



Занятие 2.5. Лекция

СОЗДАНИЕ ПОЛЬЗОВАТЕЛЬСКИХ
ИНТЕРФЕЙСОВ В ANDROID-ПРИЛОЖЕНИЯХ

ЧАСТЬ 2



Учебные вопросы

1. Объект Intent. Явные и неявные намерения
2. Fragments в Android



1. Объект Intent. Явные и неявные намерения

Понятие Intent

Intent (Намерение) представляет собой объект описания операции, которую необходимо выполнить через систему Android.

Примеры операций: выбрать фотографию, отправить письмо, сделать звонок, запустить браузер и перейти по указанному адресу.

В SDK представлен классом **android.content.Intent**.

Где используется Intent

Намерения (Intent) в Android используются в качестве механизма передачи сообщений, который может работать как внутри одного приложения, так и между приложениями.

Намерения могут применяться для:

- ☐ запуска другой activity в вашем приложении;
- ☐ запуска внешних приложений;
- ☐ обмена данными между компонентами приложения;
- ☐ запуска фонового сервиса (Service);
- ☐ отправки широковещательных сообщений (Broadcast Receiver), чтобы уведомить другие приложения о произошедших событиях.

Запуск другой activity

1. Регистрация новой activity в AndroidManifest.xml

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

Запуск другой activity

2. Создание объекта Intent

```
Intent actIntent = new Intent(getApplicationContext(), SecondActivity.class);
```

android.content.Context



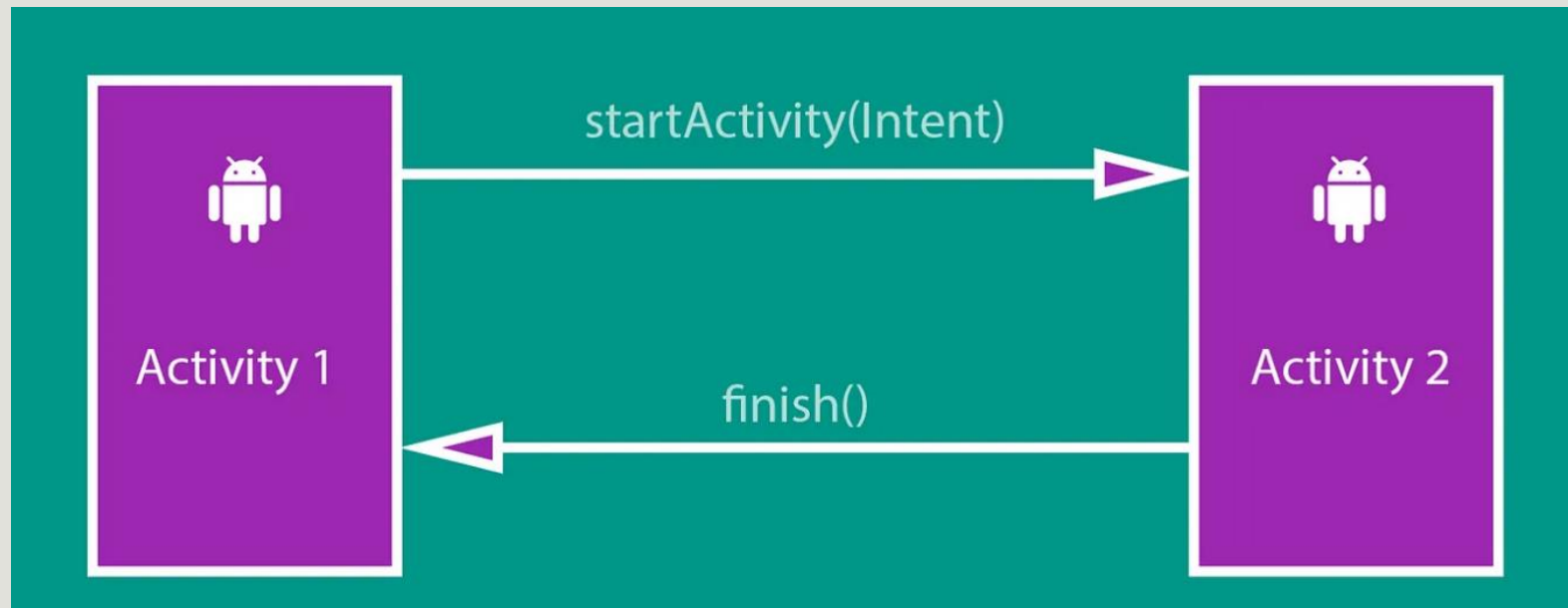
java.lang.Class



Запуск другой activity

3. Непосредственно, вызов метода запуска

```
Intent actIntent = new Intent(getApplicationContext(), SecondActivity.class);  
startActivity(actIntent);
```



Запуск другой activity с получением результата

Вызов метода запуска activity:

```
Intent actIntent = new Intent(getApplicationContext(), SecondActivity.class);  
startActivityForResult(actIntent, requestCode: 12345);
```



Запуск другой activity с получением результата

Формирование результата, который нужно вернуть:

```
Intent intent = new Intent();  
intent.putExtra(name: "data", value: "some string data");  
intent.putExtra(name: "val", value: 1);  
setResult(resultCode: 12345, intent);  
finish();
```

Запуск другой activity с получением результата

Обработка полученного результата:

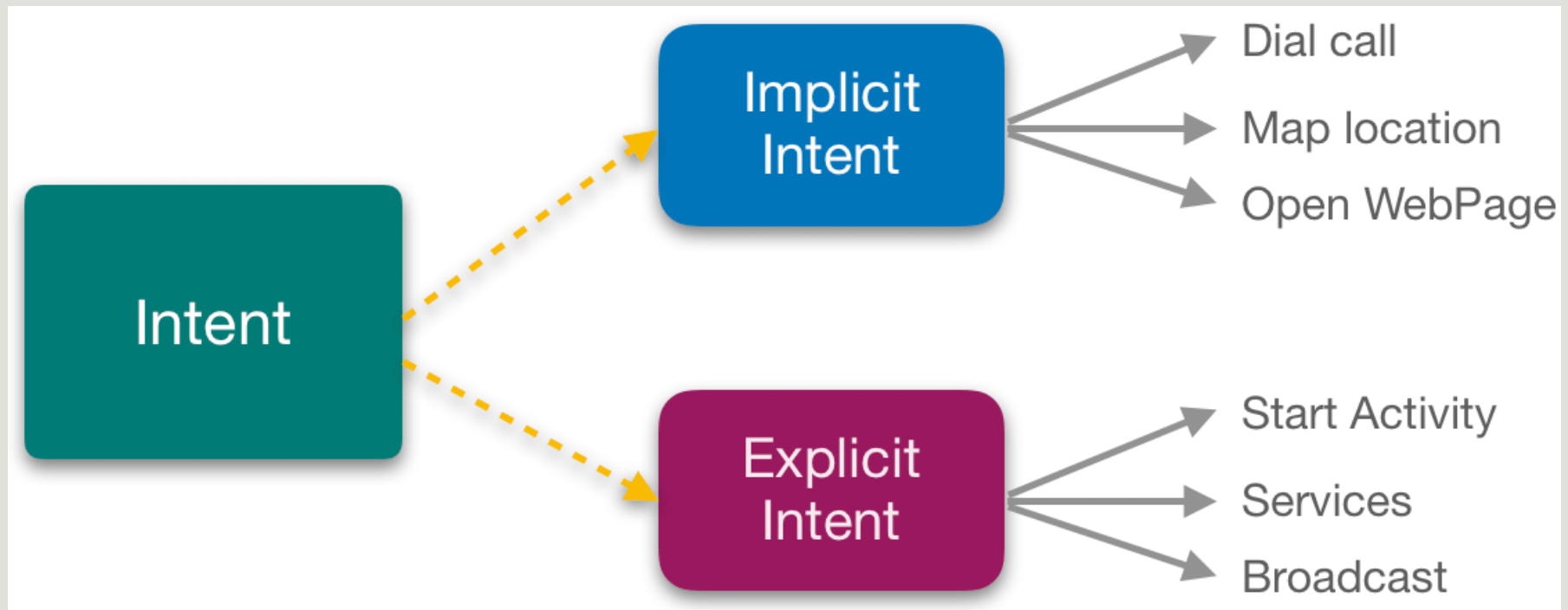
```
@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if(requestCode == 12345){
        if(data != null){
            // обработка полученных данных
            String str = data.getStringExtra(name: "data");
        }
    }
}
```

Запуск другой activity с получением результата. Что теперь предлагает Google

```
Intent actIntent = new Intent(getApplicationContext(), SecondActivity.class);
//startActivityForResult(actIntent, 12345); не используем устаревший метод
ActivityResultLauncher<Intent> activityResultLaunch = registerForActivityResult(
    new ActivityResultContracts.StartActivityForResult(),
    new ActivityResultCallback<ActivityResult>() {
        @Override
        public void onActivityResult(ActivityResult result) {
            if (result.getData() != null) {
                // обработка полученных данных
                String str = result.getData().getStringExtra(name: "data");
            }
        }
    });
activityResultLaunch.launch(actIntent); // запуск новой activity
```

Типы намерений

- ❑ явные (Explicit intent)
- ❑ неявные (Implicit Intent)



Неявные объекты Intent

- ❑ не содержат имени конкретного компонента;
- ❑ вместо этого включают действие (**action**), которое требуется выполнить;
- ❑ дополнительно может включать: наименование категории (**category**) и тип данных (**data**);

Intent Filter – это набор параметров: action, data, category.

Неявные объекты Intent

- ❑ не содержат имени конкретного компонента;
- ❑ вместо этого включают действие (**action**), которое требуется выполнить;
- ❑ дополнительно может включать: наименование категории (**category**) и тип данных (**data**);

Intent Filter – это набор параметров: action, data, category.

Action (Действие)

Определяет, какое конкретно действие необходимо выполнить. Действие для объекта Intent можно указать методом **setAction** или определить в конструкторе Intent.

Примеры:

- ❑ Intent.ACTION_DIAL – передача номера телефона для звонка;
- ❑ MediaStore.ACTION_IMAGE_CAPTURE – вызов приложения для фотосъемки;
- ❑ Intent.ACTION_SEND – загружает экран для отправки данных, указанных в намерении;
- ❑ Intent.ACTION_VIEW – отображение данных.

Category (Категория)

Категория – это строка, содержащая дополнительные сведения о том, каким компонентом должна выполняться обработка объекта Intent. В объект Intent можно поместить любое количество категорий. Однако большинству объектов Intent описание категории не требуется. Класс Intent для работы с категориями имеет группу методов :

- ❑ `addCategory()` – добавить категорию в объект Intent;
- ❑ `removeCategory()` – удалить ранее добавленную категорию из объекта Intent;
- ❑ `getCategories()` – получить набор категорий объекта Intent.

Наиболее распространенное значение: `android.intent.category.DEFAULT`

Данные (data) намерения

При создании объекта Intent в некоторых случаях необходимо определить данные (data). Данные представляют собой специальный объект **Uri**, который можно добавить к intent с помощью метода **setData**. Тип данных (MIME) можно определить методом **setType**. При необходимости определения обоих параметров можно вызвать метод **setDataAndType**.



Структура URI

```
<scheme>://<host>:<port>/[<path> | <pathPrefix> | <pathPattern>]
```

- ❑ scheme (схема) – режим Uri, например: http, ftp, file, content, tel, geo
- ❑ host – адрес хоста (при наличии)
- ❑ port – порт хоста (при наличии)
- ❑ path/pathPrefix/pathPattern – указывают информацию о пути

Примеры создания URI

- ❑ `Uri u = Uri.parse("http://developer.android.com/reference/android/net/Uri.html")`
- ❑ `Uri u2 = Uri.parse("geo:55.754283,37.62002")`
- ❑ `Uri u3 = Uri.parse("tel:89191234567")`
- ❑ `Uri u4 = Uri.parse("file:///sdcard/Download/practical_2.8.docx")`
- ❑ `Uri u5 = Uri.parse("mailto:support@mirea.ru")`

Для добавления в Intent дополнительных данных, не подлежащих форматированию через Uri (например, текст), необходимо использовать метод **putExtra**.

MIME-типы

Internet Media Types (MIME-типы) – типы данных, которые могут быть переданы посредством сети Интернет с применением стандарта MIME.

MIME (Multipurpose Internet Mail Extensions) – стандарт, описывающий передачу различных типов данных по электронной почте, а также, в общем случае, спецификация для кодирования информации и форматирования сообщений таким образом, чтобы их можно было пересылать по Интернету.

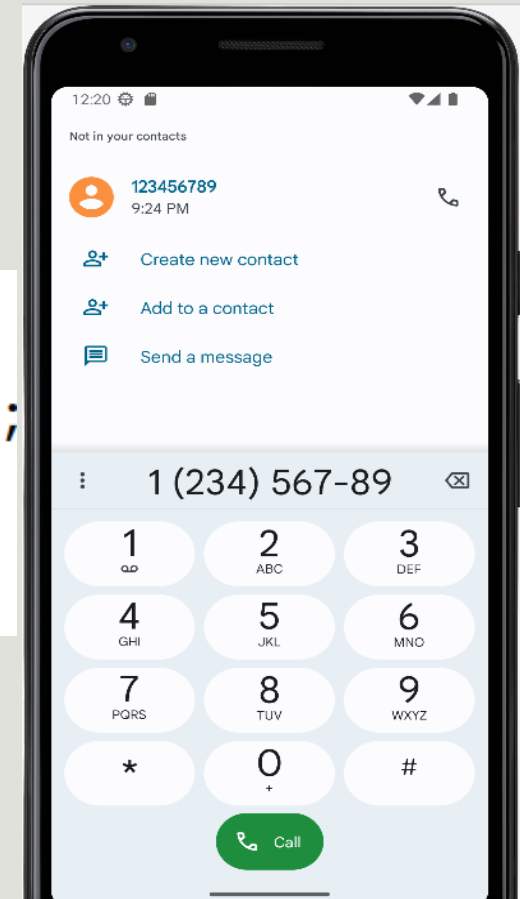
MIME-типы

- ☐ application;
- ☐ audio;
- ☐ example;
- ☐ image;
- ☐ message;
- ☐ model;
- ☐ multipart;
- ☐ text;
- ☐ video

Уточняющий формат определяется через символ «/». Например:
image/jpeg, **text/html**, **video/mp4**.

Пример использования неявного намерения

```
Uri uriNumber = Uri.parse("tel:123456789");  
Intent callIntent = new Intent(Intent.ACTION_DIAL);  
callIntent.setData(uriNumber);  
startActivity(callIntent);
```

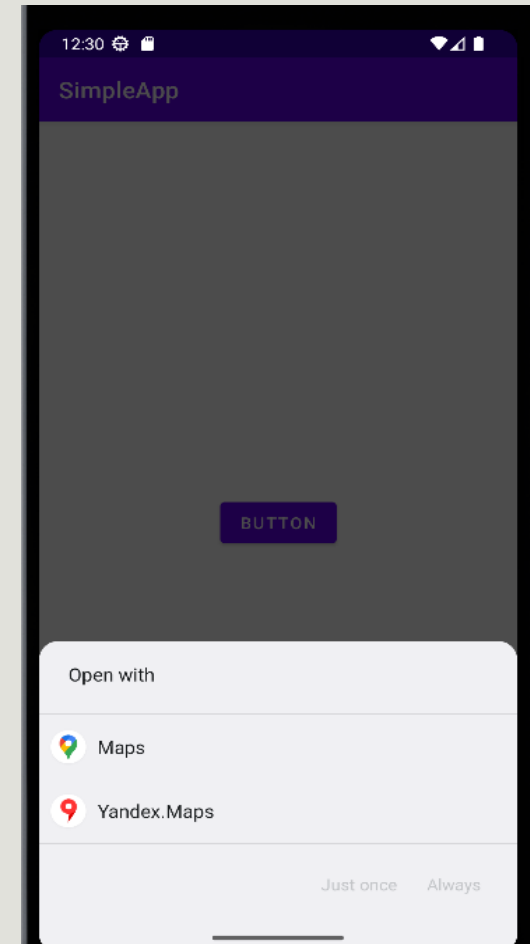


Заглянем в AndroidManifest.xml этого приложения

```
<activity android:name=".DialtactsActivity"
    android:label="@string/launcherDialer"
    android:theme="@style/DialtactsTheme"
    android:launchMode="singleTask"
    android:icon="@mipmap/ic_launcher_phone"
    android:screenOrientation="portrait">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <action android:name="android.intent.action.DIAL" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:scheme="tel" />
    </intent-filter>
</activity>
```

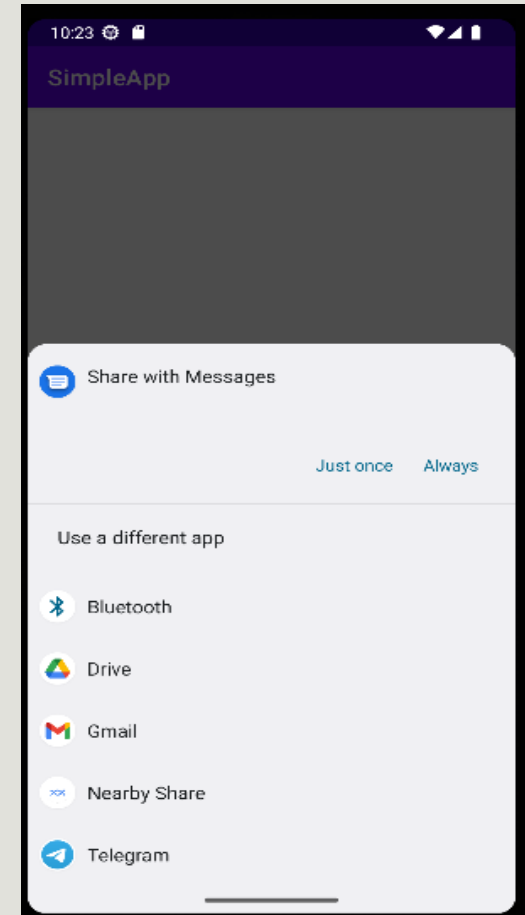

Пример использования неявного намерения

```
Uri uriLocation = Uri.parse("geo:55.754283,37.62002");  
Intent callIntent = new Intent(Intent.ACTION_VIEW);  
callIntent.setData(uriLocation);  
startActivity(callIntent);
```



Пример использования неявного намерения

```
Intent sendIntent = new Intent();  
sendIntent.setAction(Intent.ACTION_SEND);  
sendIntent.putExtra(Intent.EXTRA_TEXT, value: "This is my text to send");  
sendIntent.setType("text/plain");  
startActivity(sendIntent);
```





2. Fragments в Android

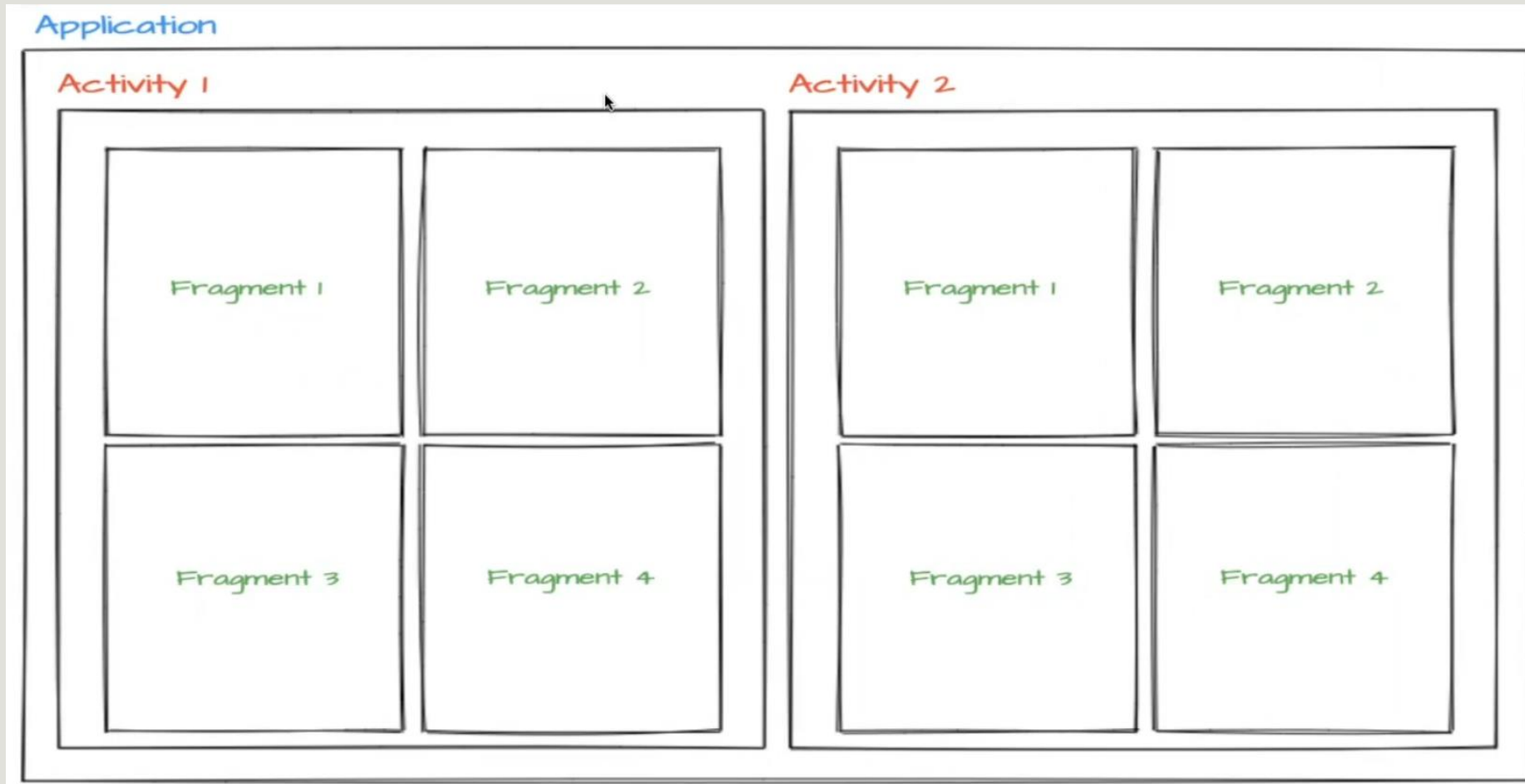
Понятие фрагментов

Фрагменты (fragments) – это UI-модули, используемые для разделения интерфейса пользователя на отдельные части, которые могут быть использованы в одной или разных activities.

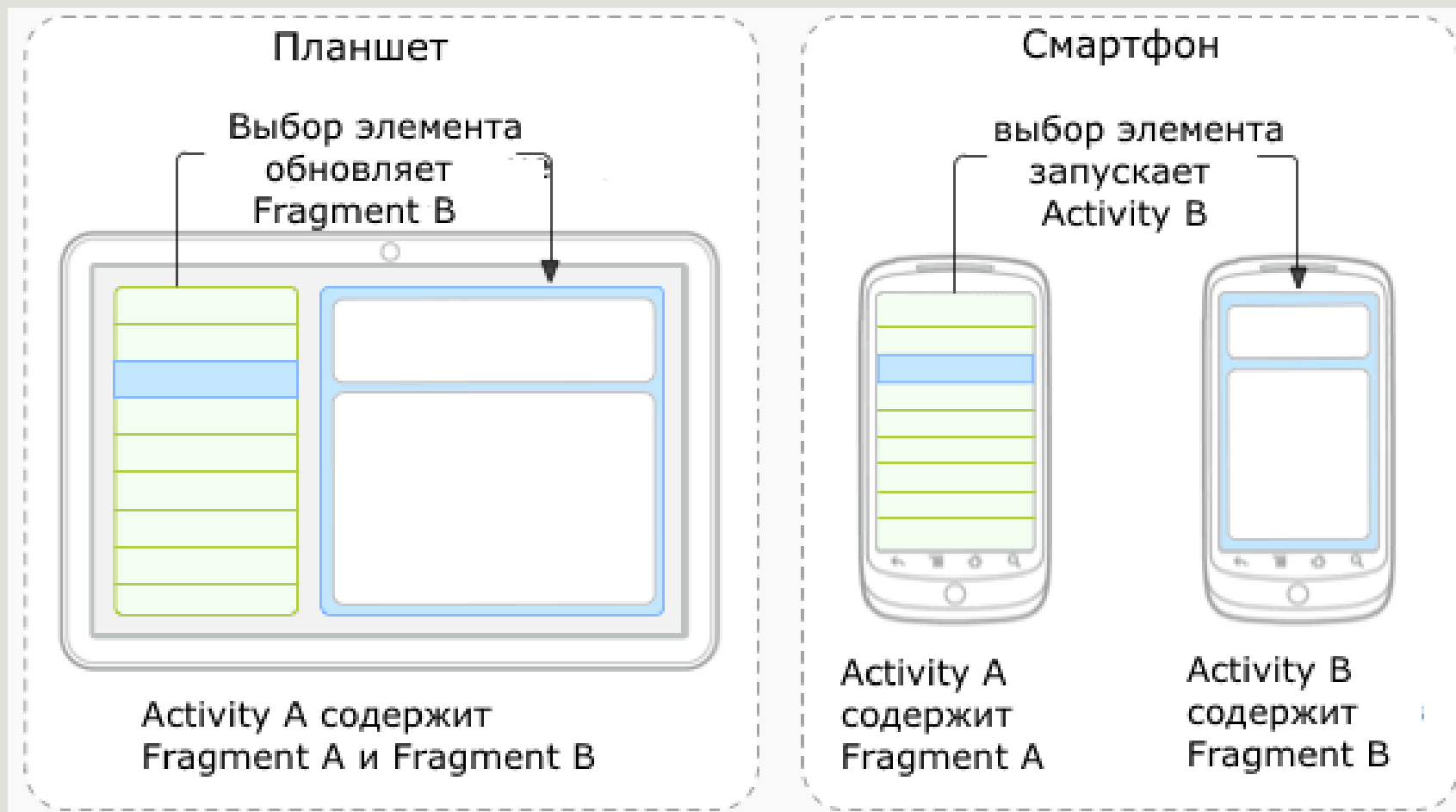
Фрагмент – это:

- ❑ класс (`androidx.fragment.app.Fragment`);
- ❑ контейнер для любых View-объектов, которые могут быть показаны пользователю;
- ❑ продолжение Activity, от которого Fragment получает всю информацию об изменениях в жизненном цикле.

Понятие фрагментов



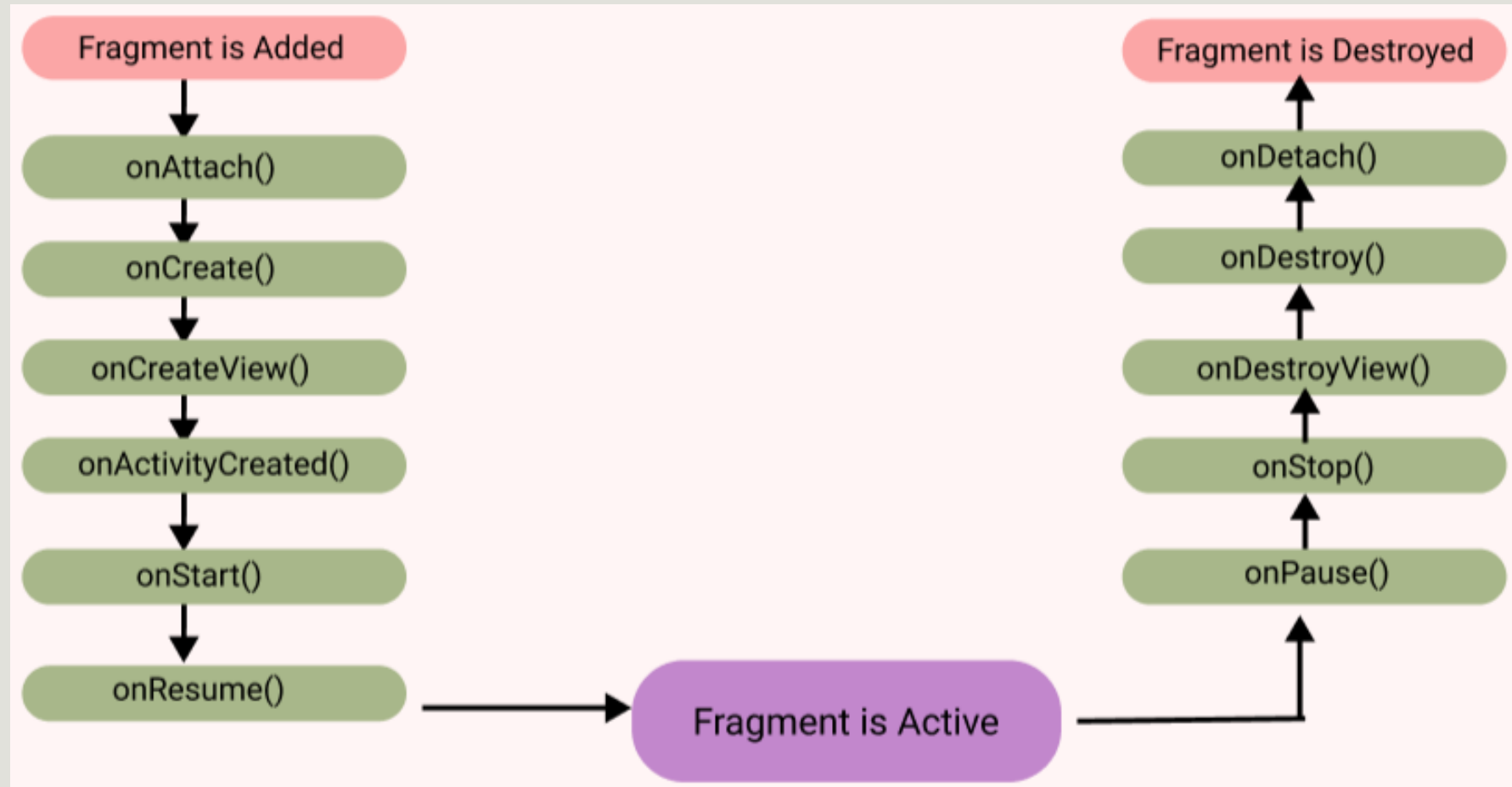
Понятие фрагментов



Преимущества использования фрагментов

- ☐ гибкость
- ☐ переиспользуемость
- ☐ масштабируемость
- ☐ модульность

Жизненный цикл фрагментов



Способы создания фрагментов

- ☐ С использованием Layout XML
- ☐ Программно в Java (Kotlin) коде

Использование Layout XML для работы с фрагментами

```
android:layout_height="match_parent"
tools:context=".MainActivity">
<fragment
    android:id="@+id/fragment1"
    android:name="com.mirea.kt.simpleapp.FirstFragment"
    android:layout_width="0dp"
    android:layout_height="0dp"
    app:layout_constraintBottom_toTopOf="@+id/fragment2"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
<fragment
    android:id="@+id/fragment2"
    android:name="com.mirea.kt.simpleapp.SecondFragment"
    android:layout_width="0dp"
    android:layout_height="0dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/fragment1" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Необходимо указывать
имя класса для каждого
фрагмента, где он
реализован

Использование Layout XML для работы с фрагментами

```
public class FirstFragment extends Fragment {  
    //...  
    @Override  
    public void onAttach(@NonNull Context context) {  
        super.onAttach(context);  
        Log.d( tag: "simple_app_tag", msg: "onAttach FirstFragment");  
    }  
  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup container,  
                             Bundle savedInstanceState) {  
        View rootView = inflater.inflate(R.layout.fragment_first, container, attachToRoot: false);  
        TextView tv = rootView.findViewById(R.id.tvFragmentWelcome);  
        tv.setText("It is a first fragment!");  
        return rootView;  
    }  
}
```

Ссылка на layout-
файл с
содержимым
фрагмента

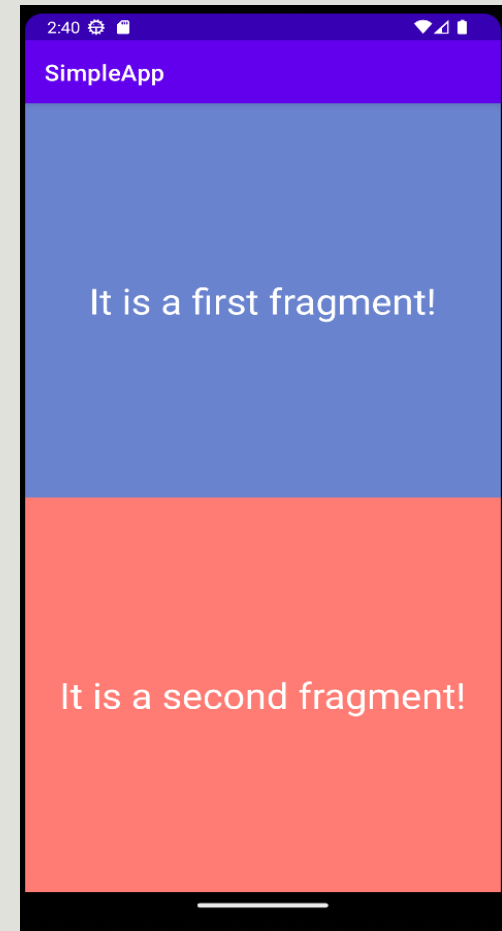


Использование Layout XML для работы с фрагментами

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/my_custom_blue"
    tools:context=".FirstFragment">

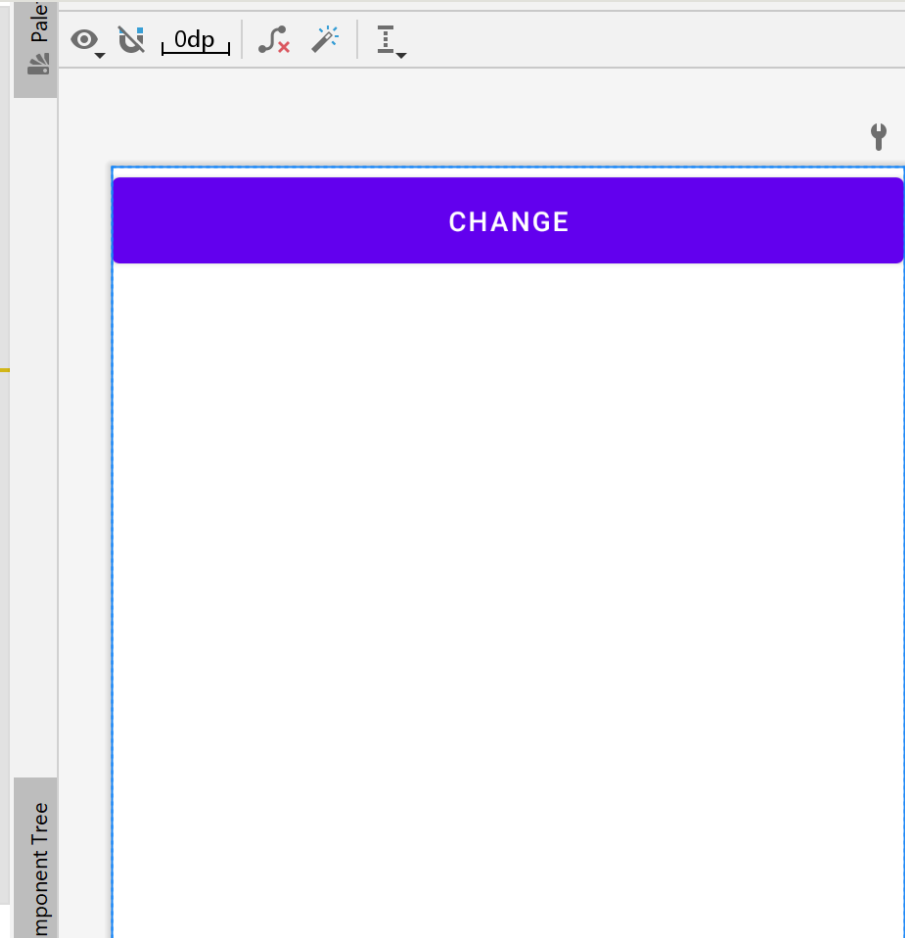
    <TextView
        android:id="@+id/tvFragmentWelcome"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="@color/white"
        android:layout_centerInParent="true"
        android:textSize="32sp"/>

</RelativeLayout>
```



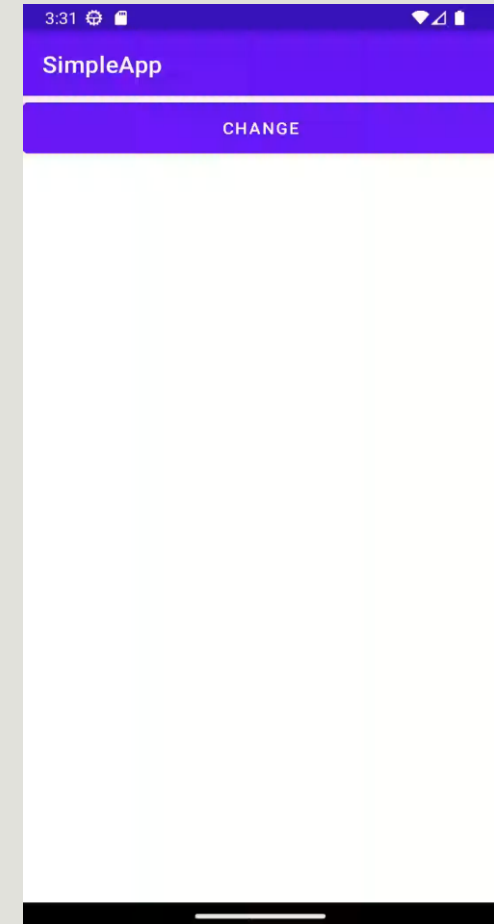
Динамическая работа с фрагментами

```
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <Button
        android:id="@+id/btn"
        android:text="Change"
        android:layout_width="match_parent"
        android:layout_height="56dp"
        app:layout_constraintTop_toTopOf="parent"/>
    <RelativeLayout
        android:id="@+id/relativeContainer"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_marginTop="56dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/btn" />
</androidx.constraintlayout.widget.ConstraintLayout>
```



Динамическая работа с фрагментами

```
public class MainActivity extends AppCompatActivity {  
    private int counterFragments = 1;  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main3);  
        Fragment exampleFrg1 = new FirstFragment();  
        Fragment exampleFrg2 = new SecondFragment();  
        Button btnChange = findViewById(R.id.btn);  
        btnChange.setOnClickListener(v -> {  
            FragmentTransaction ft = getSupportFragmentManager().beginTransaction();  
            if(counterFragments % 2 == 0){  
                ft.replace(R.id.relativeContainer, exampleFrg2);  
            }else{  
                ft.replace(R.id.relativeContainer, exampleFrg1);  
            }  
            ft.commit();  
            counterFragments++;  
        });  
    }  
}
```



FragmentManager

- ❑ позволяет управлять фрагментами и историей фрагментов
- ❑ позволяет добавлять, удалять или менять фрагмент в Activity
- ❑ осуществляет всю работу через **FragmentManager**

FragmentTransaction

- ❑ метод **add** – добавление нового фрагмента
- ❑ метод **replace** – замена текущего фрагмента на новый
- ❑ метод **remove** – удаление фрагмента
- ❑ метод **hide** — делает фрагмент невидимым
- ❑ метод **show** — отображает фрагмент
- ❑ для завершения действия обязателен вызов метода **commit**

Что еще интересного?

Анимация!

Добавим файлы с анимацией в папку res/anim:

```
Activity.java × right_in.xml × righth_out.xml × fragment_first.xml ×  
<?xml version="1.0" encoding="utf-8"?>  
<set xmlns:android="http://schemas.android.com/apk/res/android"  
    android:duration="500"  
    android:interpolator="@android:anim/accelerate_interpolator">  
    <translate android:fromXDelta="-100%" />  
</set>
```

```
Activity.java × right_in.xml × righth_out.xml × fragment_first.xml ×  
<?xml version="1.0" encoding="utf-8"?>  
<set xmlns:android="http://schemas.android.com/apk/res/android"  
    android:duration="500"  
    android:interpolator="@android:anim/accelerate_interpolator">  
    <translate android:toXDelta="100%" />  
</set>
```

Что еще интересного?

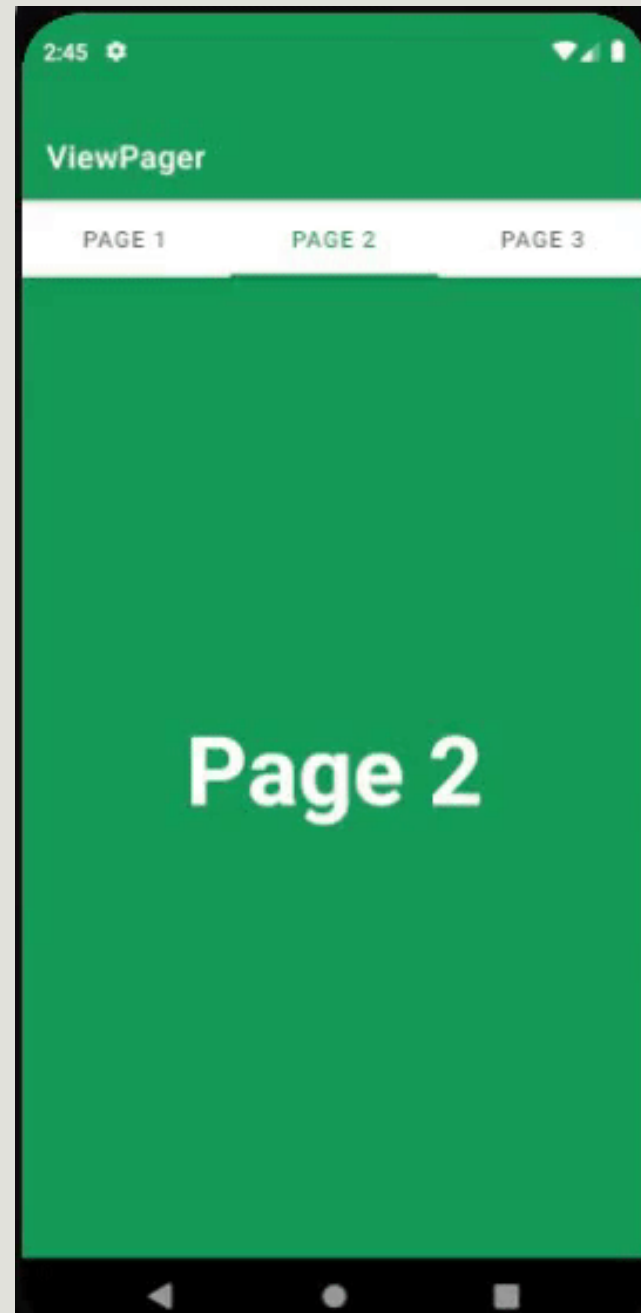
Анимация!

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main3);  
    Fragment exampleFrg1 = new FirstFragment();  
    Fragment exampleFrg2 = new SecondFragment();  
    Button btnChange = findViewById(R.id.btn);  
    btnChange.setOnClickListener(v -> {  
        FragmentTransaction ft = getSupportFragmentManager().beginTransaction();  
        ft.setCustomAnimations(R.anim.right_in, R.anim.righth_out);  
        if(counterFragments % 2 == 0){  
            ft.replace(R.id.relativeContainer, exampleFrg2);  
        }else{  
            ft.replace(R.id.relativeContainer, exampleFrg1);  
        }  
        ft.commit();  
        counterFragments++;  
    });  
}
```

⚠ 13 ⚠ 2



Что еще
интересного?
Компонент
ViewPager



Обмен данными между activity и fragments

- ❑ activity может вызвать методы экземпляра фрагмента
- ❑ activity может создать фрагмент и установить аргументы для него
- ❑ фрагмент может использовать ссылку на «самописный» интерфейс, который будет «слушателем» в activity

Что такое Bundle?

Класс **Bundle** по сути представляет собой оболочку над коллекцией `ArrayMap` для создания более комфортного в работе контейнера для хранения элементов разных типов.

Класс является потокобезопасным и может использоваться для передачи значений между разными потоками. Доступ к элементам осуществляется по парам «ключ-значение».

Методы setArguments и getArguments

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main3);
    Fragment exampleFrg1 = new FirstFragment();
    Fragment exampleFrg2 = new SecondFragment();

    Bundle bnd = new Bundle();
    bnd.putString("str_data", "Какие-то строковые данные");
    bnd.putInt("int_data", 123);
    bnd.putByteArray("bytes_data", new byte[]{1, 2, 3});
    exampleFrg1.setArguments(bnd);

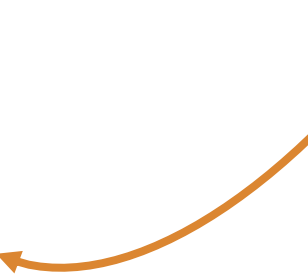
    // продолжение кода
    //...
```

Методы setArguments и getArguments

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                          Bundle savedInstanceState) {
    View rootView = inflater.inflate(R.layout.fragment_first,
                                     container,
                                     attachToRoot: false);

    Bundle bndData = getArguments();
    if(bndData != null){
        String str = bndData.getString(key: "str_data");
        // получение данных других типов
    }

    TextView tv = rootView.findViewById(R.id.tvFragmentWelcome);
    tv.setText("It is a first fragment!");
    return rootView;
}
```



Использование ссылки на интерфейс для связи fragment-activity

1. Создание интерфейса с методом (методами) обратного

```
public interface OnSendFragmentData {  
    void onSendStringData(String data);  
}
```

2. Реализация интерфейса в activity
3. Получение ссылки на интерфейс в fragment в методе onAttach (потому что там передается ссылка на activity)
4. Вызов метода (методов) интерфейса в fragment



ВОПРОСЫ?
