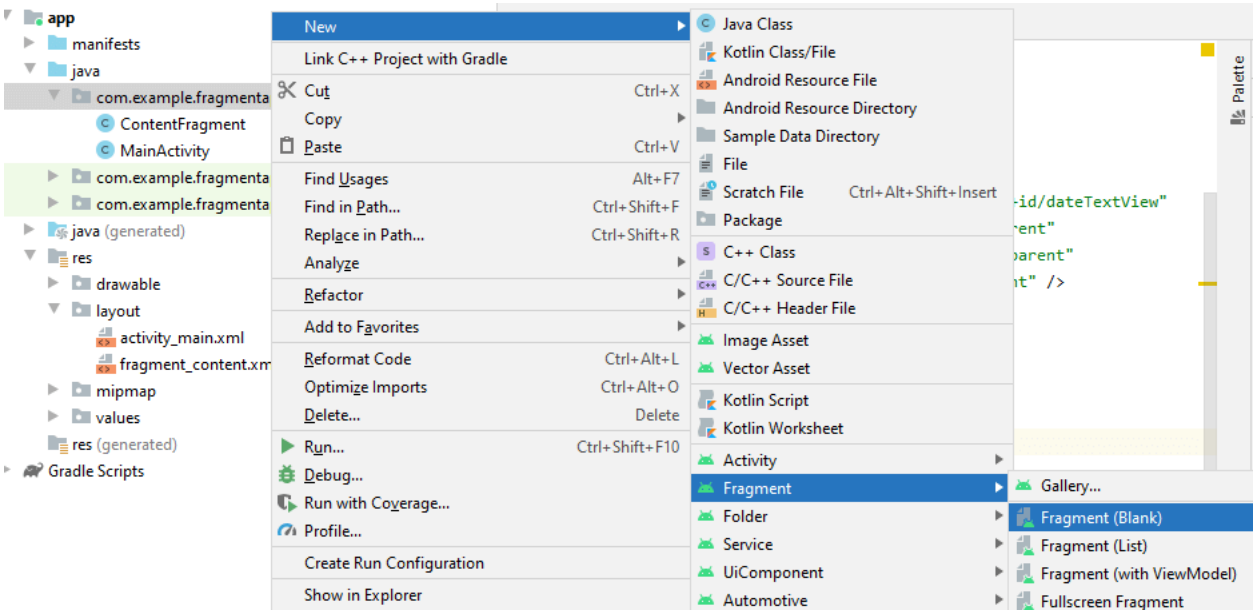
Если мы запустим приложение, то мы увидим фактически тот же самый интерфейс, который мы могли бы сделать и через activity, только в данном случае интерфейс будет определен во фрагменте:



Создание фрагмента для Android и Java

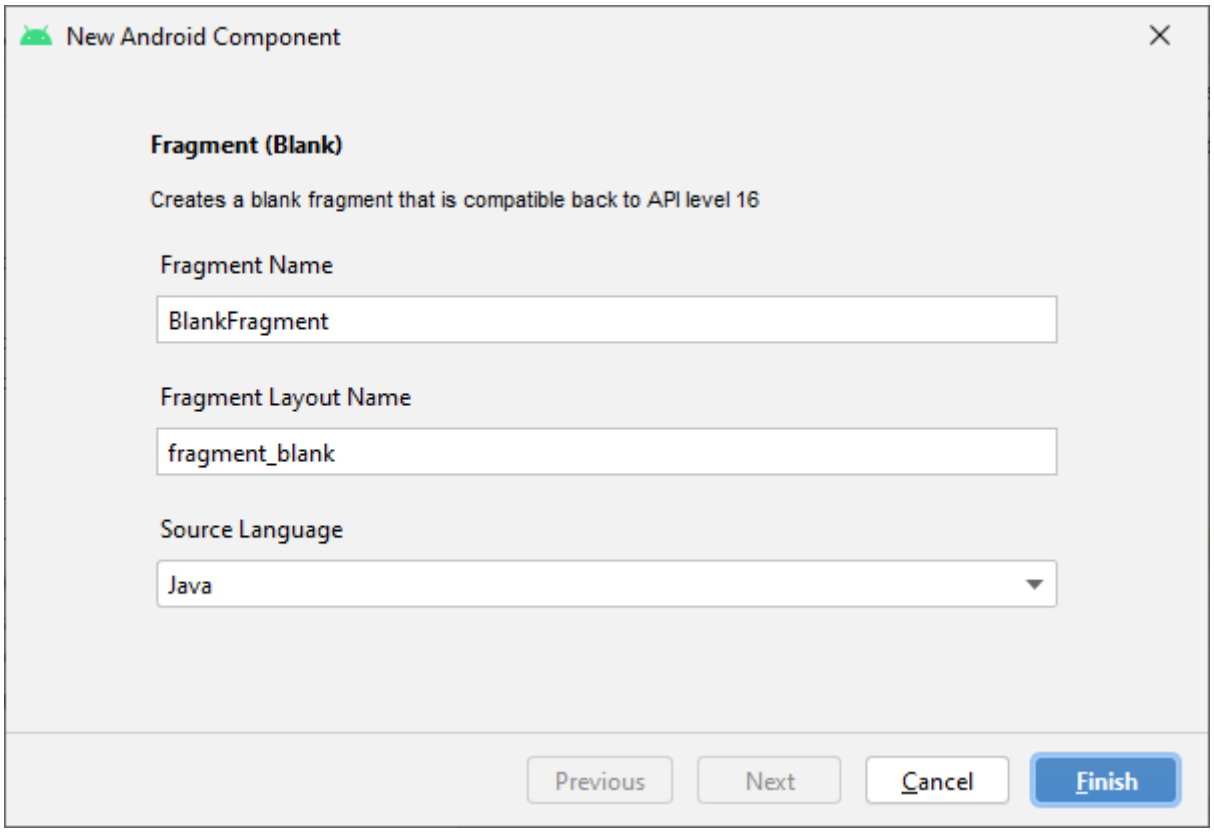
Стоит отметить, что Android Studio представляет готовый шаблон для добавления фрагмента. Собственно воспользуемся этим способом.

Для этого нажмем на папку, где находится класс **MainActivity**, правой кнопкой мыши и в появившемся меню выберем **New -> Fragment -> Fragment(Blank):**



Добавление фрагмента в Android Studio

Данный шаблон предложить указать класс фрагмента и название файла связанного с ним класса разметки интерфейса.



Добавление фрагмента в Android Studio и Java

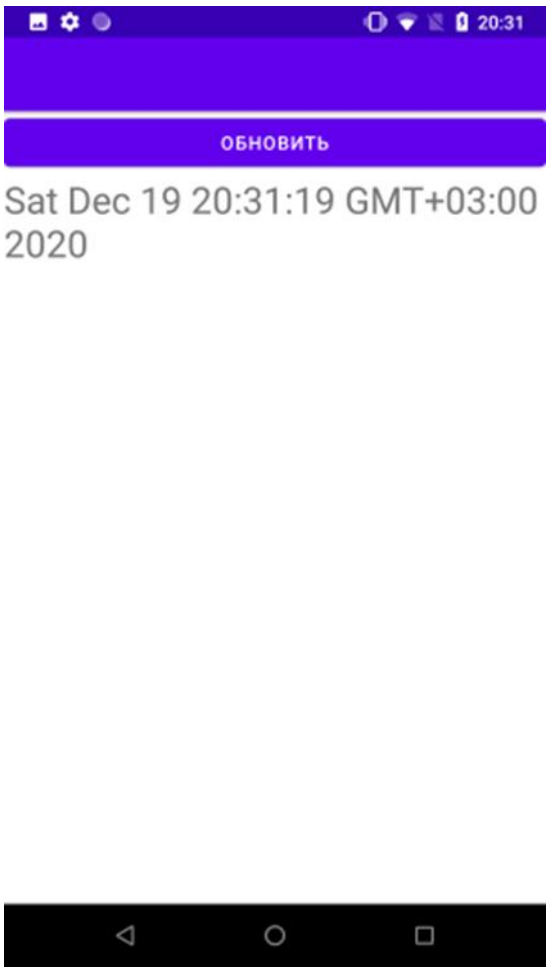
Добавление логики к фрагменту

Фрагмент определяет кнопку. Теперь добавим к этой кнопки некоторое действие. Для этого изменим класс ContentFragment:

```
package com.example.fragmentapp;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.fragment.app.Fragment;

import java.util.Date;

public class ContentFragment extends Fragment {

    public ContentFragment(){
        super(R.layout.fragment_content);
    }

    @Override
    public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        Button updateButton = view.findViewById(R.id.updateButton);
        TextView updateBox = view.findViewById(R.id.dateTextView);
```

```
        updateButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String curDate = new Date().toString();
                updateBox.setText(curDate);
            }
        });
    }
}
```

Здесь переопределен метод **onViewCreated** класса Fragment, который вызывается после создания объекта View для визуального интерфейса, который представляет данный фрагмент. Созданный объект View передается в качестве первого параметра. И далее мы можем получить конкретные элементы управления в рамках этого объекта View, в частности, TextView и Button, и выполнить с ними некоторые действия. В данном случае в обработчике нажатия кнопки в текстовом поле выводится текущая дата.



Добавление фрагмента в AndroidX Fragment Library и Java

Добавление фрагмента в коде

Кроме определения фрагмента в xaml-файле интерфейса мы можем добавить его динамически в activity.

Для этого изменим 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" />
```

И также изменим класс 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() возвращает объект FragmentManager, который управляет фрагментами.

Объект FragmentManager с помощью метода beginTransaction() создает объект **FragmentTransaction**.

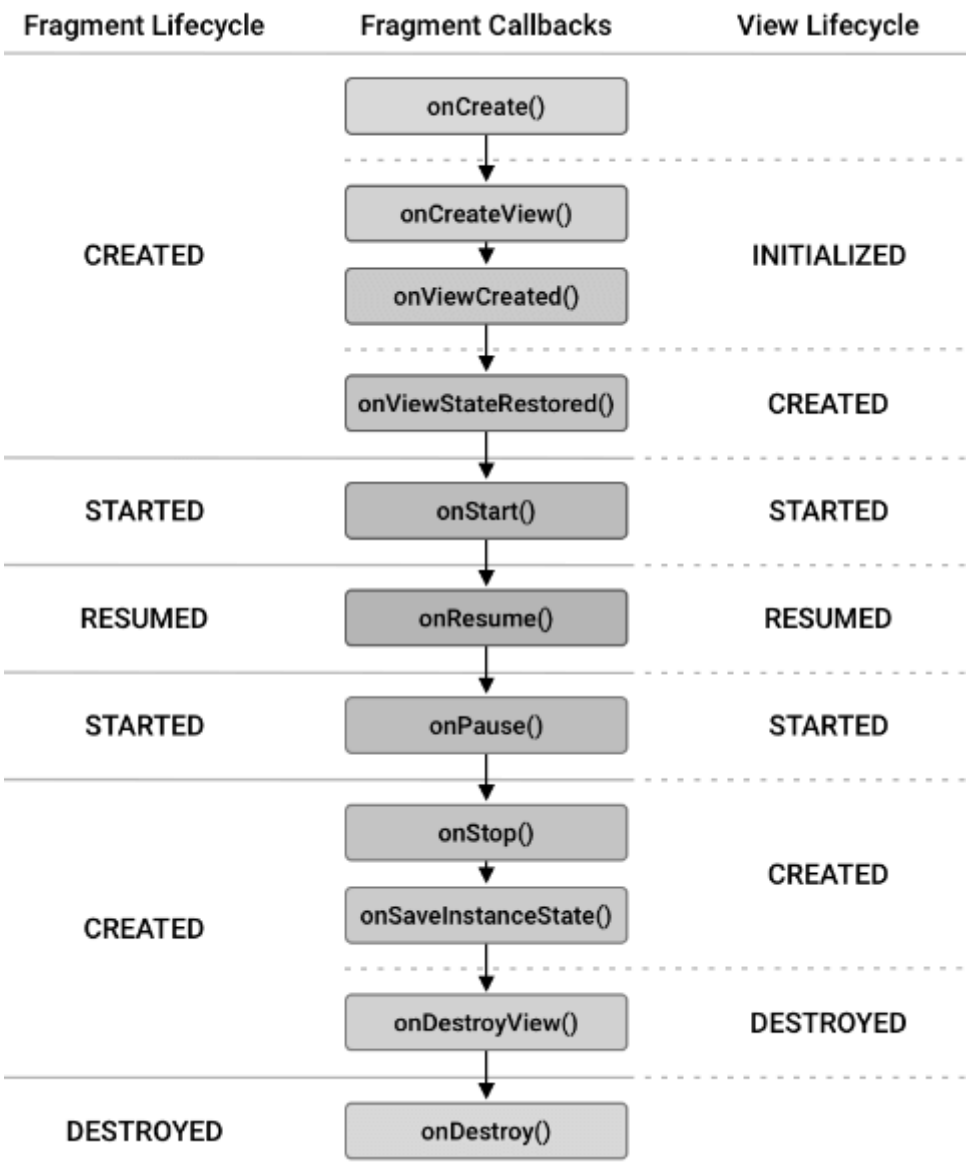
FragmentTransaction выполняет два метода: add() и commit(). Метод add() добавляет фрагмент: add(R.id.fragment_container_view, new

ContentFragment()) - первым аргументом передается ресурс разметки, в который надо добавить фрагмент (это определенный в **activity_main.xml** элемент androidx.fragment.app.FragmentContainerView). И метод commit() подтвержает и завершает операцию добавления.

Итоговый результат такого добавления фрагмента будет тем же, что и при явном определении фрагмента через элемент FragmentContainerView в разметке интерфейса.

Жизненный цикл фрагментов

Каждый класс фрагмента наследуется от базового класса Fragment и имеет свой жизненный цикл, состоящий из ряда этапов:



Жизненный цикл фрагментов в Android

Каждый этап жизненного цикла описывается одной из констант перечисления **Lifecycle.State**:

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

Стоит отметить, что представление фрагмента (его визуальный интерфейс или View) имеет отдельный жизненный цикл.

- При создании фрагмент находится в состоянии **INITIALIZED**. Чтобы фрагмент прошел все остальные этапы жизненного цикла, фрагмент необходимо передать в объект F**ragmentManager**, который далее определяет состояние фрагмента и переводит фрагмент из одного состояния в другое.

- Когда фрагмент добавляется в FragmentManager и прикрепляется к определенному классу Activity, у фрагмента вызывается метод **onAttach().** Данный метод вызывается до всех остальных методов жизненного цикла. После этого фрагмент переходит в состояние **CREATED**

- **onCreate():** происходит создание фрагмента. В этом методе мы можем получить ранее сохраненное состояние фрагмента через

параметр метода Bundle savedInstanceState. (Если фрагмент создается первый раз, то этот объект имеет значение null) Этот метод вызывается после вызова соответствующего метода onCreate() у activity.

public void onCreate(Bundle savedInstanceState)

- o**nCreateView()**: фрагмент создает представление (View или визуальный интерфейс). В этом методе мы можем установить, какой именно изуальный интерфейс будет использовать фрагмент. При выполнении этого метода представление фрагмента переходит в состояние **INITIALIZED**. А сам фрагмент все еще находится в состоянии **CREATED**

public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)

Первый параметр - объект **LayoutInflater** позволяет получить содержимое ресурса layout и передать его во фрагмент.

Второй параметр - объект **ViewGroup** представляет контейнер, в которой будет загружаться фрагмент.

Третий параметр - объект **Bundle** представляет состояние фрагмента. (Если фрагмент загружается первый раз, то равен null)

На выходе метод возвращает созданное с помощью **LayoutInflater** представление в виде объекта View - собственно представление фрагмента

- onViewCreated(): вызывается после создания представления фрагмента.

public void onViewCreated (View view, Bundle savedInstanceState)

Первый параметр - объект **View** - представление фрагмента, которое было создано посредством метода onCreateView.

Второй параметр - объект **Bundle** представляет состояние фрагмента. (Если фрагмент загружается первый раз, то равен null)

- **onViewStateRestored():** получает состояние представления фрагмента. После выполнения этого метода представление фрагмента переходит в состояние **CREATED**

public void onViewStateRestored (Bundle savedInstanceState)

- **onStart():** вызывается, когда фрагмент становится видимым и вместе с представлением переходит в состояние STARTED

public void onStart ()

- **onResume():** вызывается, когда фрагмент становится активным, и пользователь может с ним взаимодействовать. При этом фрагмент и его представление переходят в состояние **RESUMED**

public void onResume ()

- **onPause():** фрагмент продолжает оставаться видимым, но уже не активен и вместе с представлением переходит в состояние **STARTED**

public void onPause ()

- **onStop():** фрагмент больше не является видимым и вместе с представлением переходит в состояние **CREATED**

public void onStop ()

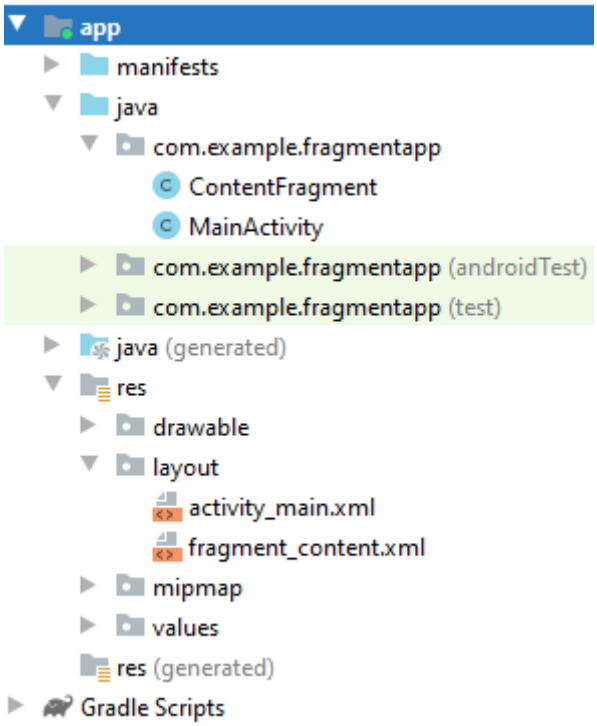
На этом этапе жизненного цикла мы можем сохранить состояние фрагмента с помощью метода **onSaveInstanceState().** Однако стоит учитывать, что вызов этого метода зависит от версии API. До API 28 onSaveInstanceState() вызывается до onStop(), а начиная API 28 после onStop().

- **onDestroyView():** уничтожается представление фрагмента. Представление переходит в состояние **DESTROYED**

- **onDestroy():** окончательно уничтожение фрагмента - он также переходит в состояние **DESTROYED**

- Метод **onDetach()** вызывается, когда фрагмент удаляется из FragmentManager и открепляется от класса Activity. Этот метод вызывается после всех остальных методов жизненного цикла.

В коде класса фрагмента мы можем переопределить все или часть из этих методов. Например, пусть у нас будет определен следующий проект:



Fragment Lifecycle жизненный цикл фрагмента в Android Studio и Java

В каталоге **res/layout** определен файл layout для фрагмента - **fragment_content.xml** со следующей разметкой:

<?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" />

    <TextView
        android:id="@+id/dateTextView"
        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_toBottomOf="@+id/updateButton" />

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

Класс фрагмента использует данный файл для установки представления, а также определяет методы для управления жизненным циклом:

```
package com.example.fragmentapp;
```

```
import android.content.Context;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.TextView;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.fragment.app.Fragment;
import android.util.Log;

import java.util.Date;

public class ContentFragment extends Fragment {

    private final static String TAG = "ContentFragment";

    public ContentFragment(){
        Log.d(TAG, "Constructor");
    }

    @Override
    public void onAttach(@NonNull Context context) {
        super.onAttach(context);
        Log.d(TAG, "onAttach");
    }
```

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

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        Log.d(TAG, "onCreateView");
        return inflater.inflate(R.layout.fragment_content, container, false);
    }
    @Override
    public void onViewCreated(@NonNull View view, @Nullable Bundle
savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        Button updateButton = view.findViewById(R.id.updateButton);
        TextView updateBox = view.findViewById(R.id.dateTextView);

        updateButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String curDate = new Date().toString();
                updateBox.setText(curDate);
            }
        });
        Log.d(TAG, "onViewCreated");
    }
```

```
@Override
public void onViewStateRestored(@Nullable Bundle savedInstanceState) {
    super.onViewStateRestored(savedInstanceState);
    Log.d(TAG, "onViewStateRestored");
}

@Override
public void onStart() {
    super.onStart();
    Log.d(TAG, "onStart");
}

@Override
public void onResume() {
    super.onResume();
    Log.d(TAG, "onResume");
}

@Override
public void onPause() {
    super.onPause();
    Log.d(TAG, "onPause");
}

@Override
public void onStop() {
    super.onStop();
    Log.d(TAG, "onStop");
```

```
    }

    @Override
    public void onDestroyView() {
        super.onDestroyView();
        Log.d(TAG, "onDestroyView");
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        Log.d(TAG, "onDestroy");
    }

    @Override
    public void onDetach() {
        super.onDetach();
        Log.d(TAG, "onDetach");
    }
}
```

В отличие от прошлого материала, где рассматривалось создание фрагмента, здесь фрагмент устанавливает представление в методе **onCreateView**. Для этого в метод inflate() объекта LayoutInflater передается идентификатор ресурса layout и контейнер - объект ViewGroup, в который будет загружаться фрагмент. В итоге метод inflate() возвращает созданное представление.

```
return inflater.inflate(R.layout.fragment_content, container, false);
```

При выполнении метода **onViewCreated()** представление уже создано и оно передается в качестве первого параметра - объекта View, через который с

помощью идентификаторов мы можем получить визуальные элементы - TextView и Button, которые определены в представлении.

Для остальных методов жизненного цикла установлено простое логгирование с помощью метода Log.d().

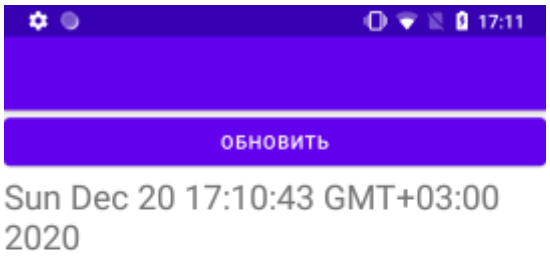
Пусть в файле **activity_main.xml** происходит добавление фрагмента:

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

И класс **MainActivity:**

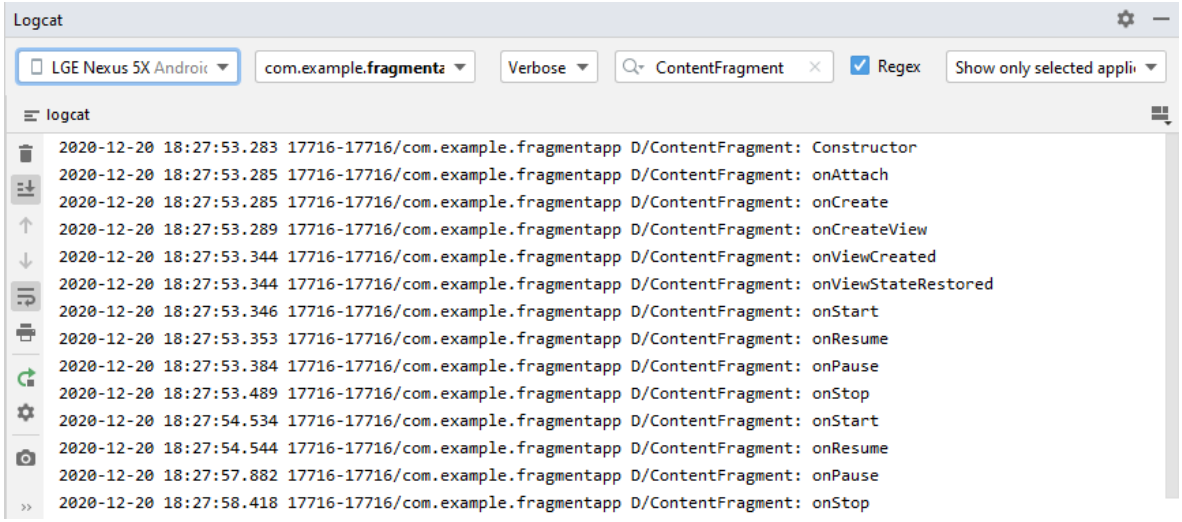
```
package com.example.fragmentapp;

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);
    }
}
```

Если мы запустим проект, то на экране устройства мы увидим визуальный интерфейс, определенный для фрагмента.



Жизненный цикл фрагмента в Android Studio и Java, onViewCreated

А в окне Logcat в Android Studio можно будет наблюдать логгирование методов жизненного цикла



Взаимодействие между фрагментами

Одна activity может использовать несколько фрагментов, например, с одной стороны список, а с другой - детальное описание выбранного элемента списка. В такой конфигурации activity использует два фрагмента, которые между собой должны взаимодействовать. Рассмотрим базовые принципы взаимодействия фрагментов в приложении.

Создадим новый проект с пустой MainActivity. Далее создадим разметку layout для фрагментов. Пусть у нас в приложении будет два фрагмента. Добавим в папку **res/layout** новый xml-файл **fragment_list.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"
    android:padding="16dp">
    <ListView
        android:id="@+id/countriesList"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent">
    </ListView>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Здесь определен элемент ListView для вывода списка объектов.

И также добавим для другого фрагмента файл разметки **fragment_detail.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"
    android:padding="16dp">
    <TextView
        android:id="@+id/detailsText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"

        android:text="Не выбрано"
        android:textSize="22sp"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Оба фрагмента будут предельно простыми: один будет содержать список, а второй - текстовой поле. Логика приложения будет такова: при выборе элемента в списке в одном фрагменте выбранный элемент должен отобразиться в текстовом поле, которое находится во втором фрагменте.

Затем добавим в проект в одну папку с MainActivity собственно классы фрагментов. Добавим новый класс **ListFragment** со следующим содержимым:

```
package com.example.fragmentapp;
```

```
import android.content.Context;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import androidx.fragment.app.Fragment;

public class ListFragment extends Fragment {

    interface OnFragmentSendDataListener {
        void onSendData(String data);
    }

    private OnFragmentSendDataListener fragmentSendDataListener;
    String[] countries = { "Бразилия", "Аргентина", "Колумбия", "Чили",
"Уругвай"};


    @Override
    public void onAttach(Context context) {
        super.onAttach(context);
        try {
            fragmentSendDataListener = (OnFragmentSendDataListener) context;
        } catch (ClassCastException e) {
            throw new ClassCastException(context.toString()
```

```
                + " должен реализовывать интерфейс
OnFragmentInteractionListener");
        }
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_list, container, false);
        // получаем элемент ListView
        ListView countriesList = view.findViewById(R.id.countriesList);
        // создаем адаптер
        ArrayAdapter<String> adapter = new ArrayAdapter(getContext(),
android.R.layout.simple_list_item_1, countries);
        // устанавливаем для списка адаптер
        countriesList.setAdapter(adapter);
        // добавляем для списка слушатель
        countriesList.setOnItemClickListener(new
AdapterView.OnItemClickListener(){
            @Override
            public void onItemClick(AdapterView<?> parent, View v, int position,
long id)
            {
                // получаем выбранный элемент
                String selectedItem = (String)parent.getItemAtPosition(position);
                // Посылаем данные Activity
                fragmentSendDataListener.onSendData(selectedItem);
            }
        });
        return view;
```

```
    }

}
```

Фрагменты не могут напрямую взаимодействовать между собой. Для этого надо обращаться к контексту, в качестве которого выступает класс Activity. Для обращения к activity, как правило, создается вложенный интерфейс. В данном случае он называется **OnFragmentSendDataListener** с одним методом.

```
interface OnFragmentSendDataListener {
        void onSendData(String data);
}
```

```
private OnFragmentSendDataListener fragmentSendDataListener;
```

Но чтобы взаимодействовать с другим фрагментом через activity, нам надо прикрепить текущий фрагмент к activity. Для этого в классе фрагмента определен метод onAttach(Context context). В нем происходит установка объекта OnFragmentSendDataListener:

```
fragmentSendDataListener = (OnFragmentSendDataListener) context;
```

При обработке нажатия на элемент в списке мы можем отправить Activity данные о выбранном объекте:

```
String selectedItem = (String)parent.getItemAtPosition(position);
fragmentSendDataListener.onSendData(selectedItem);
```

Таким образом, при выборе объекта в списке MainActivity получит выбранный объект.

Теперь определим класс для второго фрагмента. Назовем его **DetailFragment**:

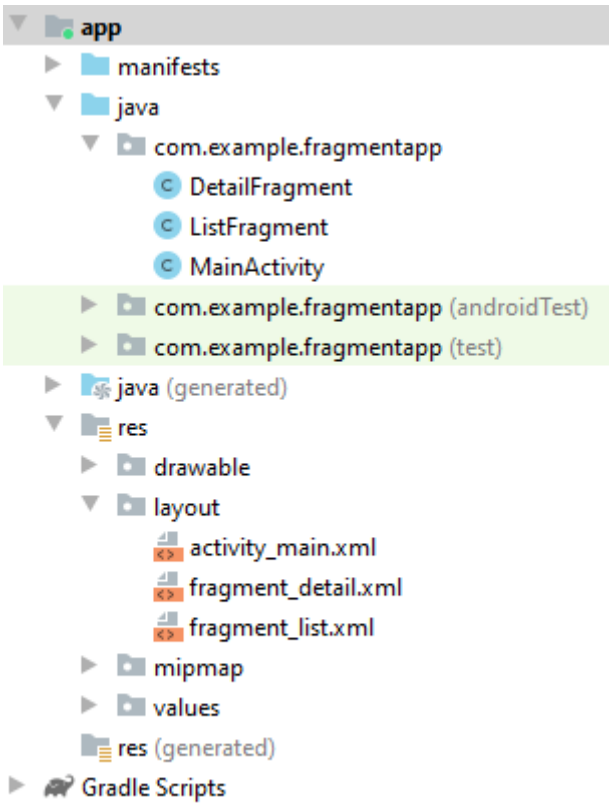
```
package com.example.fragmentapp;
```

```
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
import androidx.fragment.app.Fragment;

public class DetailFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_detail, container, false);
    }

    // обновление текстового поля
    public void setSelectedItem(String selectedItem) {
        TextView view = getView().findViewById(R.id.detailsText);
        view.setText(selectedItem);
    }
}
```

Задача этого фрагмента - вывод некоторой информации. Так как он не должен передавать никакую информацию другому фрагменту, здесь мы модем ограничиться только переопределением метода onCreateView(), который в качестве визуального интерфейса устанавливает разметку из файла **fragment_detail.xml**

Но чтобы имитировать взаимодействие между двумя фрагментами, здесь также определен метод setSelectedItem(), который обновляет текст на текстовом поле.

В итоге получится следующая структура:



Создание фрагментов в Android

Теперь изменим файл разметки **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" >

    <androidx.fragment.app.FragmentContainerView
```

```
        android:id="@+id/listFragment"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:name="com.example.fragmentapp.ListFragment"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toTopOf="@+id/detailFragment"
        app:layout_constraintRight_toRightOf="parent"/>

    <androidx.fragment.app.FragmentContainerView
        android:id="@+id/detailFragment"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:name="com.example.fragmentapp.DetailFragment"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toBottomOf="@id/listFragment"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintRight_toRightOf="parent"/>

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

С помощью двух элементов **FragmentContainerView** в MainActivity добавляются два выше определенных фрагмента.

И в конце изменим код MainActivity:

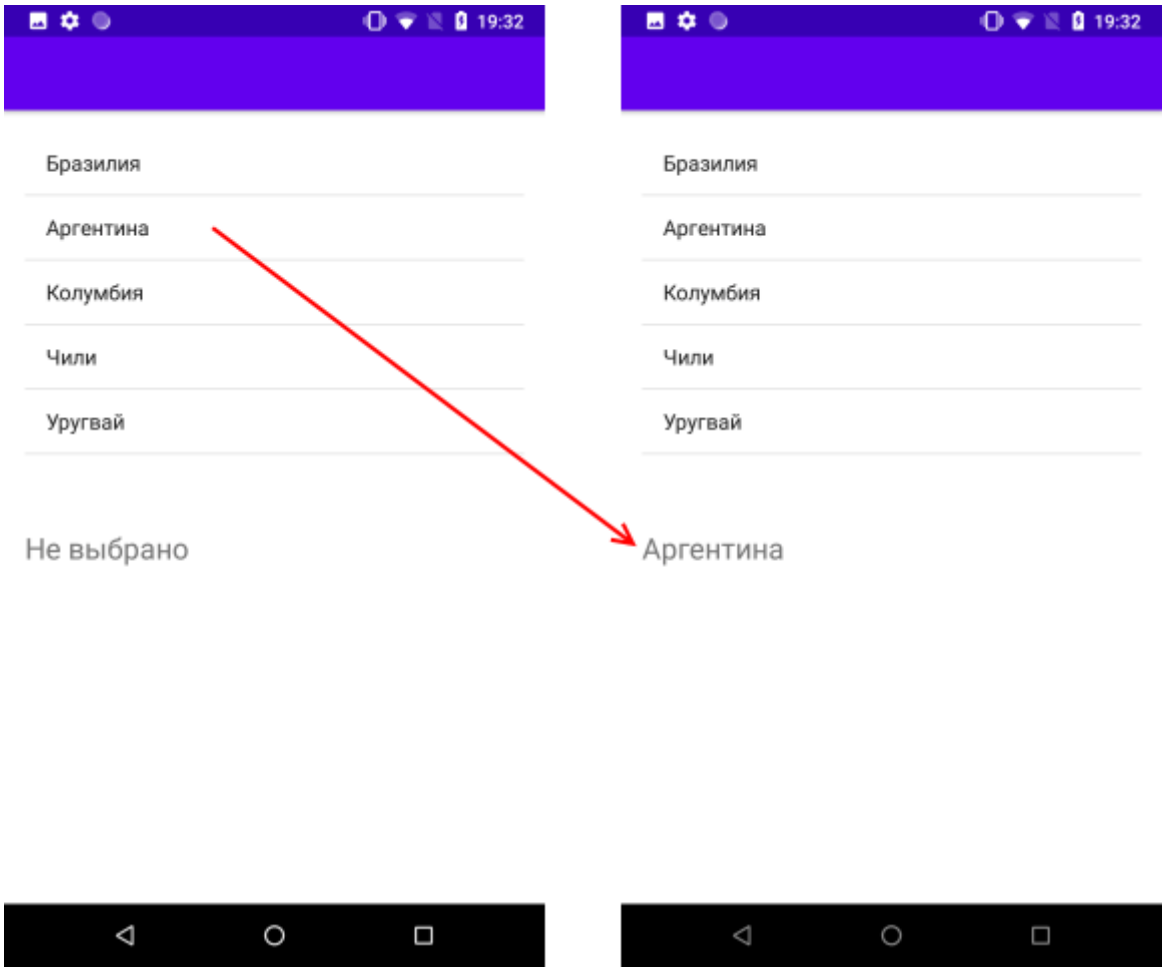
```
package com.example.fragmentapp;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
```

```
public class MainActivity extends AppCompatActivity  implements
ListFragment.OnFragmentSendDataListener {


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


    @Override
    public void onSendData(String selectedItem) {
        DetailFragment fragment = (DetailFragment) getSupportFragmentManager()
                .findFragmentById(R.id.detailFragment);
        if (fragment != null)
            fragment.setSelectedItem(selectedItem);
    }
}
```

Для взаимодействия фрагмента ListFragment с другим фрагментом через MainActivity надо, чтобы эта activity реализовывала интерфейс OnFragmentSendDataListener. Для этого реализуем метод onSendData(), который получает фрагмент DetailFragment и вызывает у него метод setSelectedItem()

В итоге получится, что при выборе в списке во фрагменте ListFragment будет срабатывать слушатель списка и в частности его метод onItemClick(AdapterView<?> parent, View v, int position, long id), который вызовет метод fragmentSendDataListener.onSendData(selectedItem);. fragmentSendDataListener устанавливается как MainActivity, поэтому при этом будет вызван метод setSelectedItem у фрагмента DetailFragment. Таким образом, произойдет взаимодействие между двумя фрагментами.

Если мы запустим проект, то на экран будут выведены оба фрагмента, которые смогут взаимодействовать между собой.

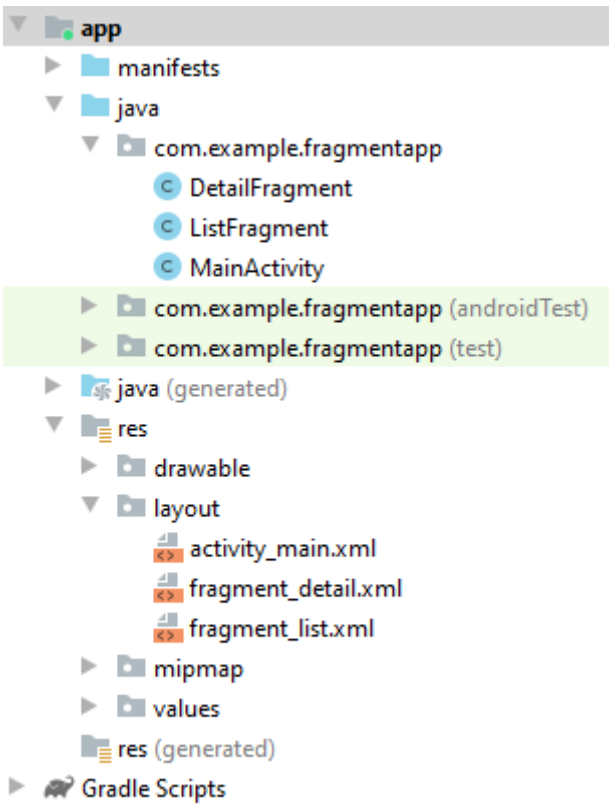


Фрагменты в activity в Android и Java

Однако пока фрагменты одинаково выводятся в одной activity как в альбомной, так и в портретной ориентации вне зависимости от устройства. Поэтому оптимизируем приложение

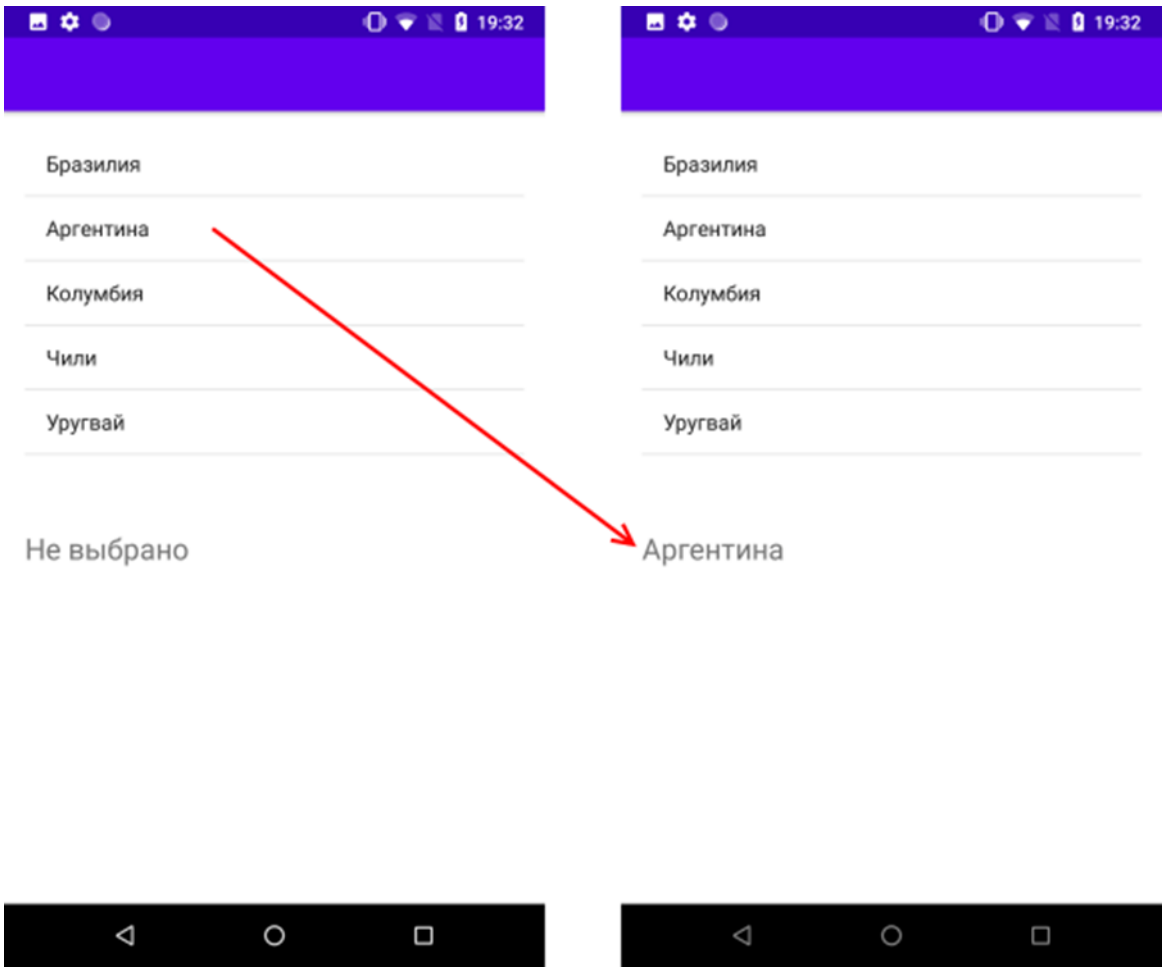
Фрагменты в альбомном и портретном режиме

В прошлом материале было разработано приложение, которое выводит оба фрагмента на экран. Продолжим работу с этим проектом. Всего было создано два фрагмента: ListFragment для отображения списка и DetailFragment для отображения выбранного элемента в списке. И MainActivity выводила оба фрагмента на экран:



Создание фрагментов в Android

Но отображение двух и более фрагментов при портретной ориентации не очень оптимально. Например, в прошлом материале приложение выглядело так:



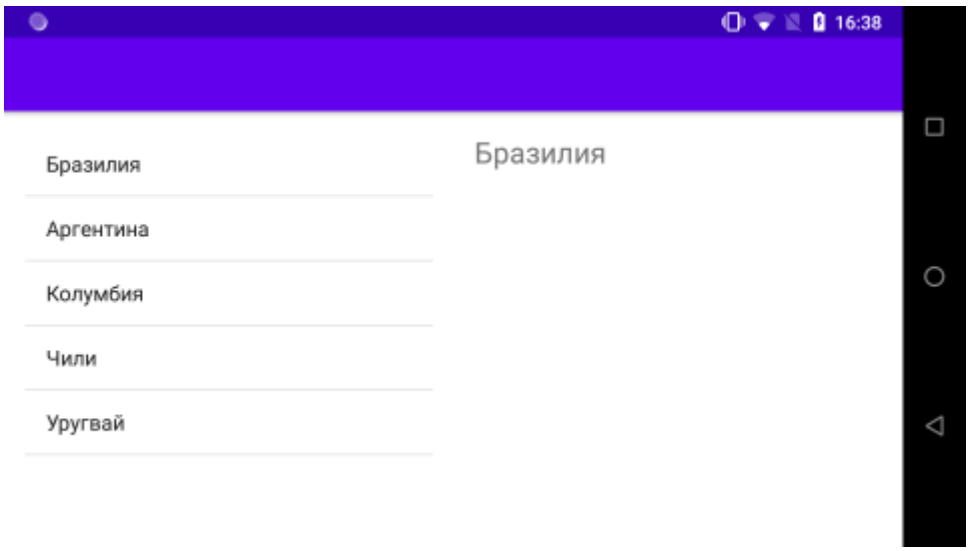
Фрагменты в activity в Android и Java

Но если список большой, то второй фрагмент, который отображает выбранный элемент, соответственно уходит вниз. При альбомной ориентации получится расположение еще более неоптимально. Поэтому сначала изменим файл activity_main.xml, чтобы более удобно располагать фрагменты в альбомной ориентации:

```
<?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" >
    <androidx.fragment.app.FragmentContainerView
        android:id="@+id/listFragment"
```

```
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:name="com.example.fragmentapp.ListFragment"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintRight_toLeftOf="@id/detailFragment"
        app:layout_constraintBottom_toBottomOf="parent" />

    <androidx.fragment.app.FragmentContainerView
        android:id="@+id/detailFragment"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:name="com.example.fragmentapp.DetailFragment"
        app:layout_constraintLeft_toRightOf="@id/listFragment"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintRight_toRightOf="parent"/>

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

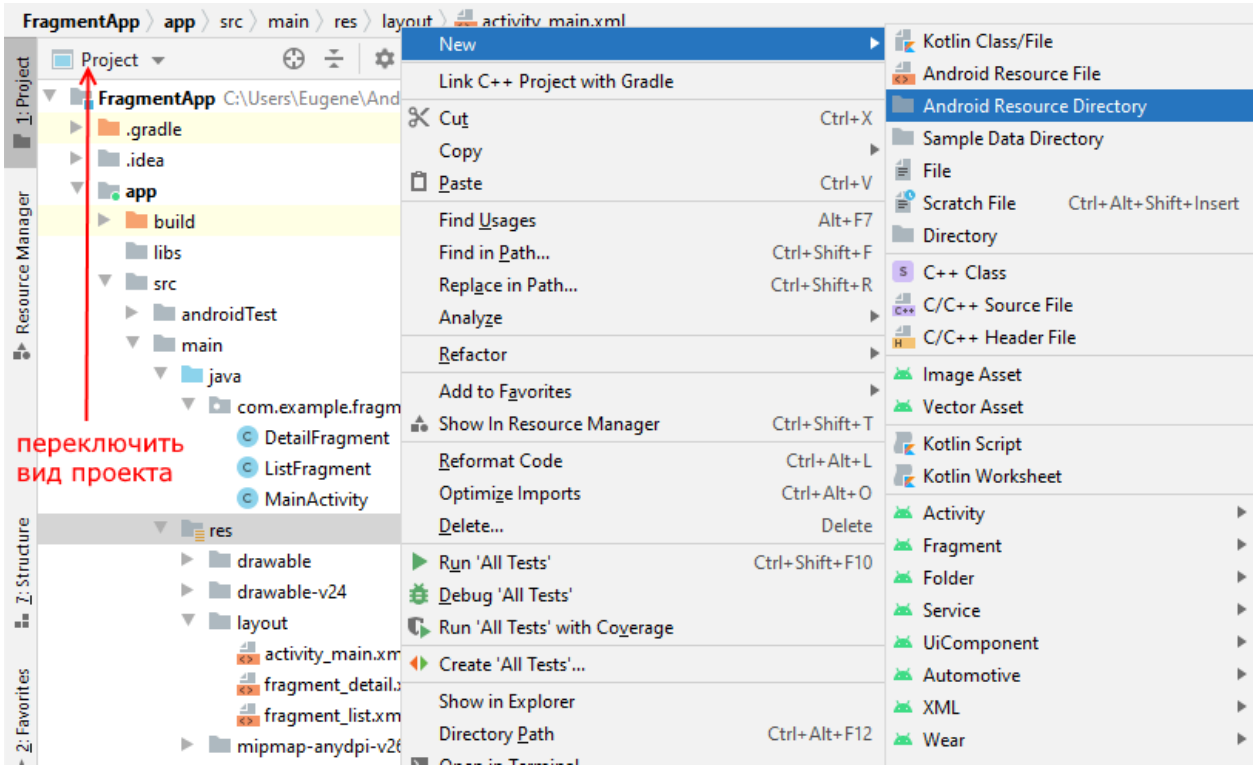
Для более удобного расположения при альбомной ориентации, как правило, решение довольно простое - два фрагмента располагаются горизонтально в ряд.



Фрагмент в альбомной ориентации в Android и Java

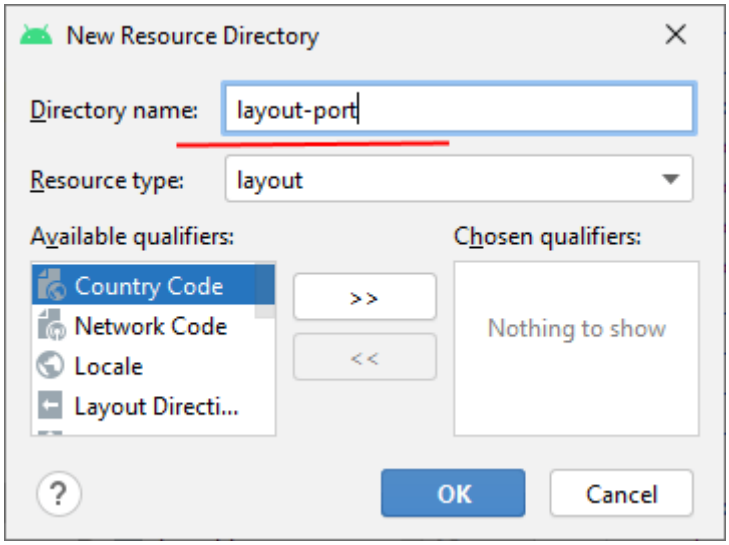
Теперь рассмотрим, как более удачно расположить фрагменты при портретной ориентации. Нередко в этом случае решение следующее - одномоментно экран отображает только один фрагмент.

Итак, создадим в проекте в папке **res**, где хранятся все ресурсы, подкаталог **layout-port**, который будет хранить файлы интерфейса для портретной ориентации. Для этого переключимся к полному виду проекта. Нажмем на папку **res** правой кнопкой мыши и в контекстном меню выберем **New -> Android Resource Directory:**



Фрагменты в портретной ориентации в Android и Java

Назовем новую папку layout-port:



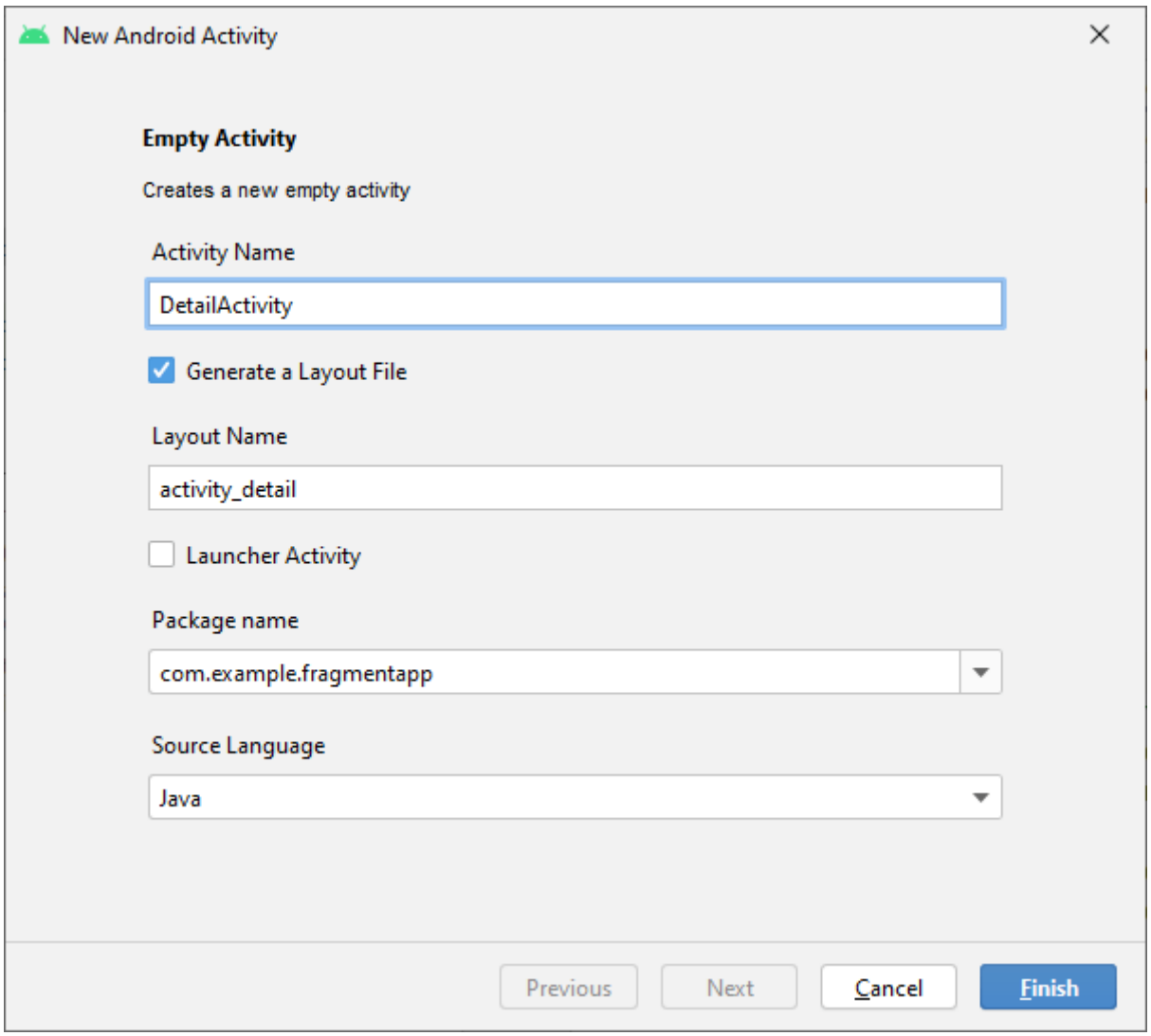
layout in portrait orientation in Android и Java

Далее добавим в res/layout-port новый файл activity_main.xml и определим в нем следующий код:

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

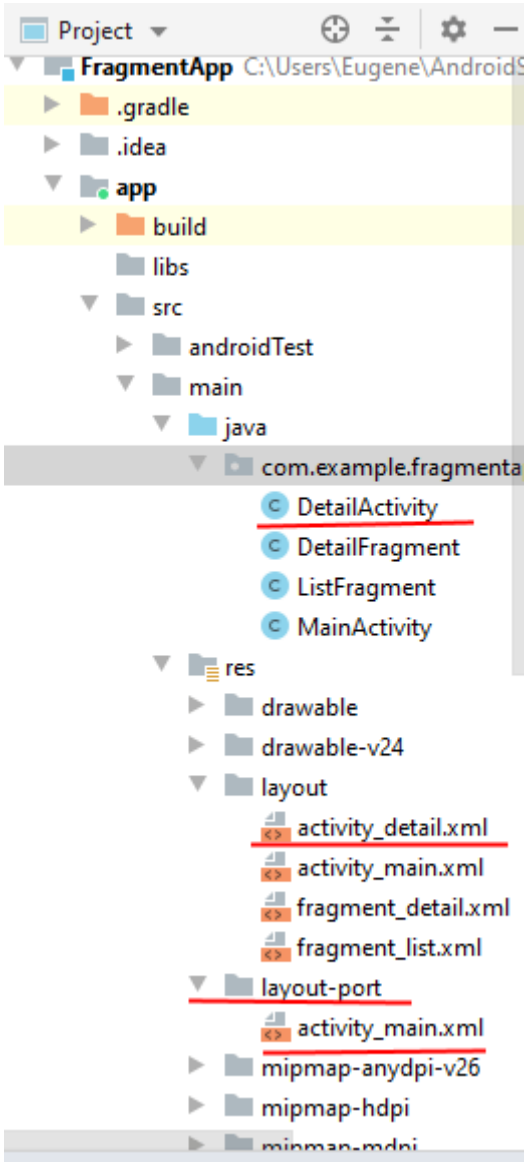
Этот файл будет использоваться для портретной ориентации MainActivity. Таким образом, MainActivity в портретном режиме будет отображать только один список.

Теперь добавим новую activity, которую назовем DetailActivity:



layout in portrait orientation and fragments in Android и Java

В итоге проект будет выглядеть так:



Фрагменты в портретной и альбомной ориентации в Android и Java

В папке **res/layout** в файле **activity_detail.xml** определим для DetailActivity следующий интерфейс:

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

Таким образом, интерфейс DetailActivity будет определяться загружаемым фрагментом DetailFragment.

Далее в коде **DetailActivity** определим следующий код:

```
package com.example.fragmentapp;

import androidx.appcompat.app.AppCompatActivity;
import android.content.res.Configuration;
import android.os.Bundle;

public class DetailActivity extends AppCompatActivity {

    public static final String SELECTED_ITEM = "SELECTED_ITEM";
    String selectedItem = "Не выбрано";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        if (getResources().getConfiguration().orientation ==
Configuration.ORIENTATION_LANDSCAPE) {
            finish();
            return;
        }
        setContentView(R.layout.activity_detail);
        Bundle extras = getIntent().getExtras();
        if (extras != null)
            selectedItem = extras.getString(SELECTED_ITEM);
    }
```

```
    @Override
    protected void onResume() {
        super.onResume();
        DetailFragment fragment = (DetailFragment) getSupportFragmentManager()
                .findFragmentById(R.id.detailFragment);
        if (fragment != null)
            fragment.setSelectedItem(selectedItem);
    }
}
```

Здесь в первую очередь проверяем конфигурацию. Так как эта activity предназначена только для портретного режима, то при альбомной ориентации осуществляем выход:

```
if (getResources().getConfiguration().orientation ==
Configuration.ORIENTATION_LANDSCAPE) {
    finish();
    return;
}
```

Если устройство находится в портретном режиме, то получаем переданные данные по ключу "SELECTED_ITEM":

```
Bundle extras = getIntent().getExtras();
if (extras != null)
    selectedItem = extras.getString(SELECTED_ITEM);
```

Предполагается, что по ключу "SELECTED_ITEM" будет передаваться выбранные элемент списка из MainActivity, когда она будет находиться в портретной ориентации.

И очень важный момент - нам надо передать это значение в текстовое поле, определенное во фрагменте. Однако надо учитывать особенности жизненного цикла представления фрагмента. В данном случае переопределяется метод **onResume()**, потому что при вызове этого метода DetailActivity уже будет видима на экране, и пользователь сможет с ней взаимодействовать. А это также значит, что в этой точке уже будет активен и фрагмент и его представление. К примеру, в методе onCreate() представление фрагмента еще полностью не создано, поэтому мы не можем в нем получить виджеты, которые определены во фрагменте. Зато можем все это сделать в методе onResume().

```
protected void onResume() {
        super.onResume();
        DetailFragment fragment = (DetailFragment) getSupportFragmentManager()
                .findFragmentById(R.id.detailFragment);
        if (fragment != null)
            fragment.setSelectedItem(selectedItem);
    }
```

И в данном случае мы получаем через метод getSupportFragmentManager() фрагмент DetailFragment и вызываем его метод setSelectedItem(). В качестве аргумента в этот метод передается строковое значение, переданное через Intent.

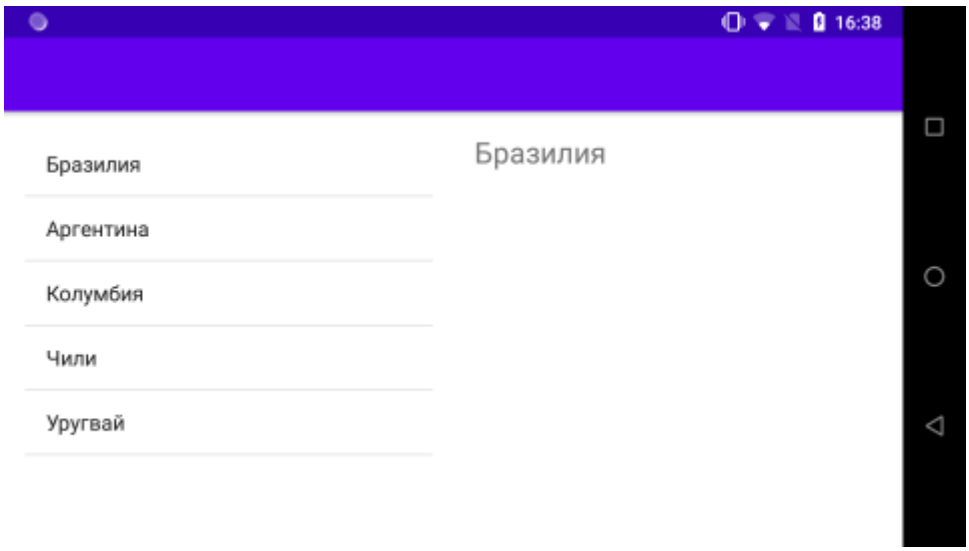
И также изменим главную activity - **MainActivity**:

```
package com.example.fragmentapp;


import androidx.appcompat.app.AppCompatActivity;


import android.content.Intent;
import android.os.Bundle;


public class MainActivity extends AppCompatActivity  implements
ListFragment.OnFragmentSendDataListener {
```

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

    @Override
    public void onSendData(String selectedItem) {
        DetailFragment fragment = (DetailFragment) getSupportFragmentManager()
                .findFragmentById(R.id.detailFragment);
        if (fragment != null && fragment.isVisible())
            fragment.setSelectedItem(selectedItem);
        else {
            Intent intent = new Intent(getApplicationContext(),
                    DetailActivity.class);
            intent.putExtra(DetailActivity.SELECTED_ITEM, selectedItem);
            startActivity(intent);
        }
    }
}
```

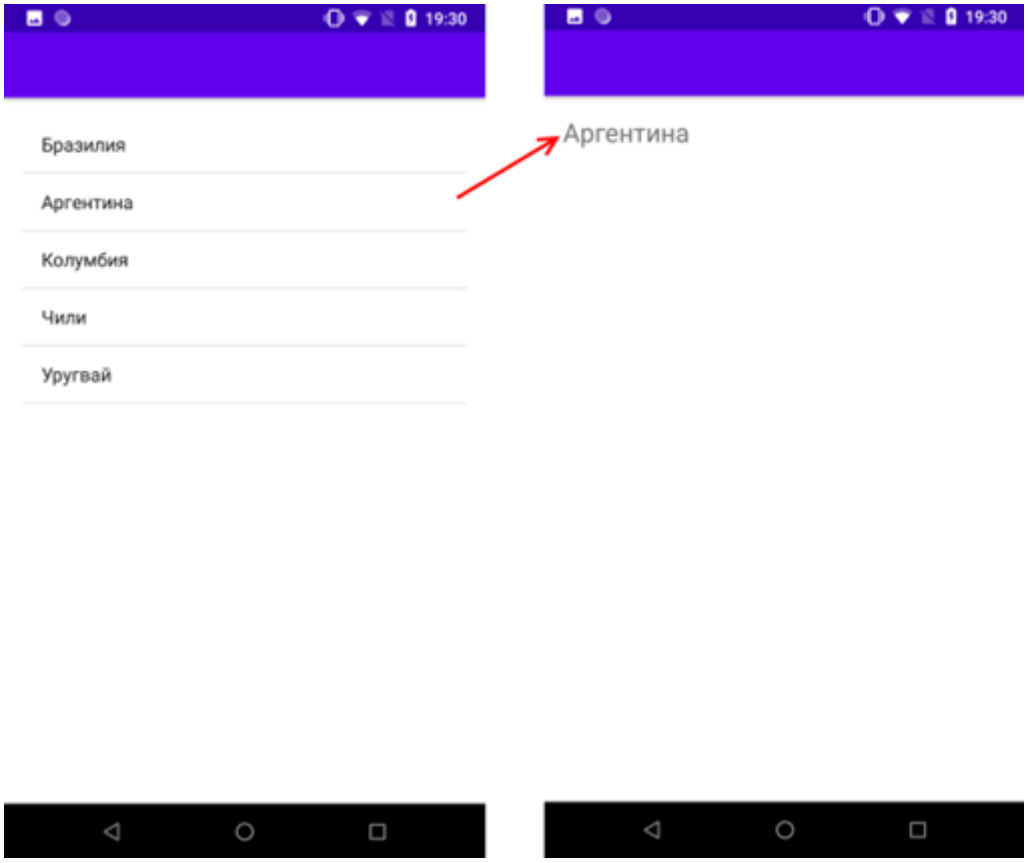
С помощью метода fragment.isVisible() мы можем узнать, активен ли определенный фрагмент в разметке интерфейса. Если фрагмента DetailFragment в данный момент не видим, то используется портретный режим, и поэтому запускается DetailActivity. Иначе идет работа с фрагментом внутри MainActivity, котора в альбомном режиме отображает сразу два фрагмента - ListFragment и DetailFragment.

Запустим приложение и перейдем в альбомный режим:



Фрагмент в альбомной ориентации в Android и Java

А при портретной ориентации экран будет выглядеть иначе:



Фрагмент в портретной ориентации в Android и Java

Задание

1. Реализовать добавление фрагмента в Activity
2. Реализовать добавление логики к фрагменту
3. Реализовать добавление фрагмента в коде
4. Реализовать взаимодействие между фрагментами
5. Реализовать фрагменты в альбомном и портретном режиме