



# План занятия

1. [Fragment](#)
2. [FragmentManager](#)
3. [Жизненный цикл фрагментов](#)
4. [Итоги](#)

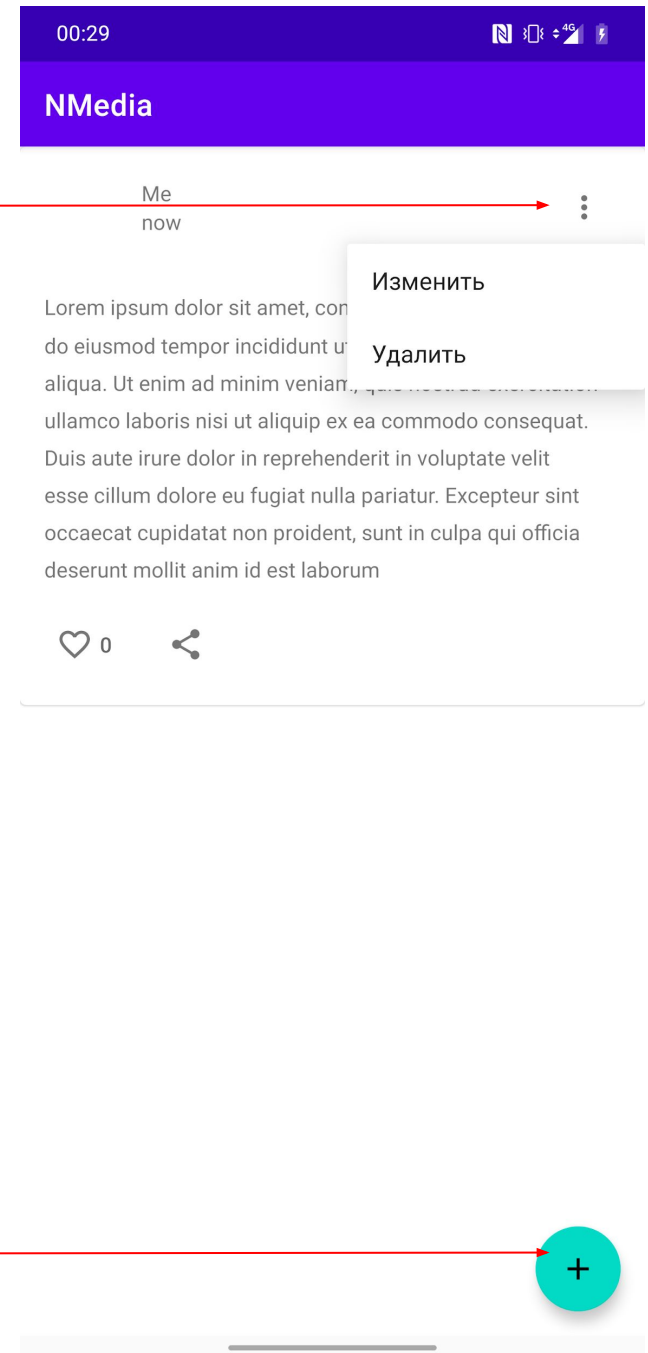


# Fragment

# Что мы имеем?

У нас есть приложение для просмотра ленты новостей:

Меню

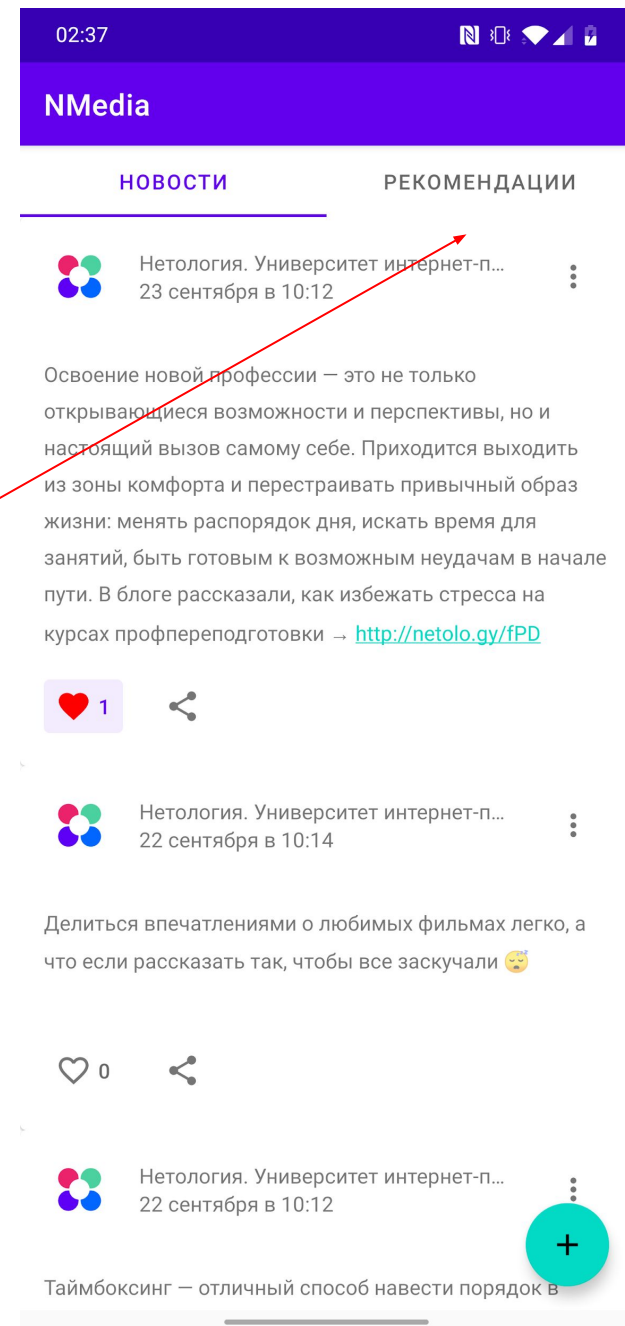


Переход к созданию постов

# Готовность к изменениям

Q: Можем ли мы создать вторую вкладку, используя вторую активити?

Новый раздел с постами



# Fragments

На данный момент общепринятый подход – разделение приложения на небольшие части, [фрагменты](#).

Существует также [подход](#) к построению декларативного UI, где вся отрисовка выносится в kotlin функции. Но данная библиотека появилась достаточно недавно и еще мало где используется. Мы рассмотрим наиболее распространенный подход - будем использовать фрагменты.

# android.app.Fragment

Впервые класс Fragment появился в Android 3.0 и поставлялся вместе с системой.

Но такой подход имеет ряд недостатков:

1. Производители могут модифицировать поведение.
2. На момент выхода фрагментов необходимо было поддерживать более старые версии ОС.
3. Для обновления библиотеки необходимо обновить прошивку девайса

# androidx.app.Fragment

В марте 2011 года появилась [отличная новость](#). Фрагменты теперь поставляются отдельно от системы в виде сторонней библиотеки и могут работать с версии 1.6. Старый android.app.Fragment объявлен устаревшим и не рекомендуется к использованию.

Скорее всего у вас уже есть доступ к фрагментам через зависимость appcompat

```
implementation 'androidx.appcompat:appcompat:1.2.0'
```

Если вам нужны все самые последние новинки, рекомендуем подключить фрагменты отдельно

```
implementation "androidx.fragment:fragment-ktx:1.3.1"
```

## Связь с Activity

По умолчанию фрагменты нельзя использовать отдельно от Activity.

Кроме того, вам нужно наследовать ваши Activity от `androidx.fragment.app.FragmentActivity`.

Ваши Activity уже наследуются от этого класса, потому что `AppCompatActivity` является наследником этого класса.

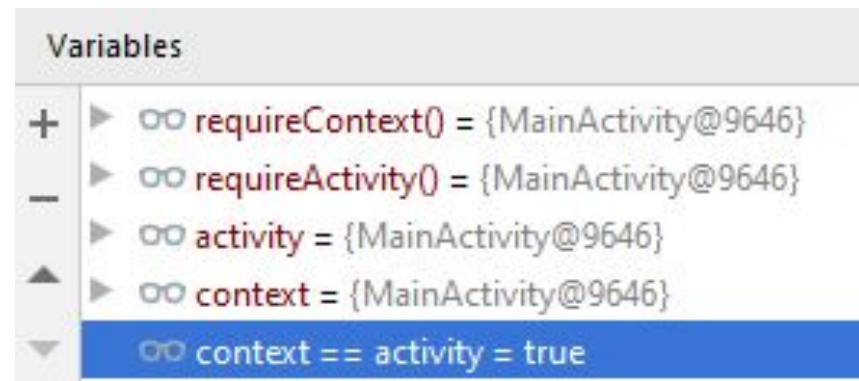
```
* </div>
*/
public class AppCompatActivity extends FragmentActivity implements AppCompatActivity,
    TaskStackBuilder.SupportParentable, ActionBarDrawerToggle.DelegateProvider {
```



## Доступ к Activity

Если из фрагмента нужно получить доступ к Activity, к которой он прикреплен, возможны следующие варианты:

1. Nullable проперти activity.
2. Nullable проперти context.
3. Функция `requireActivity()`.
4. Функция `requireContext()`.



Если вы используете фрагменты в Activity, то все эти 4 метода вернут один и тот же объект. Рекомендуется использовать `Context` вместо `Activity`, если вам не нужны какие-либо специфичные методы `Activity`.



# FragmentManager

# FragmentManager

Недостаточно просто создать экземпляр класса `Fragment`, чтобы он появился на экране.

Во-первых, необходимо создать место на экране, где фрагмент будет расположен. Подойдёт любой наследник `ViewGroup`. Например, `FrameLayout`:

```
<FrameLayout
    android:id="@+id/fragmentContainer"
    android:layout_marginTop="?actionBarSize"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```



---

# FragmentManager

Во-вторых, нужно отправить запрос на изменение состояния содержимого нашего контейнера. Для этого воспользуемся классом `androidx.fragment.app.FragmentManager`.

Он реализует методы CRUD для работы с фрагментами:

1. `add`
2. `replace`
3. `remove`
4. `findFragmentById`
5. `findFragmentByTag`

Однако выполнить первые 3 метода просто так не получится. Для этого нужно создать транзакцию.

## Транзакции FragmentManager

Для того чтобы произвести транзакцию, вызовите функцию `commit` или `commitNow`.

Примеры использования:

```
supportFragmentManager.commit { this: FragmentTransaction  
    add(R.id.fragmentContainer, WallFragment())  
}
```

Если фрагмент уже отображается, но нужно показать другой, используйте метод `replace`:

```
supportFragmentManager.commit { this: FragmentTransaction  
    replace(R.id.fragmentContainer, WallFragment())  
}
```

## Back Stack

Если необходимо, чтобы по кнопке назад возвращался предыдущий фрагмент, можно воспользоваться методом `addToBackStack()`:

```
val rootFragmentName = "root"
supportFragmentManager.commit { this: FragmentTransaction
    add(R.id.fragmentContainer, fragmentToRemove)
    addToBackStack(rootFragmentName)
}
```

При этом `rootFragmentName` может быть `null`.

Для того чтобы программно вернуться к этому фрагменту, используйте метод `popBackStack()`:

```
supportFragmentManager.popBackStack(
    rootFragmentName,
    FragmentManager.POP_BACK_STACK_INCLUSIVE
)
```



## CommitNow vs Commit

**Q:** В чём принципиальное отличие `commit` от `commitNow`?

**A:** Когда мы проводим транзакцию методом `commit`, она отправляется в очередь и применяется только во время отрисовки следующего кадра.

При использовании `commitNow` транзакция применяется сразу, но в таком случае нельзя использовать `addToBackStack`, потому что в очереди уже могут быть транзакции с вызовом этого метода.

В общем случае рекомендуется использовать `commit`.

# Android Navigation Component

Мы уже умеем работать с навигацией при помощи Activity. Можно с уверенностью сказать, что работать с несколькими активити проще, чем с фрагментами. По крайней мере, не нужно заботиться о транзакциях и back stack.

Чтобы упростить дальнейшую работу с навигацией фрагментов, мы будем использовать [библиотеку навигации от Google](#).



# Android Navigation Component

Добавим необходимые зависимости в app/build.gradle

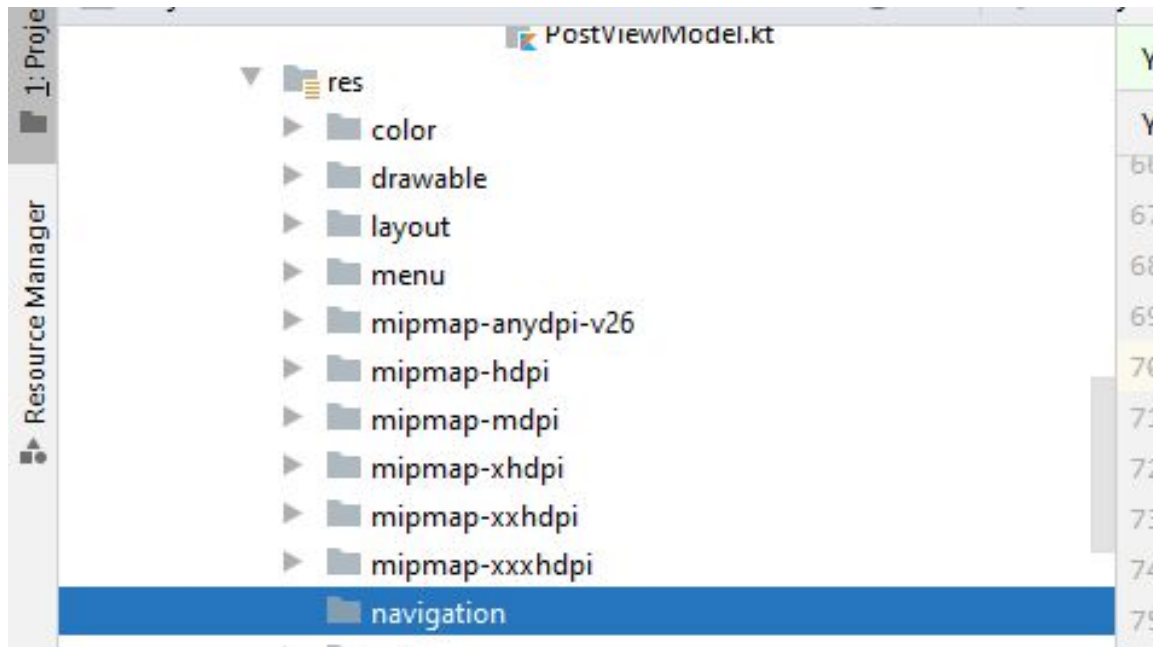
```
def nav_version = "2.3.4"

implementation
"androidx.navigation:navigation-fragment-ktx:$nav_version"

implementation
"androidx.navigation:navigation-ui-ktx:$nav_version"
```

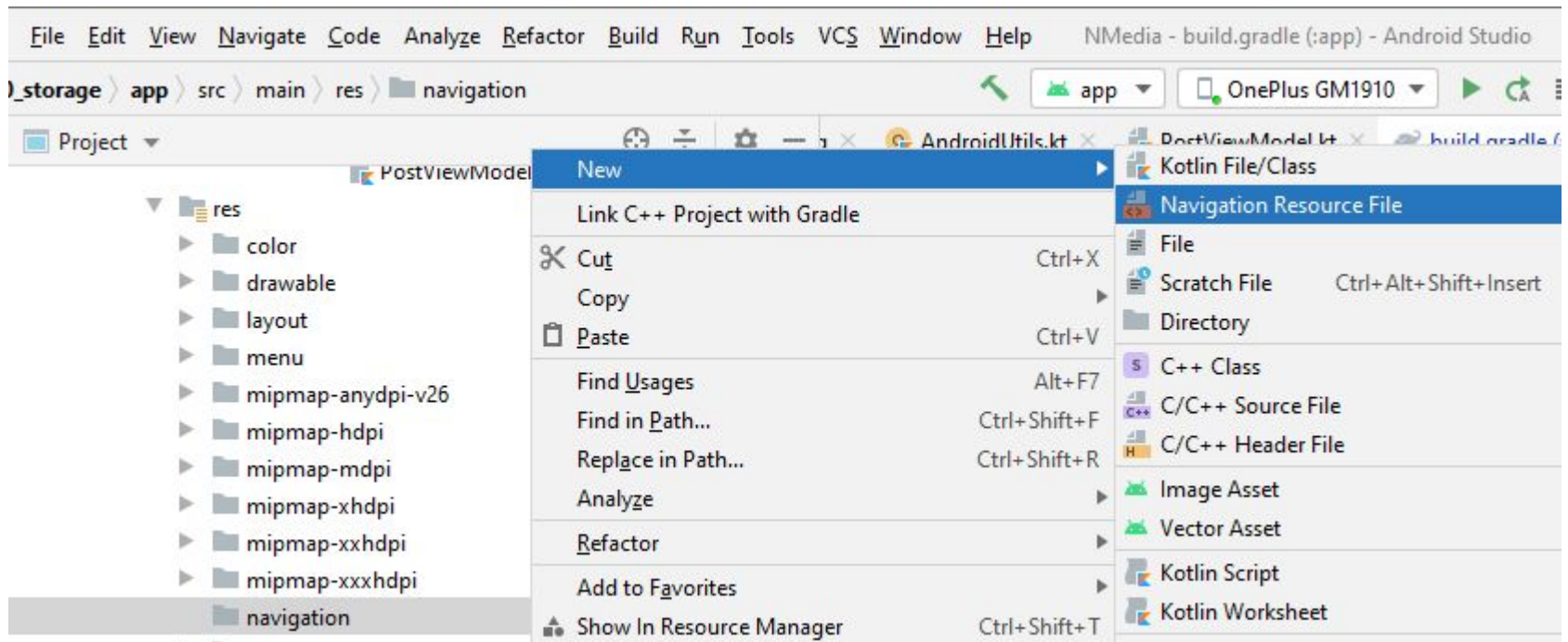
# Android Navigation Component

Графы навигации будем хранить в директории res/navigation в формате xml.



# Граф навигации

Создадим наш первый граф. Нажимаем правой кнопкой мыши на созданную navigation. Затем: New -> Navigation Resource File

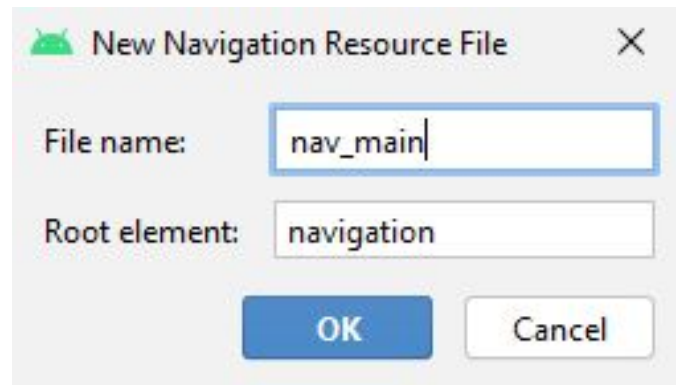


## Граф навигации

В появившемся окне необходимо назвать граф/файл.

Назовём его `nav_main`.

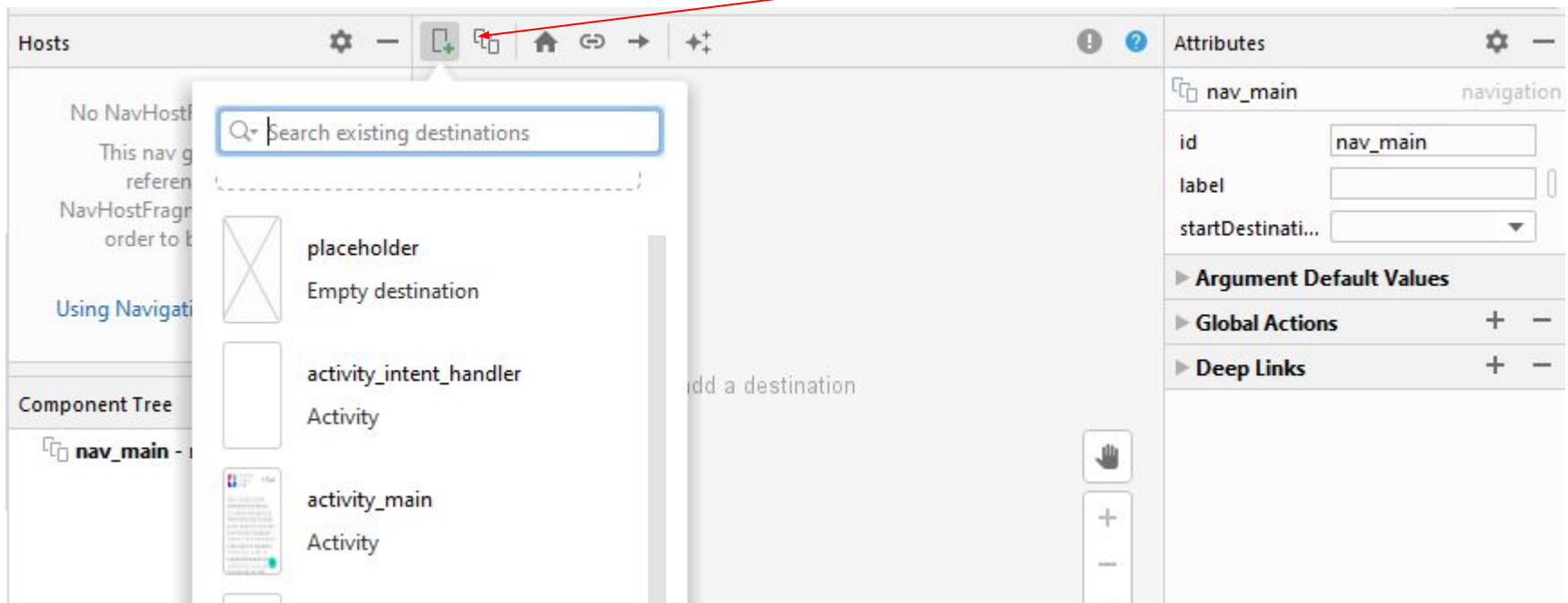
Root element оставляем без изменений.



# Граф навигации

На данном этапе мы можем создать граф, используя Activity. Но мы договорились о переходе на фрагменты.

Добавлять пункты назначения

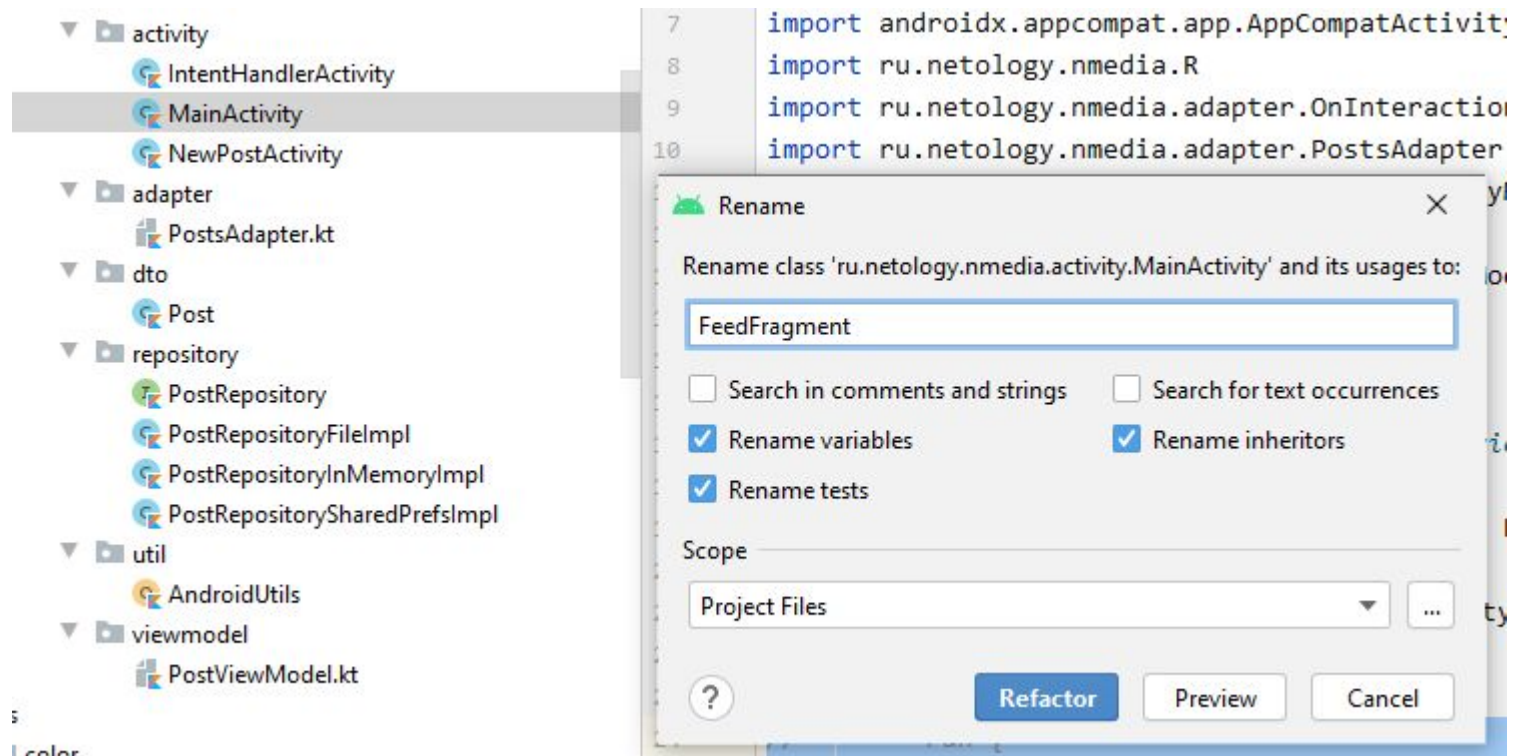


# Рефакторинг Activity -> Fragment

В первую очередь переименовываем все активити во фрагменты.

Рассмотрим миграцию на примере MainActivity.

Выделяем MainActivity, нажимаем Shift + F6 и меняем имя.



# Рефакторинг Activity -> Fragment

Наследуемся уже не от AppCompatActivity, а от  
androidx.fragment.app.Fragment:

```
class FeedFragment : Fragment() {  
    private val newPostRequestCode = 1  
    private val viewModel: PostViewModel by viewModels()
```

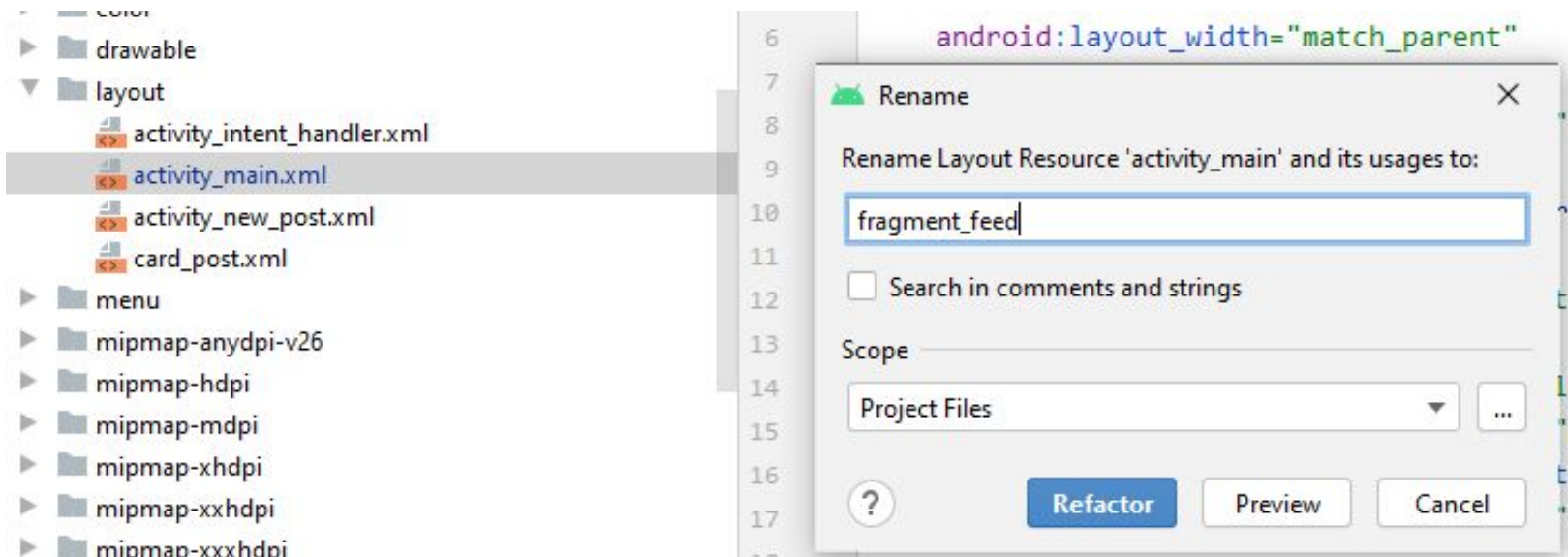
Меняем импорт для viewModels:

```
class FeedFragment : Fragment() {  
    private val newPostRequestCode = 1  
    private val viewModel: PostViewModel by viewModels()
```

Инициализация визуальных элементов у нас будет происходить в  
функции onCreateView вместо onCreate. Позже разберем  
жизненный цикл фрагментов.

# Рефакторинг Activity -> Fragment

Но для начала переименуем activity\_main.xml в fragment\_feed.xml





# Рефакторинг Activity -> Fragment

Теперь мы можем реализовать onCreateView.

Переносим весь код из onCreate в onCreateView.

```
override fun onCreateView(  
    inflater: LayoutInflater,  
    container: ViewGroup?,  
    savedInstanceState: Bundle?  
): View? {  
    val binding : FragmentFeedBinding = FragmentFeedBinding.inflate(  
        inflater,  
        container,  
        attachToParent: false  
    )  
    // Содержимое onCreate, оставшееся от Activity должно быть здесь  
    return binding.root  
}
```

## Рефакторинг Activity -> Fragment

При этом `this` в функциях `observe` нужно заменить на `viewLifecycleOwner`. Иначе может произойти утечка памяти.

Переход к созданию поста реализуем, когда появится соответствующий `destination`

```
binding.list.adapter = adapter
viewModel.data.observe( owner: this, { posts ->
    adapter.submitList(posts)
})
```

```
binding.fab.setOnClickListener {
    TODO()
}
```

Use viewLifecycleOwner as the LifecycleOwner.

Replace with viewLifecycleOwner Alt+Shift+Enter

```
ru.netology.nmedia.activity FeedFragment.kt
public final class FeedFragment : Fragment
```

NMedia.app

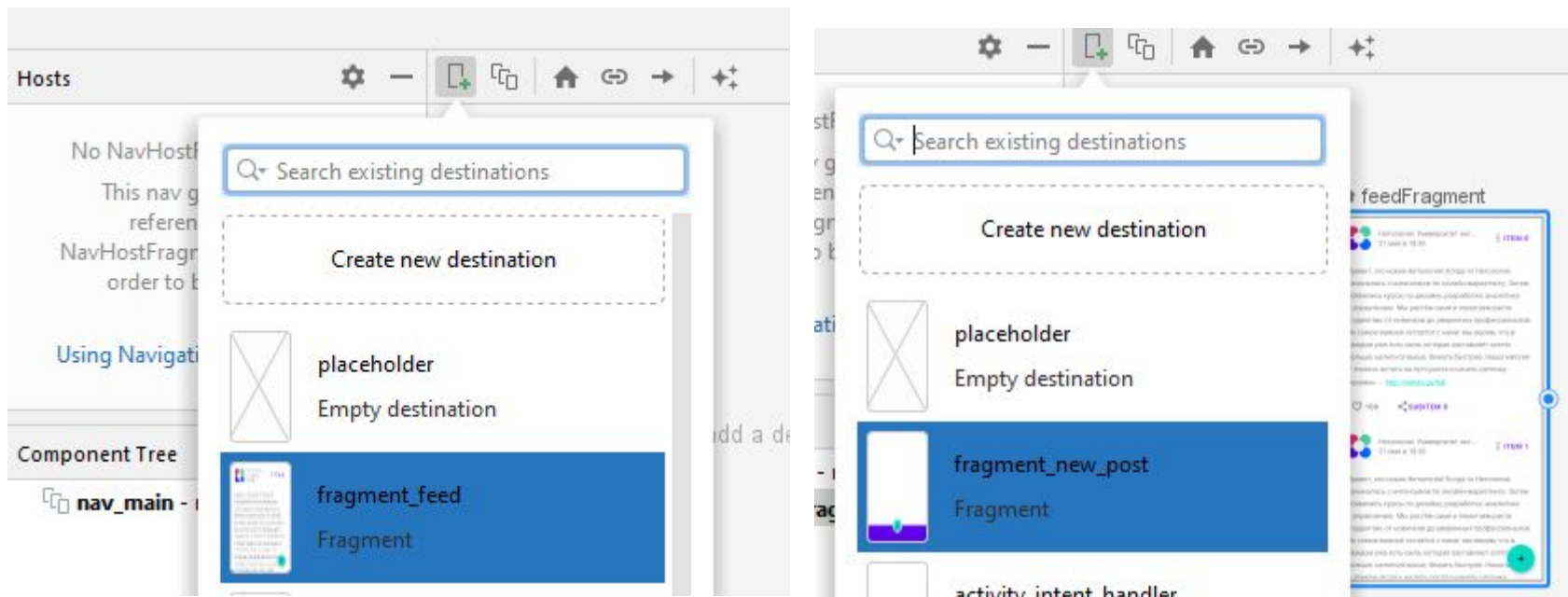
# Рефакторинг Activity -> Fragment

По аналогии реализуем NewPostFragment.

```
NewPostFragment.kt x fragment_new_post.xml x IntentHandlerActivity.kt x activity_intent_handler.xml
19         savedInstanceState: Bundle?
20     ): View? {
21         val binding : FragmentNewPostBinding = FragmentNewPostBinding.inflate(
22             inflater,
23             container,
24             attachToParent: false
25         )
26         binding.edit.requestFocus()
27         binding.ok.setOnClickListener { it: View!
28             val intent = Intent()
29             if (TextUtils.isEmpty(binding.edit.text)) {
30                 activity?.setResult(Activity.RESULT_CANCELED, intent)
31             } else {
32                 val content : String = binding.edit.text.toString()
33                 intent.putExtra(Intent.EXTRA_TEXT, content)
34                 activity?.setResult(Activity.RESULT_OK, intent)
35             }
36             findNavController().navigateUp()
37         }
38         return binding.root
39     }
```

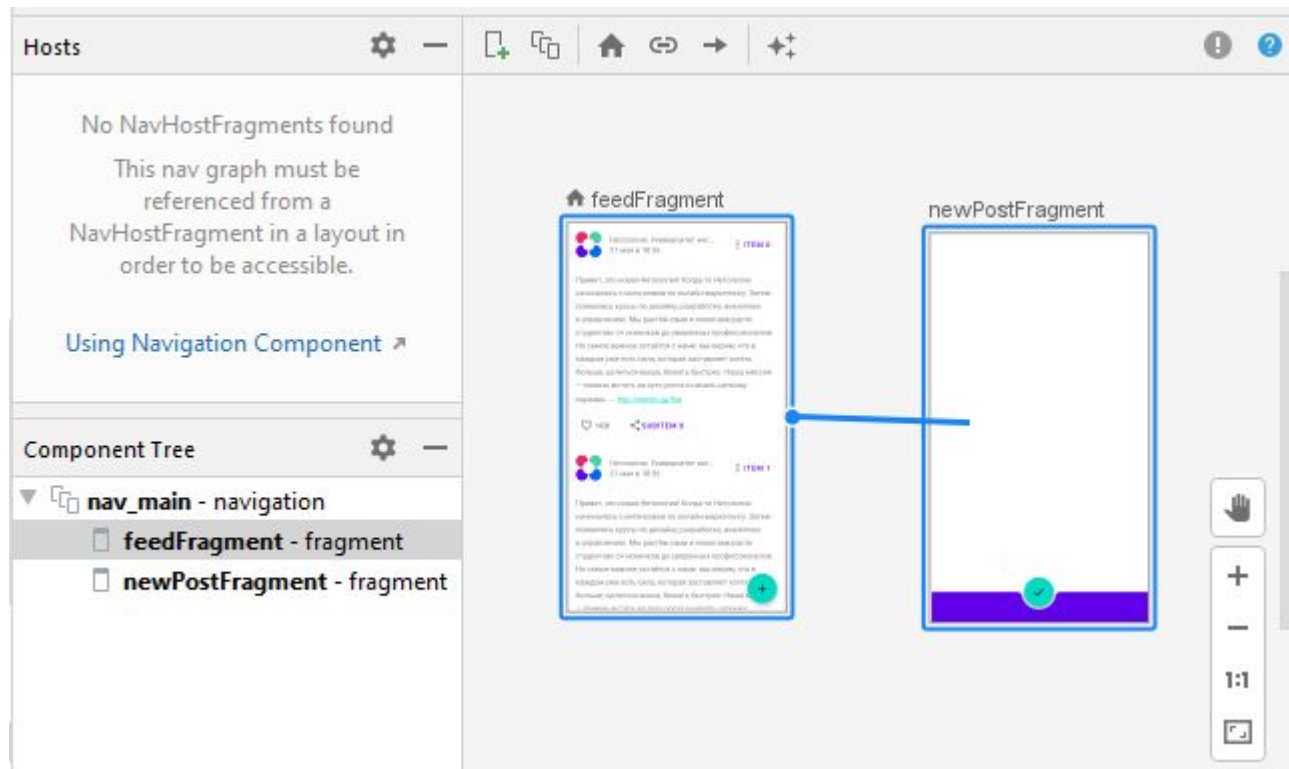
# Рефакторинг Activity -> Fragment

Теперь мы можем реализовать переход между фрагментами в графе навигации. Добавим сначала `fragment_feed`, а затем `fragment_new_post`.



# Рефакторинг Activity -> Fragment

От правого края feedFragment протянем стрелку к newPostFragment:



# Рефакторинг Activity -> Fragment

В результате будет создан переход (action) от feedFragment к newPostFragment. Скопируем сгенерированный id.

The screenshot shows the Android Studio IDE with the following components:

- Hosts Panel:** Displays a message: "No NavHostFragments found. This nav graph must be referenced from a NavHostFragment in a layout in order to be accessible. Using Navigation Component".
- Component Tree:** Shows the navigation structure:
  - nav\_main - navigation
    - feedFragment - fragment
      - action\_feedFragment\_to\_new... (selected)
      - newPostFragment - fragment
- Design View:** Shows a visual representation of the navigation. A blue arrow points from the feedFragment (containing a list of items) to the newPostFragment (a blank screen with a bottom navigation bar).
- Attributes Panel:** Shows the configuration for the selected action:
  - id: newPostFragment (highlighted)
  - destination: (dropdown menu)
  - Animations: enterAnim, exitAnim, popEnterAnim, popExitAnim (input fields)
  - Argument Default Values: (input fields)
  - Pop Behavior: popUpTo (dropdown), popUpToIncl... (checkbox)

## Рефакторинг Activity -> Fragment

Вернёмся к FeedFragment и реализуем переход между экранами

```
binding.fab.setOnClickListener { it: View!  
    findNavController().navigate(R.id.action_feedFragment_to_newPostFragment)  
}
```

На данном этапе нам осталось реализовать 2 вещи:

1. Открытие фрагмента редактирования по Intent.ACTION\_SEND.
2. Установить граф в активити.



## Передача аргументов между фрагментами

Для передачи данных между фрагментами нам понадобится уже известный класс Bundle. Он поддерживает хранение по ключу примитивов, а также объектов типа Parcelable и Serializable.

Для передачи текста воспользуемся следующим экстеншеном:

```
companion object {  
    private const val TEXT_KEY = "TEXT_KEY"  
    var Bundle.textArg: String?  
        set(value) = putString(TEXT_KEY, value)  
        get() = getString(TEXT_KEY)  
}
```



## Передача аргументов между фрагментами

Для передачи аргументов запишем в bundle текст и передадим его в функцию navigate.

```
val text : String? = it.getStringExtra(Intent.EXTRA_TEXT)
if (text?.isNotBlank() == true) {
    intent.removeExtra(Intent.EXTRA_TEXT)
    findNavController(R.id.nav_host_fragment)
        .navigate(
            R.id.action_feedFragment_to_newPostFragment,
            Bundle().apply { this: Bundle
                textArg = text
            }
        )
}
```

Внутри фрагмента обратимся к property arguments и прочитаем текст.

```
arguments?.textArg
    ?.let(binding.edit::setText)
```

## Передача аргументов между фрагментами

Помним про принцип Don't Repeat Yourself (DRY). Работу с аргументами можно делегировать другому объекту, чтобы не повторяться в каждом фрагменте.

```
object StringArg: ReadWriteProperty<Bundle, String?> {  
    override fun setValue(thisRef: Bundle, property: KProperty<*>, value: String?) {  
        thisRef.putString(property.name, value)  
    }  
  
    override fun getValue(thisRef: Bundle, property: KProperty<*>): String? =  
        thisRef.getString(property.name)  
}
```

Пример использования:

```
companion object {  
    var Bundle.textArg: String? by StringArg  
}
```

## Shared ViewModel

Так как мы работаем в рамках одной активности, мы можем предоставить одну ViewModel нескольким фрагментам для передачи данных.

```
private val viewModel: PostViewModel by viewModels(  
    ownerProducer = ::requireParentFragment  
)
```

Родительским фрагментом для FeedFragment и NewPostFragment будет NavHostFragment, значит в рамках его ЖЦ можно хранить PostViewModel (ownerProducer).

## Shared ViewModel

Убираем `ActivityResultContract` и переносим сохранение поста напрямую в `NewPostFragment`.

```
binding.ok.setOnClickListener { it: View!
    viewModel.changeContent(binding.edit.text.toString())
    viewModel.save()
    AndroidUtils.hideKeyboard(requireView())
    findNavController().navigateUp()
}
```

Еще одной альтернативой является [Fragment Result Api](#). Однако, данная библиотека ещё в alpha версии и не рекомендуется к использованию в production.

# Установка графа навигации

Переименуем `IntentHandlerActivity` в `AppActivity` и удалим лишнее.

```
1 package ru.netology.nmedia.activity
2
3 import androidx.appcompat.app.AppCompatActivity
4 import ru.netology.nmedia.R
5
6 class AppActivity : AppCompatActivity(R.layout.activity_app)
```

Соответственно, `activity_intent_handler.xml` в `activity_app.xml`, чтобы было всем очевидно — в нашем проекте только одна Activity.

```
<?xml version="1.0" encoding="utf-8"?>
<fragment 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/nav_host_fragment"
    android:name="androidx.navigation.fragment.NavHostFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:defaultNavHost="true"
    app:navGraph="@navigation/nav_main"
    tools:context=".activity.AppActivity" />
```

# Android Manifest


Не забудьте актуализировать AndroidManifest.xml. Фрагменты не являются одним из четырех компонентов Android, которые нужно указывать в данном файле.

```
<activity android:name=".activity.AppActivity">
  <nav-graph android:value="@navigation/nav_main" />
  <intent-filter>
    <action android:name="android.intent.action.SEND" />

    <category android:name="android.intent.category.DEFAULT" />

    <data android:mimeType="text/plain" />
  </intent-filter>
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />

    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
```



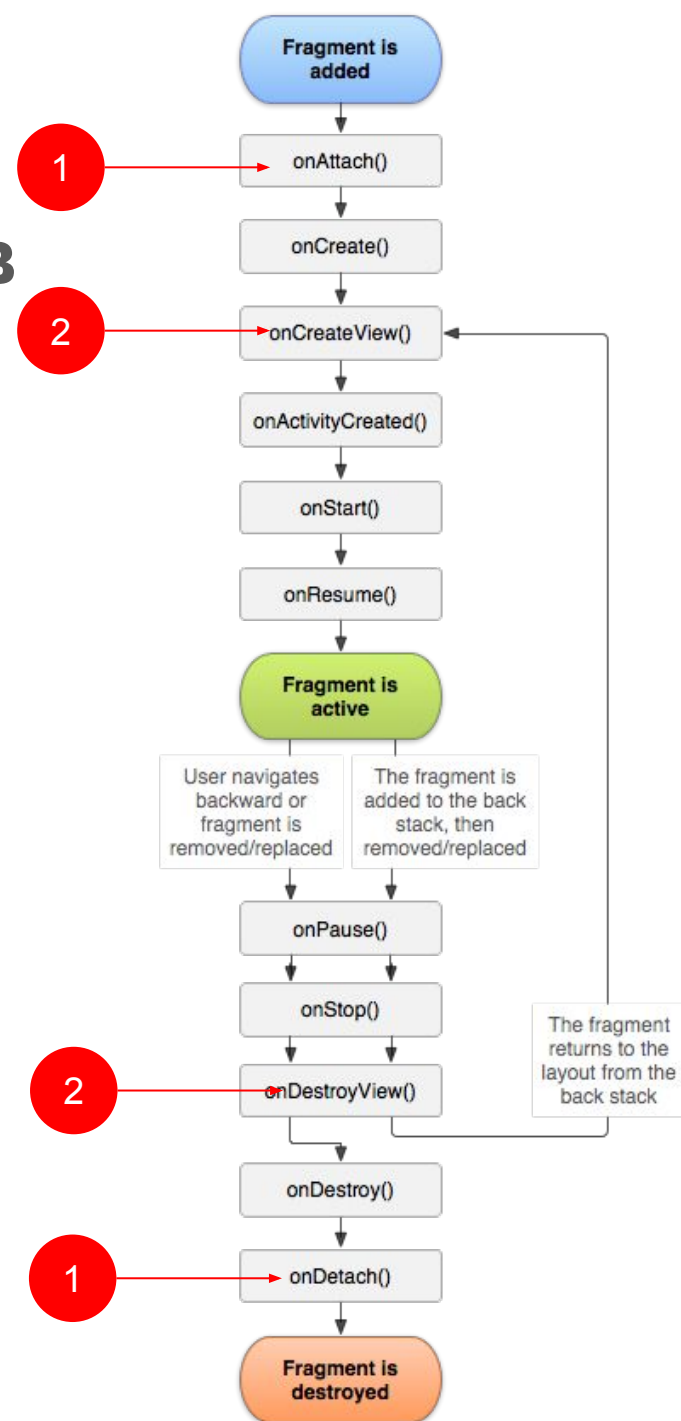
# **Жизненный цикл фрагментов**

# Жизненный цикл фрагментов

Жизненный цикл фрагментов во многом дублирует жизненный цикл активности.

Два главных отличия:

1. Фрагмент может находиться отдельно от активности. Поэтому есть `onAttach()` и `onDetach()` вызовы.
2. View фрагмента может много раз пересоздаваться в рамках одного фрагмента, поэтому есть методы `onCreateView()`, `onViewCreated()` и `onDestroyView()`.





## State loss

Транзакции фрагментов можно запускать не позже `onSaveInstanceState` активности, потому что вся информация о добавленных фрагментах и стеке навигации хранится в `Bundle`.

Пример, когда может произойти краш. Выполняется запрос к серверу, в этот момент пользователь выключает экран. Приходит результат, и нужно открыть следующий экран. Приложение падает т.к. активности находится в `stopped` состоянии.

Однако, мы используем `LiveData` и `ViewModel` для оповещения активности об изменениях, поэтому нам эта проблема не грозит.

# Передача аргументов

Q: У нас есть доступ к конструктору фрагментов (если мы не используем библиотеку от Google). Можем ли мы передавать данные в конструктор фрагмента?

A: Проблема в том, что фрагменты пересоздаются в onCreate активности на основе того, что находится в Bundle. Но у нас есть такой механизм как аргументы.

```
▼ savedInstanceState = {Bundle@9851} "Bundle[{android:viewHierarchyState=Bundle[{android:views={16908290=android.view.AbsSavedState$1@508b046, 2131230771=anc... View
  ► f mClassLoader = {PathClassLoader@9547} "dalvik.system.PathClassLoader[DexPathList[[zip file "/data/app/com.example.tabbedwall-e4ZE7GZFuFIPGwk3KknT0Q=="/b... View
  f mFlags = 0
  ▼ f mMap = {ArrayMap@9885} ArrayMap@9885, size = 5
    ► value[0] = {Bundle@9888} "Bundle[{android:views={16908290=android.view.AbsSavedState$1@508b046, 2131230771=android.view.AbsSavedState$1@508b046, 213... View
    ▼ value[1] = {FragmentManagerState@9889}
      ▼ f mActive = {ArrayList@9895} size = 1
        ▼ 0 = {FragmentManagerState@9899} "FragmentManagerState{com.example.tabbedwall.ui.main.WallFragment (4f631b5b-10ba-4011-a383-408a3e5bb5d9)}; id=0x7f0800a2 tag="tag"
          f mArguments = null
          ► f mClassName = "com.example.tabbedwall.ui.main.WallFragment"
          f mContainerId = 2131230882
```

## Передача аргументов

У фрагментов есть свойство `arguments` типа `Bundle`. В него можно по ключу передавать примитивы или сериализуемые объекты:

```
WallFragment().apply { this: WallFragment
    arguments = bundleOf( ...pairs:
        "userId" to user.id
    )
}
```

Поскольку фрагмент не всегда присоединен к активити, аргументы могут быть `null` и стоит подстраховаться.

```
class WallFragment : Fragment(R.layout.fragment_wall) {

    private val userId: Long by lazy {
        requireArguments().getLong( key: "userId")
    }
}
```

# TransactionTooLargeException

Не стоит забывать о том, что аргументы попадают в Bundle, который используется в onSaveInstanceState активити. Размер Bundle не должен превышать 1024KB, иначе приложение аварийно завершит свою работу.

Из этого можно сделать вывод, что не стоит передавать в аргументы содержимое файлов, большие коллекции. Передавайте минимум необходимой информации. Например, в случае файлов это может быть [URI](#).

---

# Single Activity

В последнее время очень популярен подход, когда в приложении есть только одна Activity на всё приложение. Наши приложения не станут исключением.

Кроме универсальности и независимости, фрагменты предоставляют еще несколько плюсов:

1. Анимации переходов между фрагментами могут быть любыми, в отличие от активности.
2. Возможность использования Shared ViewModel.
3. Проще контролировать ЖЦ.
4. Чище AndroidManifest.xml :)



# ИТОГИ

---

# ИТОГИ

Сегодня мы обсудили с вами следующие вопросы:

- что такое фрагменты и как с ними работать;
- жизненный цикл фрагментов;
- Single Activity.

Полученные знания уже сейчас можно применить в ваших проектах.