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.

Несколько слов о том, как проще воспринимать фрагмент. Считайте, что фрагмент - это тот же компонент как **Button**, **TextView** или **LinearLayout** с дополнительными возможностями. Фрагмент, как и кнопку, нужно поместить на экран активности. Но фрагмент является модульным компонентом и один и тот же фрагмент можно встроить в две разные активности. С кнопкой такой номер не пройдёт. Для каждой активности вы должны создать свою отдельную кнопку, даже если их нельзя будет отличить друг от друга.

Фрагмент также немного похож на активность. Но фрагменты - это не замена активности, они не существуют сами по себе, а только в составе активностей. Поэтому в манифесте прописывать их не нужно. Но в отличие от стандартной кнопки, для каждого фрагмента вам придётся создавать отдельный класс, как для активности.

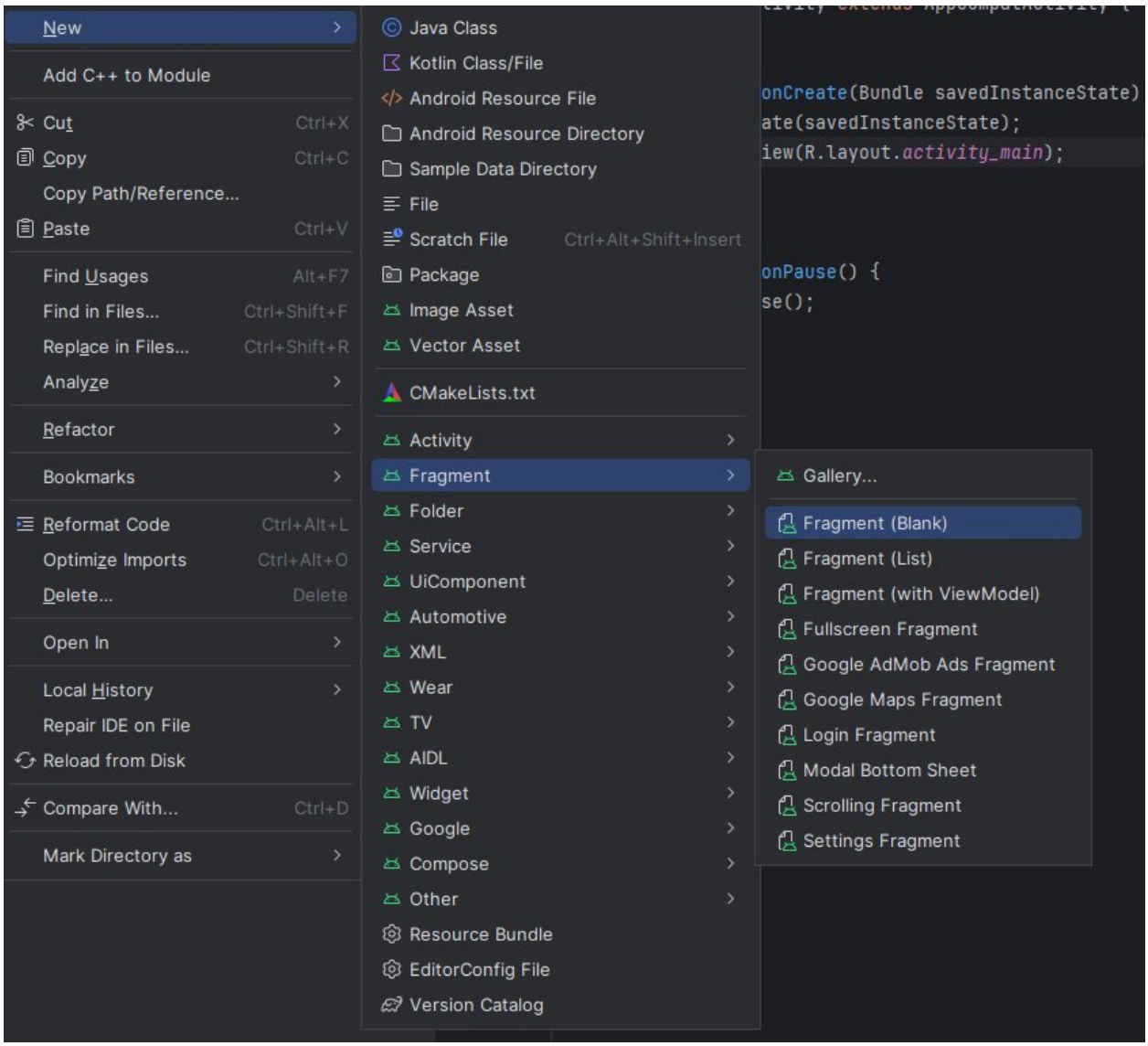
В составе активности есть специальный менеджер фрагментов, который может контролировать все классы фрагментов и управлять ими. О нём позже.

Фрагменты являются строительным материалом для приложения. Вы можете в нужное время добавить новый фрагмент, удалить ненужный фрагмент или заменить один фрагмент на другой. Точно так же мы собираем пазл - подносим фрагмент кота в общую картину, иногда ошибаемся и тогда заменяем кусочек пазла на другой и т.д.

Фрагмент может иметь свою разметку, а может обойтись без неё. Также у фрагмента есть свой жизненный цикл, во многом совпадающий с жизненным циклом активности. Пожалуй, это единственное сходство с активностью.

Давайте создадим ваш первый фрагмент.

Для этого нажмем на папку, где находятся наши классы, правой кнопкой мыши и в появившемся меню выберем **New -> Fragment -> Fragment(Blank)**


New
Add C++ to Module
Cut Ctrl+X
Copy Ctrl+C
Copy Path/Reference...
Paste Ctrl+V
Find Usages Alt+F7
Find in Files... Ctrl+Shift+F
Replace in Files... Ctrl+Shift+R
Analyze
Refactor
Bookmarks
Reformat Code Ctrl+Alt+L
Optimize Imports Ctrl+Alt+O
Delete... Delete
Open In
Local History
Repair IDE on File
Reload from Disk
Compare With... Ctrl+D
Mark Directory as
Java Class
Kotlin Class/File
Android Resource File
Android Resource Directory
Sample Data Directory
File
Scratch File Ctrl+Alt+Shift+Insert
Package
Image Asset
Vector Asset
CMakeLists.txt
Activity
Fragment
Folder
Service
UiComponent
Automotive
XML
Wear
TV
AIDL
Widget
Google
Compose
Other
Resource Bundle
EditorConfig File
Version Catalog
Gallery...
Fragment (Blank)
Fragment (List)
Fragment (with ViewModel)
Fullscreen Fragment
Google AdMob Ads Fragment
Google Maps Fragment
Login Fragment
Modal Bottom Sheet
Scrolling Fragment
Settings Fragment
onCreate(Bundle savedInstanceState)
ate(savedInstanceState);
iew(R.layout.activity_main);
onPause() {
se();

Перейдите на класс вашего новосозданного фрагмента и очистите его, чтобы остался только этот код:

```
package com.walerider.myapplication;

import ...

public class NewFragment extends Fragment {



    public NewFragment() {
    }

    public static NewFragment newInstance(String param1, String param2) {  no usages
        NewFragment fragment = new NewFragment();
        Bundle args = new Bundle();
        fragment.setArguments(args);
        return fragment;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (getArguments() != null) {
        }
    }

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

Теперь, перейдите на разметку вашего фрагмента. Обычно, она называется fragment_НазваниеФрагмента.

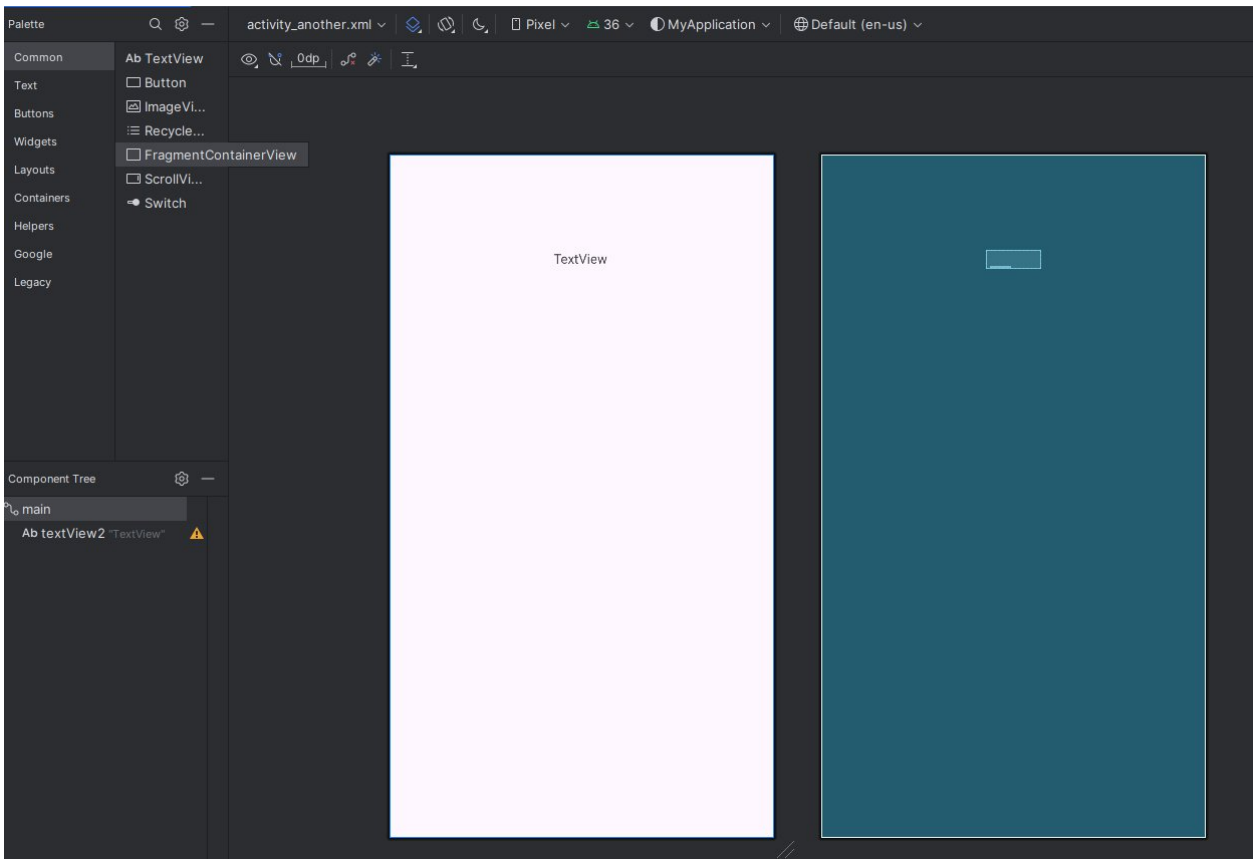
Замените FrameLayout на более удобный для нас ConstraintLayout.

```
<FrameLayout 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"
    tools:context=".NewFragment">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Hello blank fragment" />

</FrameLayout>
```

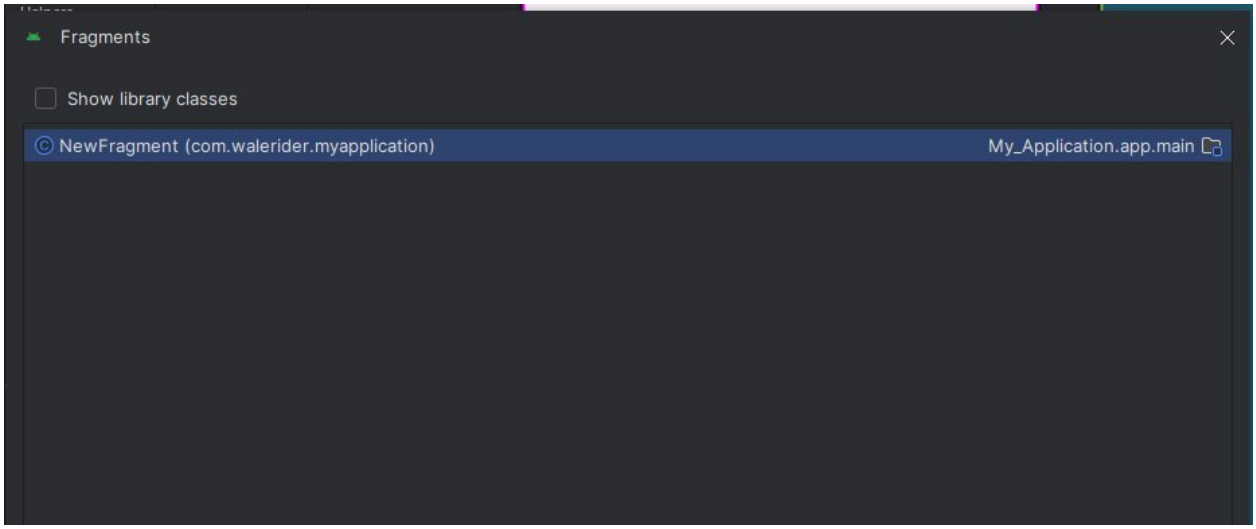
```
<Cons xmlns:android="http://schemas.android.com/apk/res/android"
androidx.constraintlayout.widget.ConstraintLayout
```

```
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/a
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".NewFragment">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Hello blank fragment" />

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

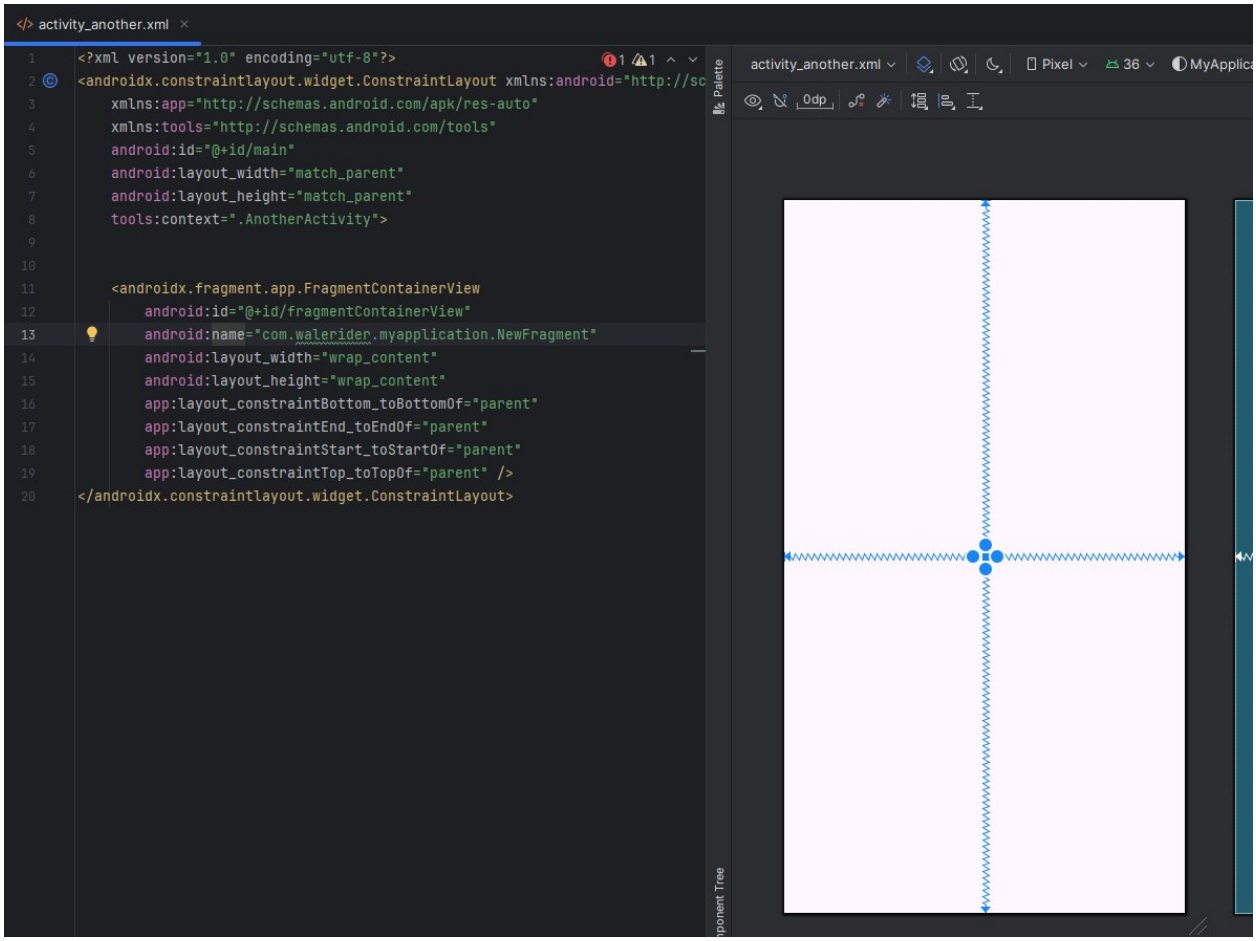
Для использования фрагмента добавим его в **MainActivity**. Для этого изменим файл **activity_main.xml**, которая определяет интерфейс для MainActivity:



Выберем подходящий нам фрагмент:



Обозначим связи и уберём все остальные элементы, чтобы осталось так:



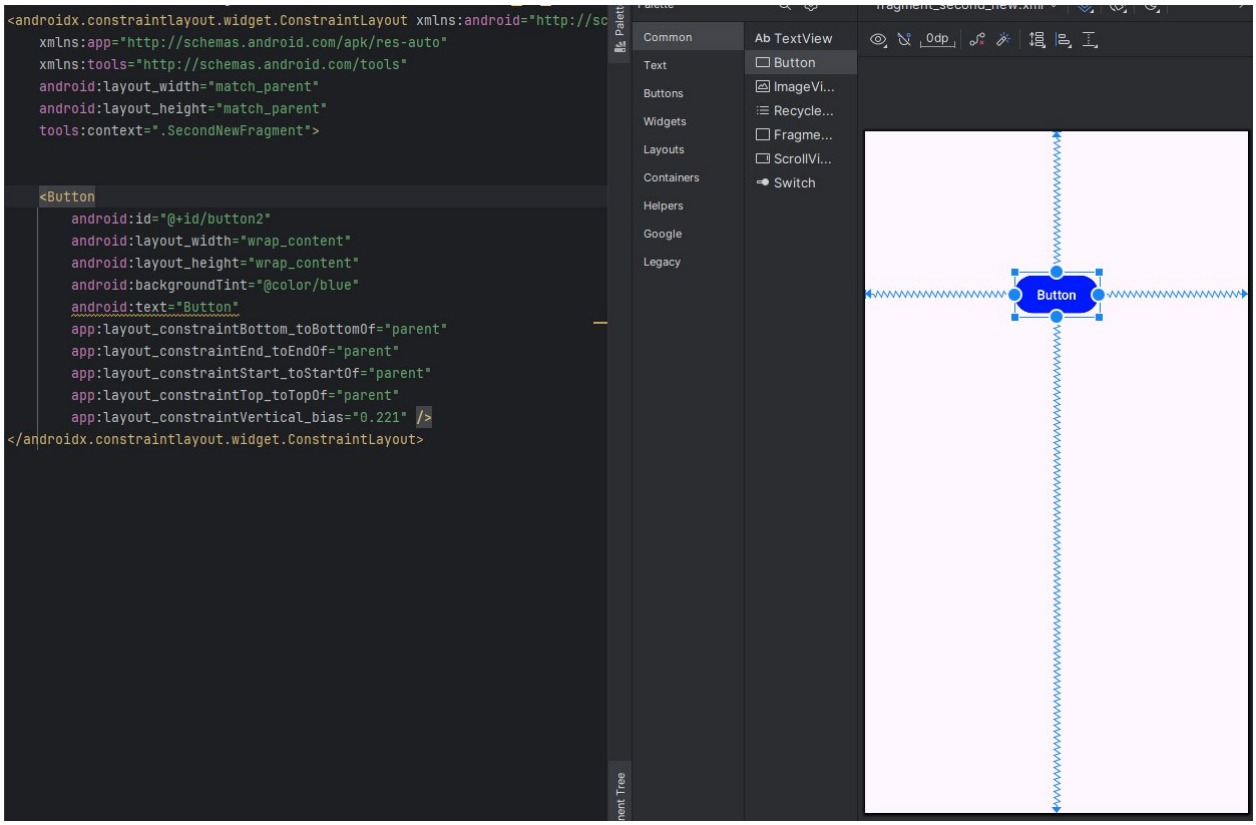
Для добавления фрамента применяется элемент **FragmentContainerView**. По сути FragmentContainerView представляет объект View, который расширяет класс **FrameLayout** и предназначен специально для работы с фрагментами. Собственно кроме фрагментов он больше ничего содержать не может.

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

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

Давайте создадим ещё один fragment. Определите какой-нибудь дизайн для

своего нового фрагмента. У меня для примера такой:




```
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://sc
    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=".SecondNewFragment">

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:backgroundTint="@color/blue"
        android:text="Button"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.221" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Ну и логику конечно.

```
activity_another.xml   fragment_second_new.xml   SecondNewFragment.java

package com.walerider.myapplication;

import ...

public class SecondNewFragment extends Fragment {

    Button button;  2 usages
    public SecondNewFragment() {
    }

    public static SecondNewFragment newInstance(String param1, String param2) {  no usages
        SecondNewFragment fragment = new SecondNewFragment();
        return fragment;
    }

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

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_second_new, container,  attachToRoot: false);
        button = view.findViewById(R.id.button2);
        button.setOnClickListener(v ->{
            Toast.makeText(getContext(), text: "text",Toast.LENGTH_LONG);
        });
        return view;
    }
}
```

Объяснение:

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.

```
inflater.inflate(R.layout.fragment_second_new, 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()`.

Все получения значений, для которых в Activity мы использовали this мы заменяем методами, возвращающие это значение или обращаемся к Activity, к которому привязан fragment с помощью getActivity();

И также изменим класс **MainActivity**

```
import ...

public class AnotherActivity extends AppCompatActivity {


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable( $this$enableEdgeToEdge: this);
        setContentView(R.layout.activity_another);
        if (savedInstanceState == null) {
            getSupportFragmentManager().beginTransaction()
                    .add(R.id.fragmentContainerView, NewFragment.class,  args: null)
                    .commit();
        }
    }

    @Override
    protected void onResume() {
        super.onResume();
    }
}
```

Метод `getSupportFragmentManager()` возвращает объект `FragmentManager`, который управляет фрагментами.

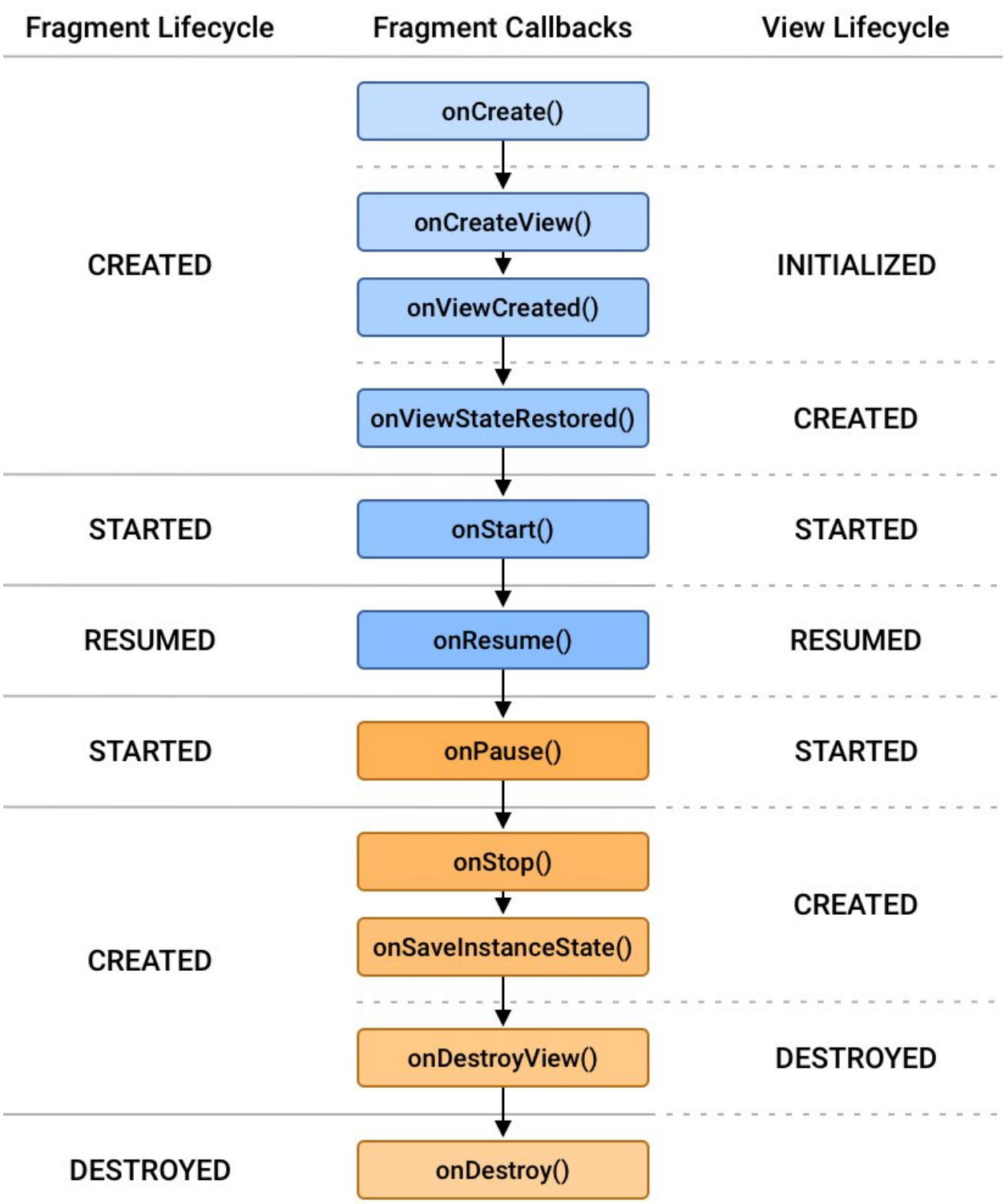
Объект FragmentManager с помощью метода `beginTransaction()` создает объект **FragmentTransaction.**

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:



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 **FragmentManager**, который далее определяет состояние фрагмента и переводит фрагмент из одного состояния в другое.
- **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)
```

- **onCreateView()**: 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**

Дополнительно для фрагмента определено два метода обратного вызова, которые связаны с прикреплением фрагмента к activity:

- 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. Начиная с этого момента фрагмент становится активным, и FragmentManager начинает управлять его жизненным циклом.
- 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. К примеру у меня это будет в SecondNewFragment.

```
package com.walerider.myapplication;

import android.content.Context;
import android.os.Bundle;

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

import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.*;
import android.widget.Toast;

import java.util.Date;

public class SecondNewFragment extends Fragment {

    private final static String TAG = "SecondNewFragment";
```

```java
public SecondNewFragment(){
    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_second_new, container, false);
}
@Override
public void onViewCreated(@NonNull View view, Bundle savedInstanceState) {
    super.onViewCreated(view, savedInstanceState);
    Button updateButton = view.findViewById(R.id.button2);
    TextView updateBox = view.findViewById(R.id.textView2);

    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");
    }
}
```

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.

Измените 2 своих новосозданных фрагмента так, чтобы : один содержал список, а второй - текстовой поле. Логика будет такова: при выборе элемента в списке в одном фрагменте выбранный элемент должен отобразиться в текстовом поле, которое находится во втором фрагменте.

Измените свой дизайн активности для вывода фрагмента:

```
<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:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".AnotherActivity">
    <androidx.fragment.app.FragmentContainerView
        android:id="@+id/fragmentContainerView"
        android:name="com.walerider.myapplication.NewFragment"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHeight_percent="0.5"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.0" />
    <androidx.fragment.app.FragmentContainerView
        android:id="@+id/fragmentContainerView2"
        android:name="com.walerider.myapplication.SecondNewFragment"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHeight_percent="0.5"
        app:layout_constraintHorizontal_bias="1.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
```

```
        app:layout_constraintVertical_bias="1.0" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

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. К примеру:

```
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 определите метод с таким же количеством аргуметов, как и в вложенном интерфейсе. К примеру:

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

Теперь перейдём в нашу Activity, где имплементируем наш вложенный в интерфейс:

```
public class AnotherActivity extends AppCompatActivity implements
NewFragment.OnFragmentSendDataListener{
```

И переопределим метод нашего интерфейса:

```
@Override
public void onSendData(String data) {
    SecondNewFragment fragment = (SecondNewFragment)
getSupportFragmentManager()
            .findFragmentById(R.id.fragmentContainerView2);
```

```
    if (fragment != null)
        fragment.setSelectedItem(data);
}
```

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

В итоге получится, что при выборе в списке во фрагменте NewFragment 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 SecondNewFragment. Таким образом, произойдет взаимодействие между двумя фрагментами.

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