

Часть 1. Передача данных в вызвавший открытие activity с помощью Activity Result API.

В прошлых темах было рассмотрено как вызывать новую Activity, передавать ей некоторые данные и возвращаться обратно. Но мы можем не только передавать данные запускаемой Activity, но и ожидать от нее некоторого результата работы.

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

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

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

```
ActivityResultLauncher<I> registerForActivityResult (  
    ActivityResultContract<I, O> contract,  
    ActivityResultCallback<O> callback)
```

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

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

Например:

```

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

            // обработка result

        }
    });

```

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

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

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

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

ПРИМЕР

Допустим у нас есть поле для ввода имени. После ввода имени на второй activity, мы можем выбрать запретить доступ, разрешить доступ или отменить операцию вовсе.

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

```
ity_main.xml × activity_new.xml × MainActivity.java × MainActivity.java ×
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}

2 usages
static final String ACCESS_MESSAGE="ACCESS_MESSAGE";

1 usage
ActivityResultLauncher<Intent> mStartForResult = registerForActivityResult
    (new ActivityResultContracts.StartActivityForResult(),
    new ActivityResultCallback<ActivityResult>() {
        @Override
        public void onActivityResult(ActivityResult result) {
            EditText editText = findViewById(R.id.name);
            if(result.getResultCode() == Activity.RESULT_OK){
                Intent intent = result.getData();
                String accessMessage = intent.getStringExtra(ACCESS_MESSAGE);
                editText.setText(accessMessage);
            }
            else{
                editText.setText("Ошибка доступа");
            }
        }
    });
});
```

Прежде всего, мы определяем объект **ActivityResultLauncher**, с помощью которого будем запускать вторую activity и передавать ей данные. Объект **ActivityResultLauncher** типизируется типом **Intent**, так как объект этого типа будет передаваться в метод **launch()** при запуске второй activity.

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

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

При обработке мы проверяем полученный код результата:

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

В качестве результата, как правило, применяются встроенные константы **Activity.RESULT_OK** и **Activity.RESULT_CANCELED**.

На уровне условностей **Activity.RESULT_OK** означает, что activity успешно обработала запрос, а **Activity.RESULT_CANCELED** – что activity отклонила обработку запроса.

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

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

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

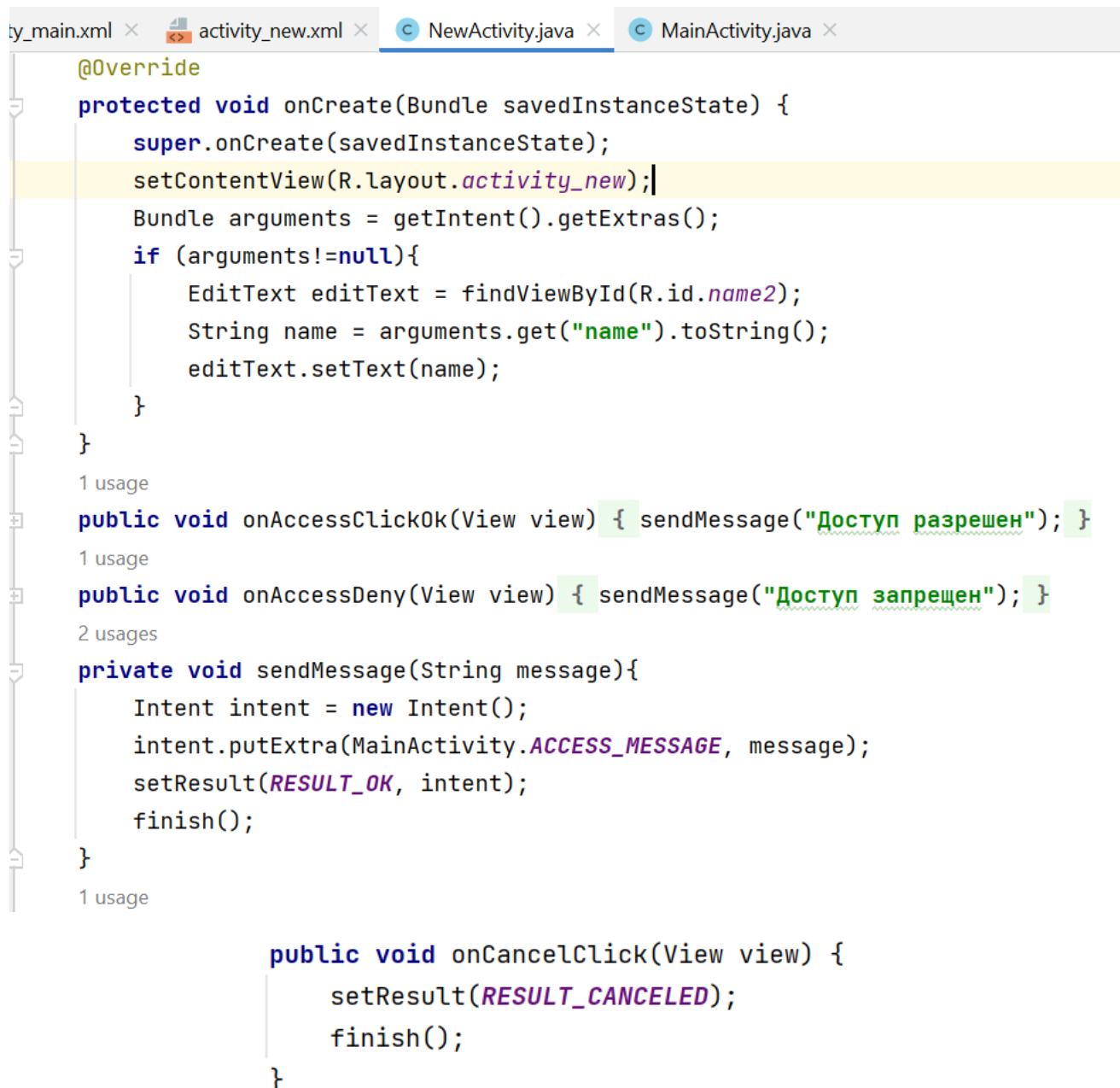
Таким образом, мы определили объект **ActivityResultLauncher**.

Далее в обработчике нажатия **onClick** с помощью этого объекта запускаем вторую activity.

```
public void onNextActivity(View view){  
    //Получаем введенное имя, которое будем проверять  
    EditText textName = findViewById(R.id.name);  
    String name = textName.getText().toString();  
    Intent intent = new Intent( packageContext: this, NewActivity.class);  
    intent.putExtra( name: "name", name);  
    mStartForResult.launch(intent);  
}
```

В обработчике нажатия кнопки **onClick()** получаем введенное в текстовое поле имя, добавляем его в объект **Intent** и запускаем **SecondActivity** с помощью метода **launch()**.

После этого определим логику работы второй **Activity**.



```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_new);
    Bundle arguments = getIntent().getExtras();
    if (arguments != null) {
        EditText editText = findViewById(R.id.name2);
        String name = arguments.get("name").toString();
        editText.setText(name);
    }
}

1 usage
public void onAccessClickOk(View view) { sendMessage("Доступ разрешен"); }
1 usage
public void onAccessDeny(View view) { sendMessage("Доступ запрещен"); }
2 usages
private void sendMessage(String message) {
    Intent intent = new Intent();
    intent.putExtra(MainActivity.ACCESS_MESSAGE, message);
    setResult(RESULT_OK, intent);
    finish();
}
1 usage

    public void onCancelClick(View view) {
        setResult(RESULT_CANCELED);
        finish();
    }

```

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

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

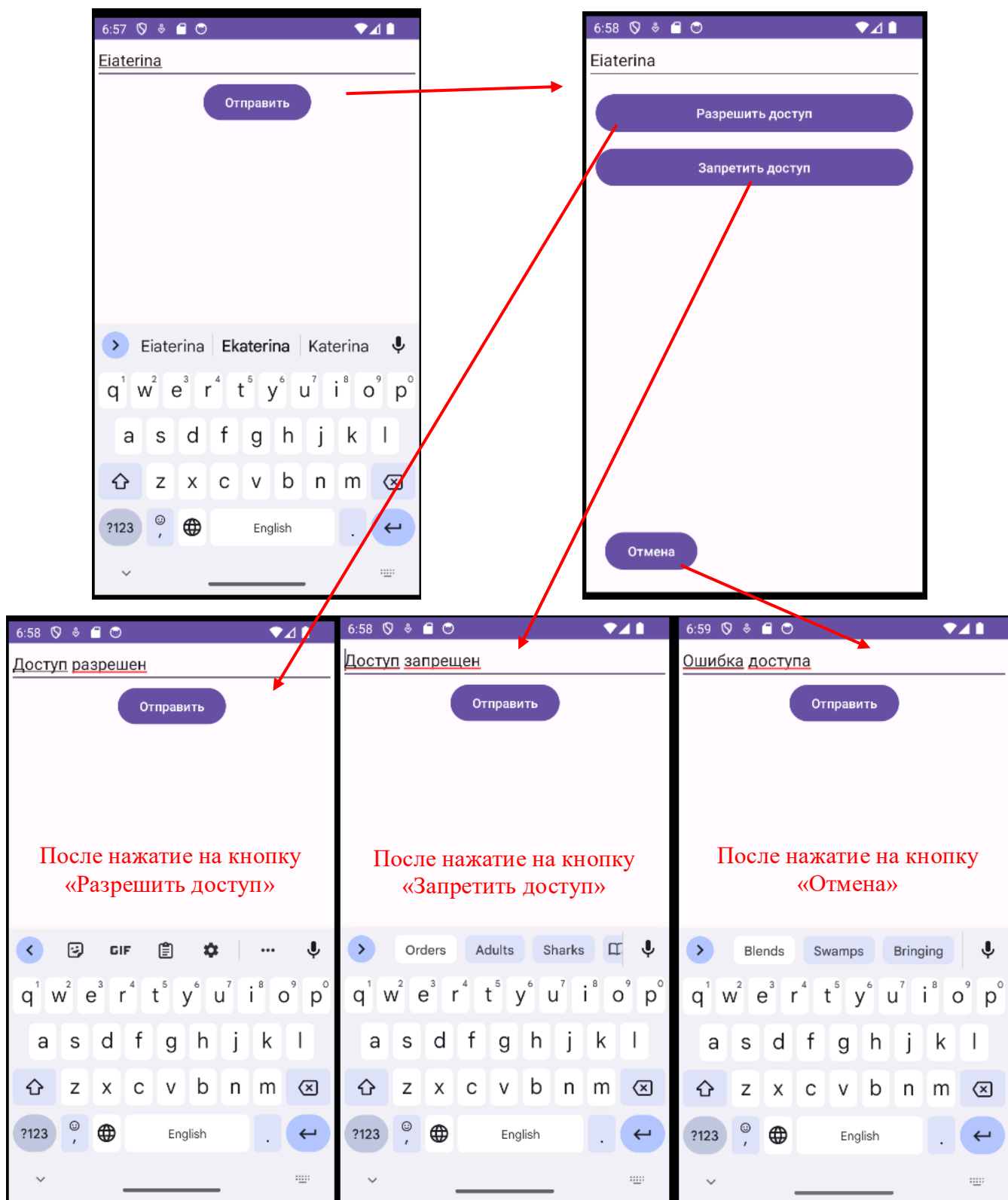
- числовой код результата;
- отправляемые данные.

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

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

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

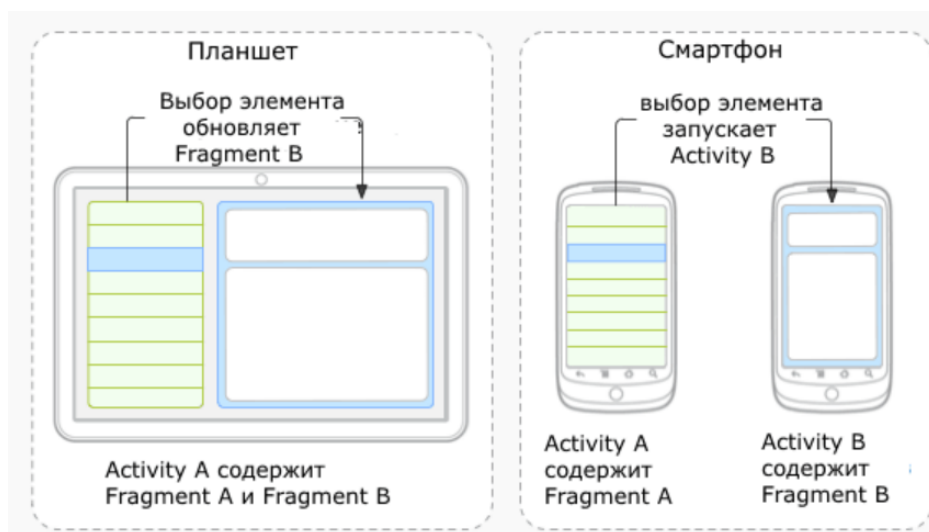
В зависимости от нажатой кнопки на SecondActivity мы будем получать разные результаты в MainActivity.



Часть 2. 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 представляют собой модульный сегмент пользовательского интерфейса в активности. Они используются для создания более гибких и адаптивных интерфейсов. Фрагменты могут быть добавлены, удалены, заменены или изменены в активности во время выполнения приложения. Это обеспечивает более динамичное и адаптивное взаимодействие с пользователем.

Среди основных характеристик фрагментов можно выделить:

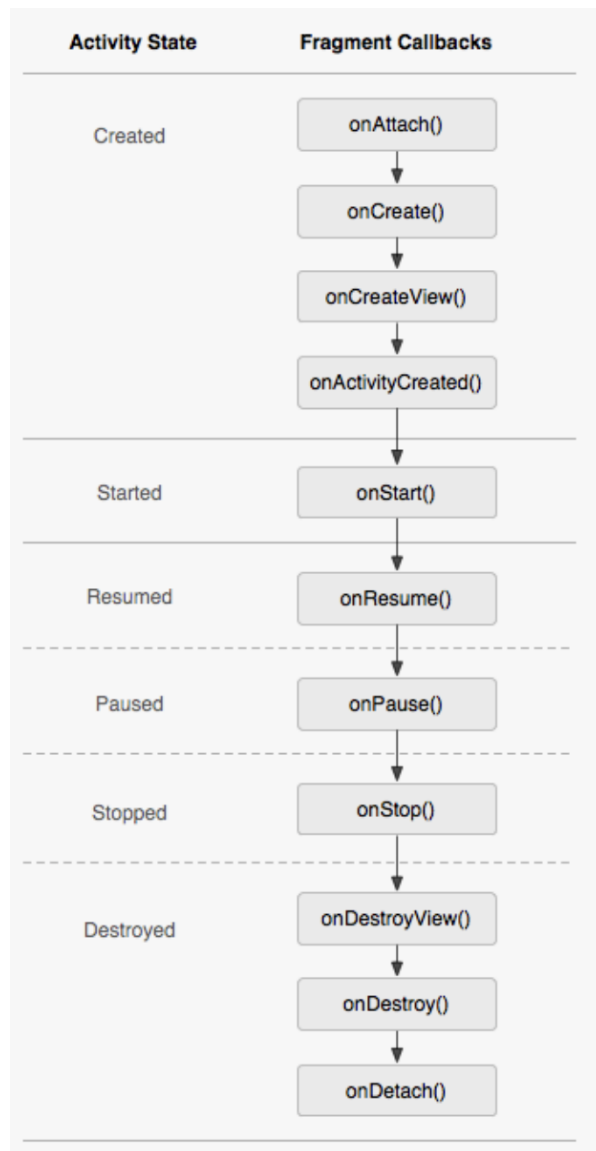
1. Модульность: Фрагменты позволяют разделить активность на несколько компонентов, каждый из которых имеет свой собственный жизненный цикл и обрабатывает свои собственные вводы. Это упрощает управление

различными частями интерфейса пользователя и повторное использование компонентов в разных активностях.

2. **Адаптивность:** Фрагменты помогают создавать интерфейсы, которые легко адаптируются к различным размерам экрана и ориентациям, что особенно важно для устройств с разными размерами экранов, таких как телефоны и планшеты.
3. **Управление жизненным циклом:** Каждый фрагмент имеет свой собственный жизненный цикл, но он тесно связан с жизненным циклом своей хост-активности. Это позволяет фрагментам управлять своим состоянием и поведением в зависимости от состояния активности.

Часть 3. Жизненный цикл фрагмента

Жизненный цикл фрагмента в Android представляет собой последовательность состояний, через которые проходит фрагмент во время своего существования. Эти состояния позволяют управлять поведением фрагмента на различных этапах его взаимодействия с пользователем и системой.



Привязка к активности: Когда фрагмент привязывается к активности, вызывается метод **onAttach()**. Это означает, что фрагмент теперь ассоциирован с активностью, и разработчик может взаимодействовать с ней.

Создание фрагмента: Вызов метода **onCreate()** означает, что создается объект фрагмента. Здесь можно инициализировать компоненты, необходимые фрагменту для функционирования, но при этом не связанные с графическим интерфейсом.

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
}
```

Создание представления фрагмента: Метод **onCreateView()** вызывается для создания представления фрагмента. Здесь загружается макет, определяет пользовательский интерфейс фрагмента. После создания представления оно возвращается системе для отображения.

```
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
saveInstanceState)
{
    return inflater.inflate(R.layout.fragment_first, container, false);
}
```

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

Активность фрагмента: После создания представления, метод **onActivityCreated()** вызывается, когда активность, к которой привязан фрагмент, полностью создана. Это хорошее место для выполнения финальных инициализаций, которые зависят от активности, например, настройка компонентов интерфейса или получение данных.

Запуск фрагмента: Когда фрагмент становится видимым для пользователя, вызывается метод **onStart()**. Здесь можно запускать анимации или выполнять задачи, которые должны быть видны пользователю.

```
public void onStart(){
    super.onStart();
}
```

Возобновление фрагмента: В этом состоянии фрагмент полностью активен и взаимодействует с пользователем. Метод **onResume()** вызывается, когда фрагмент готов к пользовательскому взаимодействию.

Приостановка фрагмента: Когда фрагмент перестает взаимодействовать с пользователем, система вызывает метод **onPause()**. Это происходит, например, при переключении на другой фрагмент или активность.

Остановка фрагмента: Метод **onStop()** вызывается, когда фрагмент больше не виден пользователю. В этом состоянии можно освободить ресурсы, которые не нужны, пока фрагмент не активен.

Уничтожение представления фрагмента: Перед удалением фрагмента из активности или при его замене, метод **onDestroyView()** вызывается для очистки ресурсов, связанных с пользовательским интерфейсом фрагмента.

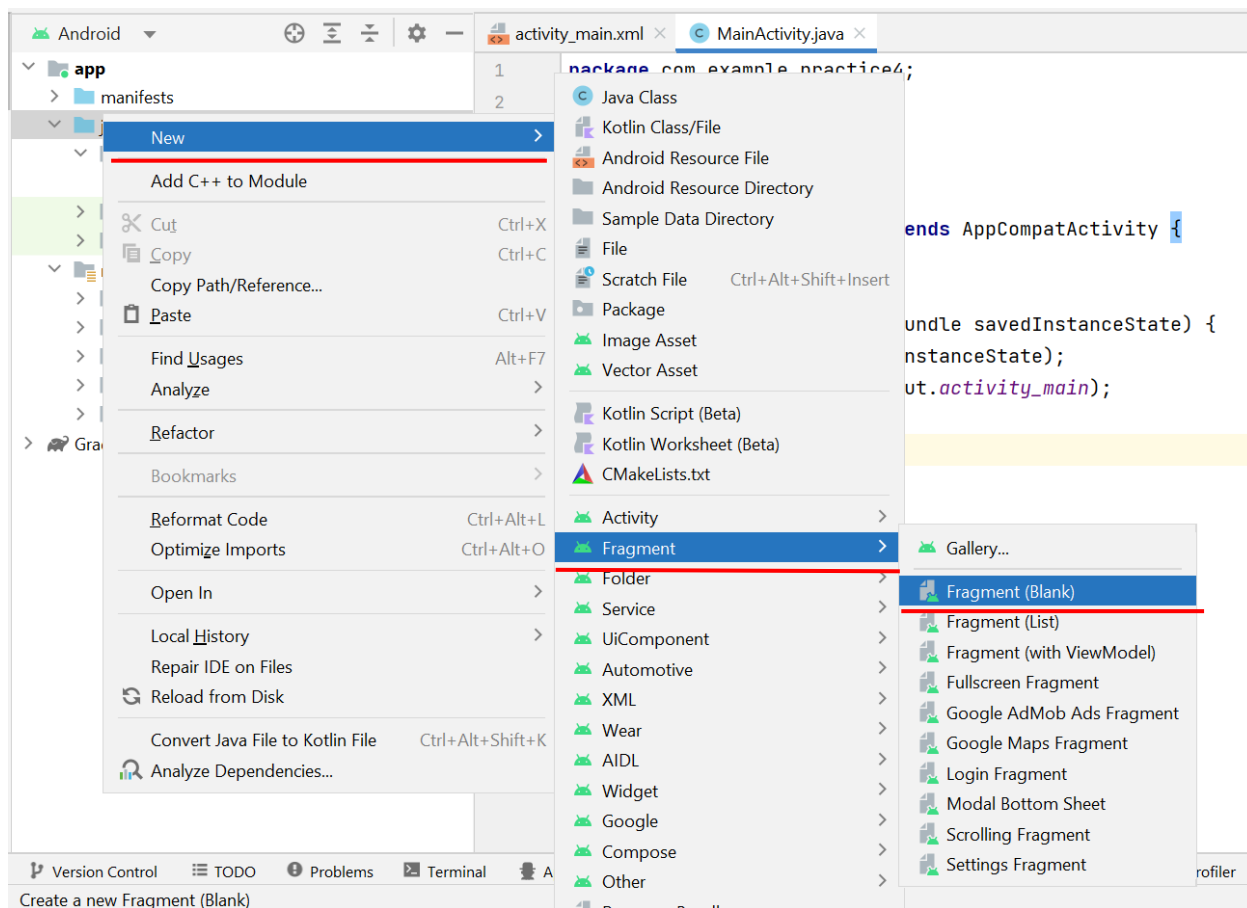
Уничтожение фрагмента: На этом этапе вызывается метод **onDestroy()**, который сигнализирует о том, что объект фрагмента скоро будет уничтожен. Это последний шанс для освобождения оставшихся ресурсов.

Открепление фрагмента от активности: Последний этап в жизненном цикле фрагмента — это его открепление от активности. Метод **onDetach()** вызывается, когда фрагмент отсоединяется от активности, что означает полное удаление фрагмента.

Часть 4. Создание и размещение фрагментов

Можно добавить по отдельности класс Java, который представляет фрагмент, и файл xml для хранения в нем разметки интерфейса, который будет использовать фрагмент, однако Android Studio представляет готовый шаблон для добавления фрагмента. Этим способ и будем пользоваться

Принцип создания новых фрагментов схож с созданием активностей. Нужно выбрать модуль "app", затем в верхней панели выбрав **"File" → "New" → "Fragment"** и выбрать тип активности, например "Fragment (Blank)" либо нажать правой кнопкой мыши на **"java" → "New" → "Fragment"**, после чего выбрать название фрагмента и как в случае с активностью будут созданы файлы класса и разметки.



После создания фрагмента, класс для реализации фрагмента и xml-файл для разметки будут созданы автоматически.

Класс фрагмента должен наследоваться от класса **Fragment**. Чтобы указать, что фрагмент будет использовать определенный xml-файл layout, идентификатор ресурса

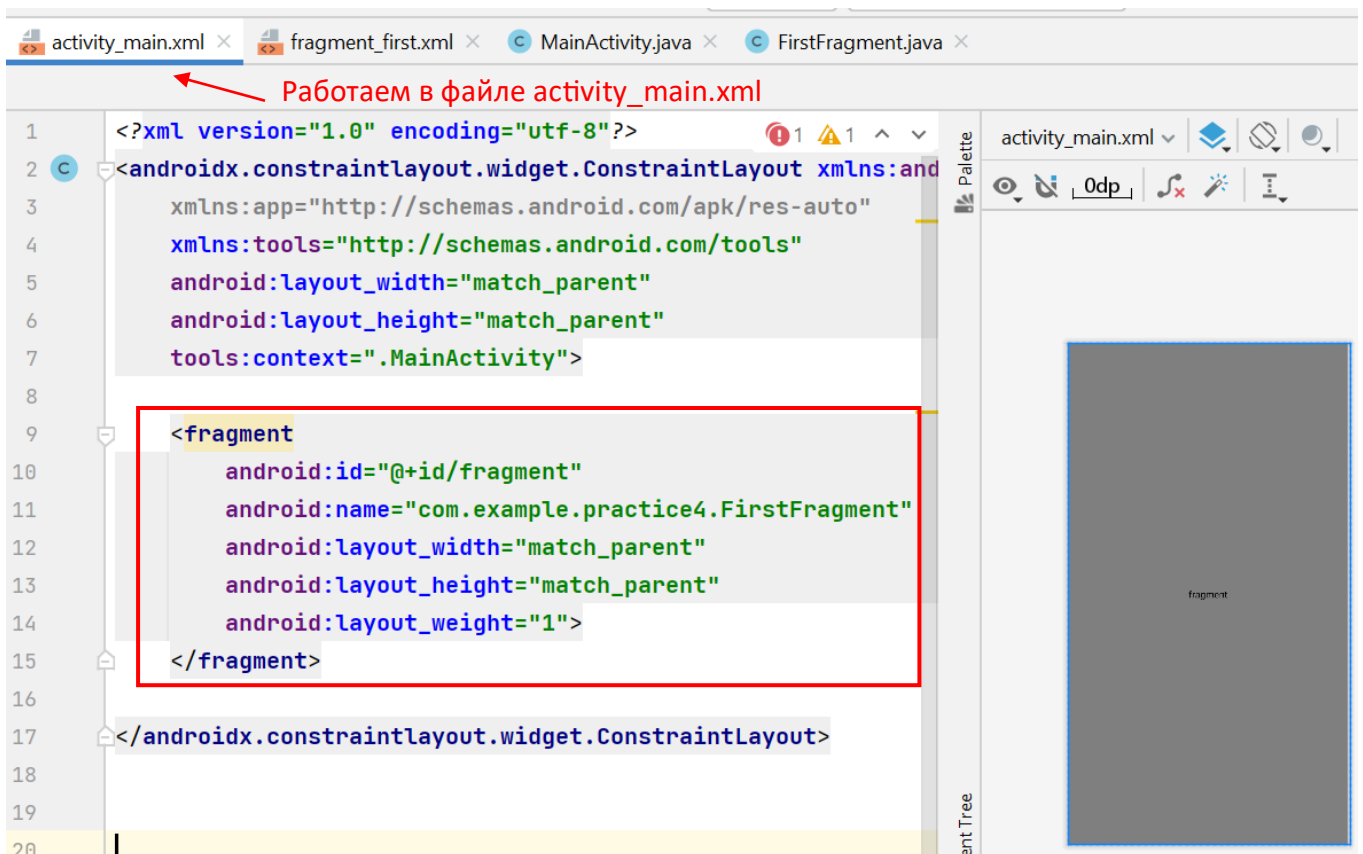
layout передается в вызов конструктора родительского класса (то есть класса `Fragment`).



Теперь после создания фрагмента его необходимо разместить на активности. Сделать это можно тремя способами: **статически**, **динамически** и **через элемент `fragmentContainerView`**.

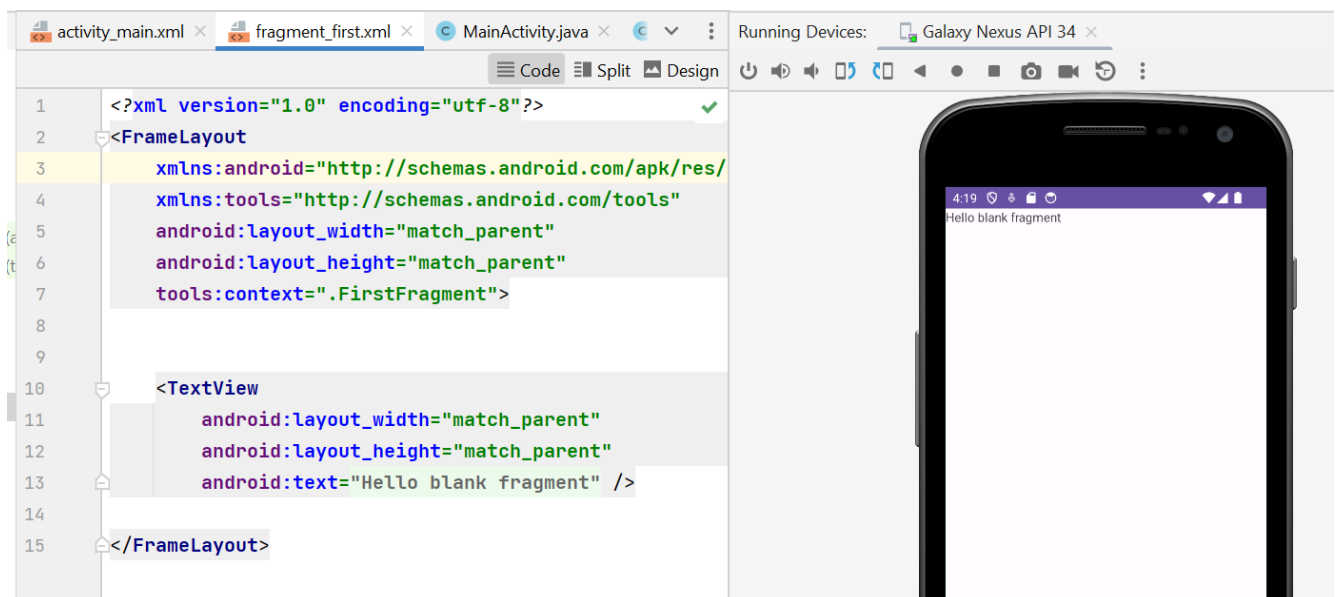
Для размещения фрагмента **статическим способом** необходимо добавить в XML файл активности элемент `fragment`. Например:

```
<fragment
    android:id="@+id/fragment"
    android:name="com.example.practice4.FirstFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_weight="1">
</fragment>
```



Атрибут **android:name** указывает на полное имя класса фрагмента с учетом пакета, который будет использоваться.

В качестве примера во фрагменте было определено текстовое поле с некоторым текстовым значением.



Теперь данный фрагмент будет показываться при запуске приложения.

Главным недостатком такого размещения фрагмента заключается в том, что статические фрагменты тесно связаны с жизненным циклом активности, что может усложнить управление их состоянием. Например, сохранение и восстановление

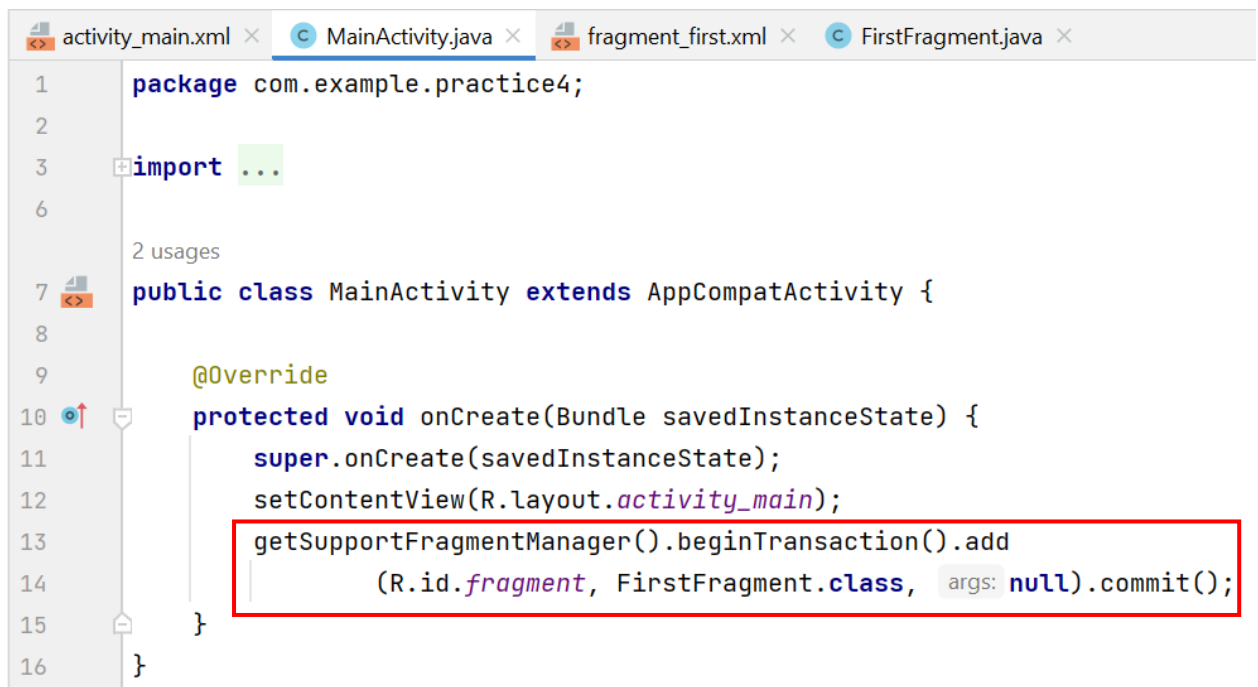
состояния фрагмента при пересоздании активности может потребовать дополнительной логики.

Все преимущества фрагментов раскрываются при их **динамическом изменении** в процессе работы приложения. Фрагменты, при динамическом размещении, управляются аналогично обычным View элементам. Для облегчения управления, каждый фрагмент размещается в отдельном контейнере. Обычно для этой цели выбираются контейнеры типа `FrameLayout`.

```
<FrameLayout
    android:id="@+id/fragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Далее в классе активности необходимо «разместить» данный фрагмент, используя метод класса **"FragmentManager"** — **"getSupportFragmentManager"**.

```
getSupportFragmentManager().beginTransaction().add(R.id.fragment,
FirstFragment.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:

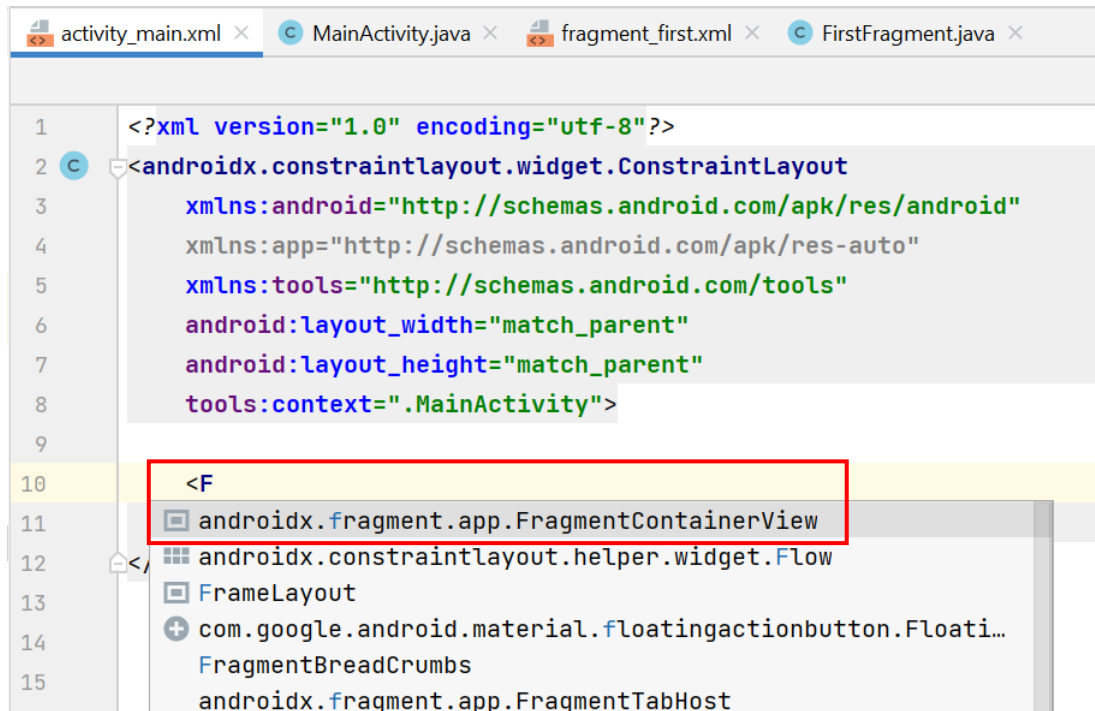
```
add(R.id.fragment_container_view, new ContentFragment());
```

И метод `commit()` подтверждает и завершает операцию добавления.

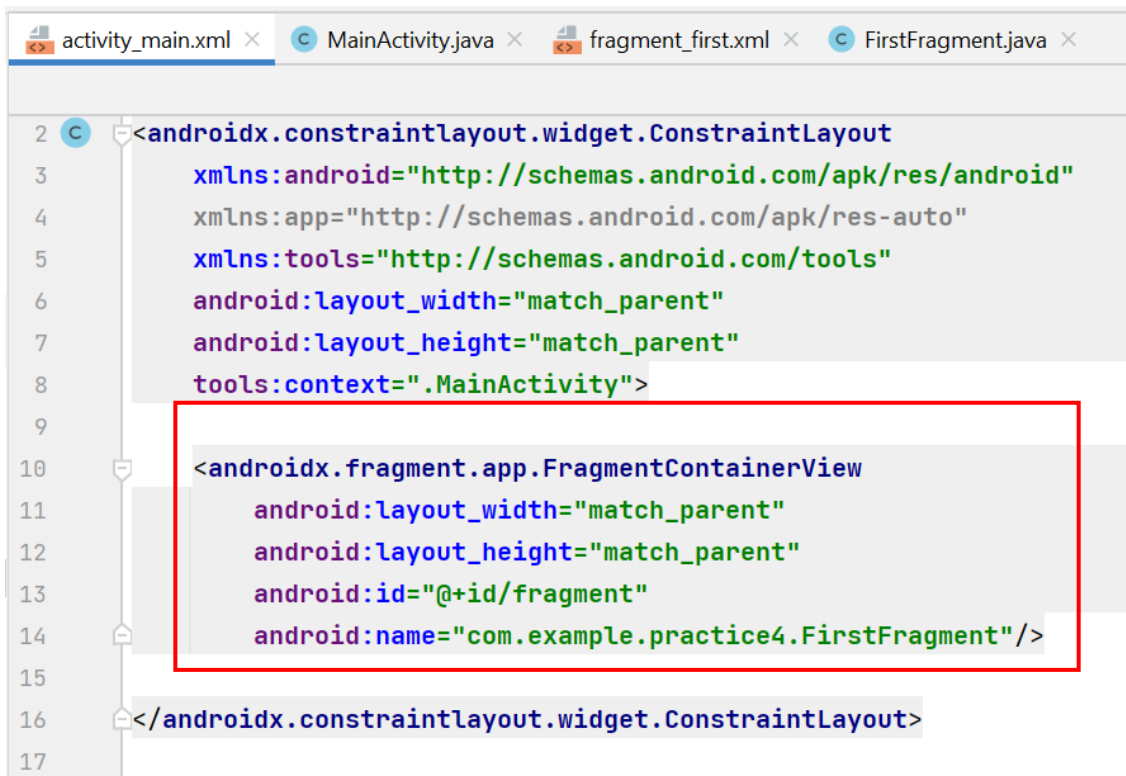
Последний способ размещения фрагментов является ключевым элементом современного подхода к управлению фрагментами в приложении. **FragmentManager** — это специализированный контейнер для фрагментов, который предлагает улучшенную замену `FrameLayout` при работе с фрагментами. Он более оптимизирован для работы с `FragmentManager` и предоставляет дополнительные возможности и преимущества.

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

Для его использования нужно разместить данный элемент в XML файле активности и в открывшемся окне выбрать нужный фрагмент, и он автоматически будет размещен внутри `FragmentManager`.



```
1 <?xml version="1.0" encoding="utf-8"?>
2 <androidx.constraintlayout.widget.ConstraintLayout
3     xmlns:android="http://schemas.android.com/apk/res/android"
4     xmlns:app="http://schemas.android.com/apk/res-auto"
5     xmlns:tools="http://schemas.android.com/tools"
6     android:layout_width="match_parent"
7     android:layout_height="match_parent"
8     tools:context=".MainActivity">
9
10    <F
11    androidx.fragment.app.FragmentContainerView
12    androidx.constraintlayout.helper.widget.Flow
13    FrameLayout
14    com.google.android.material.floatingactionbutton.Floati...
15    FragmentBreadcrumbs
16    androidx.fragment.app.FragmentTabHost
```



```
2 <androidx.constraintlayout.widget.ConstraintLayout
3     xmlns:android="http://schemas.android.com/apk/res/android"
4     xmlns:app="http://schemas.android.com/apk/res-auto"
5     xmlns:tools="http://schemas.android.com/tools"
6     android:layout_width="match_parent"
7     android:layout_height="match_parent"
8     tools:context=".MainActivity">
9
10    <androidx.fragment.app.FragmentContainerView
11        android:layout_width="match_parent"
12        android:layout_height="match_parent"
13        android:id="@+id/fragment"
14        android:name="com.example.practice4.FirstFragment"/>
15
16 </androidx.constraintlayout.widget.ConstraintLayout>
17
```

Часть 5. Навигация

Навигация между фрагментами осуществляется с помощью `FragmentManager`, который управляет операциями, такими как добавление, удаление или замена фрагментов в контейнере. Ключевым понятием в навигации между фрагментами

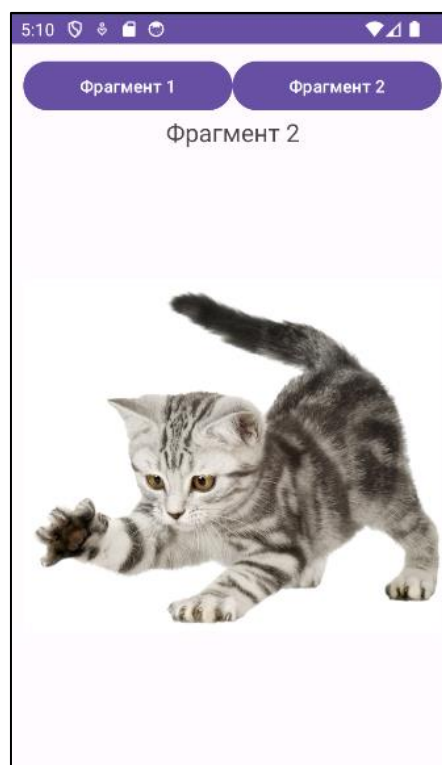
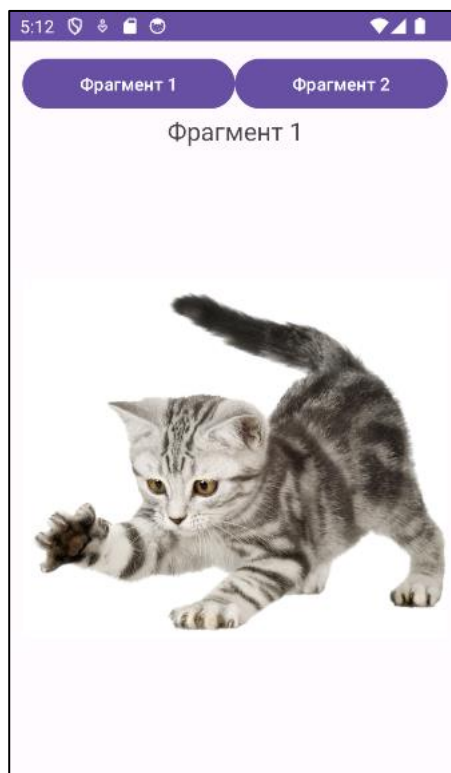
является транзакция. Транзакция фрагмента — это серия действий, которые выполняются вместе. Эти действия могут включать добавление, удаление и замену фрагментов, а также добавление их в стек возврата, чтобы пользователь мог вернуться обратно, используя кнопку «Назад».

Для управления навигацией между фрагментами можно использовать программные методы, такие как вызовы **"replace"** и **"commit"** класса **FragmentManager**. Для этого нужно создать объект того фрагмента куда планируется осуществить перемещение и добавить его в метод **replace** и сохранить изменения через метод **commit**.

```
FragmentManager fragmentManager = getSupportFragmentManager();
FirstFragment fragment1 = new FirstFragment();
FragmentManager fragmentManager.beginTransaction()
    .replace(R.id.container, fragment1, "fragment1");
fragmentManager.commit();
```

Где **FirstFragment** – это название java класса первого фрагмента, а **container** – идентификатор **FrameLayout**, который расположен в разметки **activity_main**.

Тогда при нажатии на кнопку будут меняться используемые фрагменты.



Задание

1. Создать три активности. На первой Activity расположить поле для ввода фамилии и имени и кнопку Next для запуска второго Activity. При нажатии кнопки выполнить передачу данных.

Во втором Activity создать два текстовых поля (TextView) для вывода переданной информации о пользователе (имя и фамилия), пустое по умолчанию текстовое поле (TextView) для ввода предмета (на который хотели бы дополнительно записаться), кнопку Ввести информацию.

В третьем Activity создать 3 редактируемых текстовых полей (EditText) для ввода информации о днях посещения (день, время, комментарии), кнопку ОК для возврата во второе Activity. При нажатии кнопки ОК реализовать возврат во второе Activity с передачей в качестве результата информации о предпочтительном времени организации дополнительного занятия. Вывести всплывающее сообщение о том, что время занятия успешно передано.

По желанию можно добавить еще поля, кнопки или другие элементы управления, изменить тему и т.д.

2. Создать новый проект. Разместить три фрагмента с разным наполнением тремя способами (статически, динамически и через элемент fragmentManagerView).

3. Сделать перемещение между созданными фрагментами.

Источники

- 1) <https://developer.android.com/guide/fragments>
- 2) <https://metanit.com/java/android/8.1.php?ysclid=lsrgj49f17519532211>
- 3) <https://metanit.com/java/android/8.3.php>
- 4) <https://developer.alexanderklimov.ru/android/fragment.php?ysclid=lsrju60aoa610185033>
- 5) https://developer.alexanderklimov.ru/android/fragment_translate_cat.php?ysclid=lsrjgx1pd464423823
- 6) <https://developer.alexanderklimov.ru/android/fragment4.php>