

Санкт-Петербургский политехнический университет Петра Великого  
Институт компьютерных наук и технологий  
Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**Лабораторная работа № 3**

Дисциплина: Проектирование мобильных приложений

Тема: Навигация в приложении

Выполнил студент гр. 3530901/90201 \_\_\_\_\_ С.А. Федоров  
(подпись)

Принял старший преподаватель \_\_\_\_\_ А.Н. Кузнецов  
(подпись)

“ \_\_\_\_ ” \_\_\_\_\_ 2021 г.

Санкт-Петербург  
2021

## Оглавление

<b>Цели .....</b>	<b>3</b>
<b>Задачи .....</b>	<b>3</b>
<b>Lifecycle-Aware Components.....</b>	<b>4</b>
<b>Задача 1a. Выполнение Lifecycle-Aware Components Codelabs.....</b>	<b>5</b>
<b>Создание новых Activity .....</b>	<b>9</b>
<b>Задача 2. Решение при помощи метода startActivityForResult .....</b>	<b>10</b>
<b>Задача 3. Решение при помощи флагов Intent .....</b>	<b>15</b>
<b>Задача 4. Дополнение графа навигации новым переходом.....</b>	<b>17</b>
<b>Задача 5. Решение при помощи Fragments, Navigation Graph .....</b>	<b>22</b>
<b>Выводы .....</b>	<b>27</b>
<b>Список источников .....</b>	<b>29</b>
<b>Время выполнения лабораторной работы .....</b>	<b>29</b>
<b>Лабораторная работа №4 .....</b>	<b>30</b>
<b>Задача 2. Тестирование навигации .....</b>	<b>30</b>
<b>Задача 3. Рефакторинг .....</b>	<b>31</b>
<b>Задача 4. Исправление ошибок .....</b>	<b>33</b>
<b>Выводы .....</b>	<b>34</b>

## Цели

- Познакомиться с Google Codelabs и научиться его использовать как способ быстрого изучения новых фреймворков и технологий
- Изучить возможности навигации внутри приложения: создание новых действий, график навигации

## Задачи

1. Познакомьтесь с Lifecycle-aware Components по документации и выполнить кодовые таблицы (в разделе codelabs)
2. Реализуйте навигацию между экранами одного приложения согласно изображению ниже с помощью Activity, Intent и метода startActivityForResult
3. Решить предыдущую задачу с помощью Activity, Intent и флагов Intent либо атрибутов Activity.
4. Дополнить граф навигации новым(-и) переходом(-ами) с целью демонстрации какого-нибудь (на свое усмотрение) атрибута Activity или флага Intent, который еще не использовался для решения задачи. Поясните пример и работу флага/атрибута.
5. Решить исходную задачу с использованием navigation graph. Все Activity должны быть заменены на Fragment, кроме Activity 'About', которая должна остаться самостоятельной Activity. В отчете сравните все решения.

## Lifecycle-Aware Components

*Lifecycle-Aware components* (компоненты с учетом жизненного цикла) выполняют действия в ответ на изменение статуса жизненного цикла другого компонента, например действий и фрагментов. Эти компоненты помогают создавать более структурированный и легкий код, который легче поддерживать.

При использовании этих компонентов можно переместить код зависимых компонентов из методов жизненного цикла в сами компоненты. Действия зависимых компонентов можно реализовывать и в самих методах жизненного цикла, но это приведет к:

- Большому количеству вызовов методов жизненного цикла
- Увеличению кода реализации этих методов
- Затруднению обслуживания методов жизненного цикла из-за большого объема кода
- Увеличению количества ошибок
- Тому, что компонент может запуститься до того, как действие или фрагмент будут остановлены (ошибки при выполнении длительных операций). Например, это может привести к состоянию гонки, когда метод `onStop()` завершается до `OnStart()`, сохраняя компонент в рабочем состоянии дольше, чем это необходимо.)

Пакет `androidx.lifecycle` предоставляет классы и интерфейсы, которые позволяют создавать *lifecycle-aware components* – это компоненты, которые могут автоматически корректировать их поведение на основе текущего состояния жизненного цикла.

Рассмотрим несколько компонентов архитектуры с учетом жизненного цикла для создания приложений Android:

- *ViewModel* - предоставляет способ создания и извлечения объектов, привязанных к определенному жизненному циклу. Сам *ViewModel* обычно хранит состояние данных представления и взаимодействует с другими компонентами, такими как хранилища данных или уровень домена, который обрабатывает бизнес-логику.
- *LifecycleOwner* – это интерфейс, реализованный классами `AppCompatActivity` и `Fragment`. За счет этого интерфейса можно наблюдать за жизненными циклами объектов, реализующих этот

интерфейс за счет подписки (изнутри других компонентов можно следить за изменениями в жизненном цикле владельца)

- *LiveData* – позволяет наблюдать за изменениями данных в нескольких компонентах приложения, не создавая явных жестких путей зависимости между ними. *LiveData* учитывает сложные жизненные циклы компонентов android приложения, включая действия, фрагменты, службы и любого владельца жизненного цикла, определенного в приложении. *LiveData* управляет подписками наблюдателей, приостанавливая подписки на остановленные *LifecycleOwner* объекты и отменяя подписки на *LifecycleOwner* завершенные объекты.

Для изучения вышеописанных компонентов на практике выполним следующий пункт лабораторной работы (Задача 1а).

## **Задача 1а. Выполнение Lifecycle-Aware Components Codelabs.**

### Шаг 1. Настройка среды

На данном шаге нам представлено Android приложение, которое содержит счетчик времени работы приложения. При повороте экрана счетчик сбрасывается, что является ошибкой, так как приложение продолжает свою работу. Эта ошибка возникает из-за того, что при повороте экрана Activity разрушается и создается новое (в горизонтальной раскладке).

### Шаг 2. Добавление ViewModel

Чтобы исправить ошибку сбрасывания счетчика воспользуемся классом *ViewModel*, который создает связи с областью видимости (фрагментом или activity) и будет сохраняться, пока существует область видимости (пока приложение запущено).

То есть при вращении, когда Activity будет разрушен, *ViewModel* останется и новый владелец (Activity в горизонтальной раскладке) просто подключится к *ViewModel* и возьмет оттуда значение счетчика.

Получается, что основная цель ViewModel – это получение и сохранение информации, необходимой для activity или fragment. Обязанность ViewModel заключается в управлении данными для пользовательского интерфейса.

### Шаг 3. Использование LiveData

На данном шаге мы заменяем chronometer на собственный класс, который использует Timer для обновления пользовательского интерфейса каждую секунду. То есть счетчик времени у нас независим (находится в другом классе) и начинается в момент открытия приложения, а для отображения времени мы будем использовать виджет TextView, вместо Chronometer.

Теперь, когда мы используем другой виджет возникает проблема, так как использование ViewModel для сохранения ссылок на экземпляры классов Context или View может привести к утечке памяти (например, когда activity требуется собрать сборщикам мусора). Поэтому следует обернуть ViewModel в LiveData.

Когда мы оборачиваем ViewModel в LiveData, создается так называемый наблюдатель, который будет уведомлен об изменениях обернутых данных, когда владелец будет в состоянии STARTED или RESUMED (теперь наблюдатель будет изменять значение виджета TextView, где отображается время). Когда владелец перейдет в состояние DESTROYED (например, при повороте), тогда наблюдатель будет удален.

Получается, что класс LiveData предназначен для хранения отдельных полей ViewModel, но также может использоваться для разделения данных между различными модулями приложениями независимо друг от друга.

#### Шаг 4. Подписка на события жизненного цикла

В предыдущем пункте при обертывании мы создавали наблюдателя, который реагировал на состояния владельца RESUMED или STARTED, а также DESTROYED, но что, если требуется отслеживать состояние PAUSED или STOPPED (например, когда приложение сворачивается при звонке на устройство). В таких случаях наблюдатель останется и продолжит работу, что может привести к утечкам памяти.

Поэтому при создании наблюдателя следует оформлять подписку (создавать наблюдателя жизненного цикла), чтобы следить за жизненным циклом владельца и в определенный момент отменять эту подписку.

Для решения задачи мы создаем lifecycleOwner, а далее при помощи аннотации объекта (чтобы он мог при необходимости вызывать соответствующий метод) в методе onResume() мы оформляем подписку, а в методе onPause() отменяем ее.

#### Шаг 5. Совместное использование ViewModel между фрагментами

На данном шаге предлагается связать два фрагмента между собой при помощи ViewModel. Как было написано ранее, главная цель ViewModel – это получение и хранение информации необходимой для Fragment или activity.

Поэтому для начала следует получить SeekBarViewModel, далее создать обертку LiveData для созданной ViewModel и описать изменения seekBar.

По итогу получается, что пользователь обновляет информацию во ViewModel, а наблюдатель обновляет seekBar при изменении ViewModel

## Шаг 6. Сохранение состояния ViewModel во время воссоздания процесса

Когда пользователь сворачивает приложение, оно становится невидимо для пользователя и помещается в кэш. Так как системе не хватает памяти, она завершает процессы в кэше, начиная с того, который использовался не так давно. Поэтому при возвращении в приложение, система перезапускает его в новом процессе, следовательно, состояние приложения может быть удалено.

Иногда требуется сохранить состояние приложения или его части, чтобы эта информация не была потеряна в случае остановки процесса (например, свернули окно приложения).

Для сохранения воспользуемся модулем `lifecycle-viewmodel-savedstate`, который обеспечивает доступ к сохраненному состоянию в `ViewModels`.

Для решения задачи по сохранению имени воспользуемся `SavedStateHandler` - это сопоставление “ключ-значение”, которое переживает смерть процесса. С его помощью можно сохранять и восстанавливать примитивы, пакеты и другие типы данных.



## Создание новых Activity

Activity – один из основных компонентов Android-приложения, который представляет собой схему представления Android-приложений. Например, экран, который видит пользователь. Android-приложение может иметь несколько Activity и может переключаться между ними во время выполнения приложения.

Если приложение еще не было запущено, то создается новый Task этого приложения, куда помещается главное Activity. Если же приложение уже было запущено, то система будет работать с созданным ранее стэком.

Все объекты Activity, которые есть в приложении, управляются системой в виде стека Activity, который называется **back stack**. При запуске новой Activity она помещается поверх стека и выводится на экран устройства, пока не появится новая Activity. Когда текущая Activity заканчивает свою работу (например, пользователь делает back action), то она удаляется из стека, и возобновляет работу та Activity, которая ранее была второй в стеке. Получается, что back stack работает по принципам очереди LIFO (last in first out, то есть элементы который пришел последний уйдет первым).

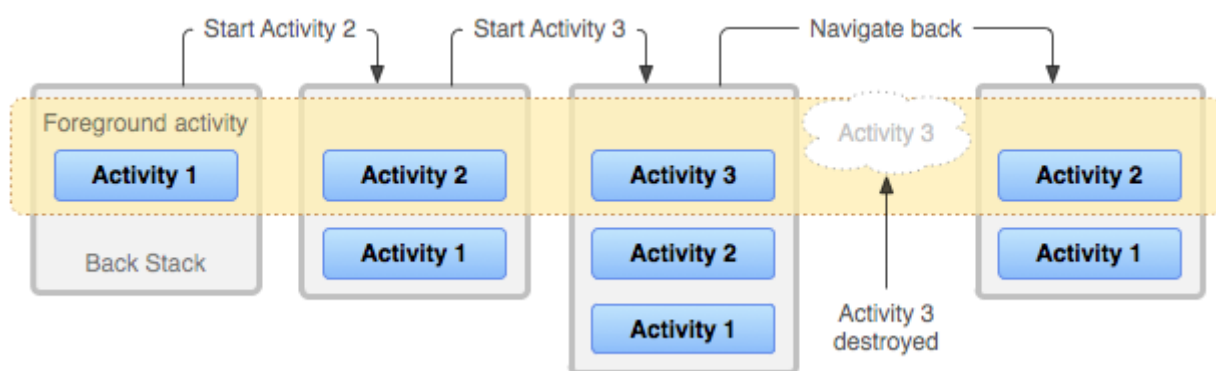


Рис. 1 back stack

Иногда могут возникать ситуации, когда в стэке будет сразу несколько одних и тех же Activity, поэтому стоит следить за тем, чтобы не появлялось дубликатов Activity и порядок Activity правильный. Для этого можно использовать *adb shell dumpsys activity*.

## Задача 2. Решение при помощи метода `startActivityForResult`

По заданию требуется реализовать навигацию между экранами одного приложения с помощью методов Activity: `startActivity`, `startActivityForResult`, `setResult`, `onActivityResult`, `finish` согласно изображению ниже:



Рис. 2 Activity приложения

Рассмотрим подробнее каждый метод Activity:

- `startActivity (Intent)` – метод для создание новой Activity, которая будет помещена на верхушку стека. Принимает единственный аргумент `Intent`, который описывает выполняемое действие
- `startActivityForResult (Intent, int)` – метод создания Activity с ожиданием возврата результата по заданному коду.

- `onActivityResult (int, int, Intent)` – метод, который обрабатывает результат работы Activity после ее завершения. Первый `int` – `requestCode` – идентификатор, второй `int` – `resultCode` – целочисленное представление кода завершения (например, `RESULT_OK = -1`)
- `setResult (int)` – метод, который возвращает данные (устанавливает результат) родителю, при завершении данного activity.
- `finish ()`– метод для завершения работы activity.

Также по заданию требуется, чтобы в `backstack` не было дубликатов одного и того же Activity, поэтому следует предусмотреть всевозможные переходы между Activity и завершать их в нужные моменты.

Теперь перейдем к программной реализации данной задачи:

1. Создадим три файла Activity и соответствующие им xml файлы
2. Для реализации переходов в прямом порядке (с 1 на 2) между Activity по нажатию соответствующей кнопок требуется:
  - a. Вызвать у кнопки метод `setOnClickListener ()`, куда передадим функцию открытия Activity.
  - b. Реализовать функцию открытия следующего Activity. Для этого воспользуемся методом `startActivity()`.

Листинг 1. FirstActivity. Переход к SecondActivity

```
private lateinit var binding: FirstBinding

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding = FirstBinding.inflate(layoutInflater)
    setContentView(binding.root)
    binding.buttonFirstSecond.setOnClickListener { moveToSecond() }

    binding.navView.setNavigationItemSelectedListener{ moveToAbout(it) }
}

private fun moveToSecond() {
    startActivity(Intent(this, SecondActivity::class.java))
}
```

3. Для того, чтобы перейти на предыдущий по порядку экран (со 2 на 1 или с 3 на 2) воспользуемся методом `finish()`.

Листинг 2. ThirdActivity. Переход (возвращение) к SecondActivity

```
private lateinit var binding: ThirdBinding

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding = ThirdBinding.inflate(layoutInflater)
    setContentView(binding.root)
    binding.buttonThirdFirst.setOnClickListener { moveToFirst() }
    binding.buttonThirdSecond.setOnClickListener { moveToSecond() }

    binding.navView.setNavigationItemSelectedListener { moveToAbout(it) }
}

private fun moveToSecond() {
    finish()
}
```

Листинг 3. SecondActivity. Переход (возвращение) к FirstActivity

```
private lateinit var binding: SecondBinding

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding = SecondBinding.inflate(layoutInflater)
    setContentView(binding.root)
    binding.buttonSecondFirst.setOnClickListener { moveToFirst() }
    binding.buttonSecondThird.setOnClickListener { moveToThird() }

    binding.navView.setNavigationItemSelectedListener { moveToAbout(it) }
}

private fun moveToFirst() {
    finish()
}
```

4. Для реализации перехода от ThirdActivity к FirstActivity нельзя воспользоваться алгоритмом, описанным выше, так как возникает проблема дублирования FirstActivity или же закрытия только ThirdActivity, тогда пользователь увидит на экране следующее в стеке Activity (а именно SecondActivity). Поэтому здесь воспользуемся другим алгоритмом:

- а. При создании ThirdActivity из SecondActivity воспользуемся методом `startActivityForResult()`, чтобы при закрытии ThirdActivity получить код завершения

#### Листинг 4. SecondActivity. Переход к ThirdActivity

```
private fun moveToThird() {
    startActivityForResult(Intent(this, ThirdActivity::class.java),
        RESULT_CODE)
}

companion object {
    const val RESULT_CODE = 0
}
```

- б. При реализации перехода из ThirdActivity к FirstActivity требуется выставить результат при помощи метода setResult().

#### Листинг 5. ThirdActivity. Использование setResult()

```
private fun moveToFirst() {
    this.setResult(Activity.RESULT_OK)
    finish()
}
```

- с. Также переопределить метод onActivityResult() в SecondActivity, чтобы при закрытии ThirdActivity с соответствующим кодом (RESULT\_OK, значит пользователь осуществляет переход из 3 в 1), мы также завершили SecondActivity при помощи finish()

#### Листинг 6. SecondActivity. Переопределение onActivityResult()

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    if (requestCode == RESULT_CODE && resultCode == Activity.RESULT_OK) {
        finish()
    }
}
```

5. По заданию также требуется реализовать ActivityAbout, который выводит на экран содержание about.xml. ActivityAbout должен быть доступен из любого Activity, а доступ к нему будет при помощи Navigation Drawer. Для этого в каждом Activity реализуем его вызов

#### Листинг 6. FirstActivity. Пример функции открытия ActivityAbout

```
private fun moveToAbout(item: MenuItem) : Boolean {
    return if (item.itemId == R.id.nav_message) {
        startActivity(Intent(this, ActivityAbout::class.java))
        true
    } else {
        false
    }
}
```

Осуществим проверку нашей реализации:

Запустим приложение и при помощи adb (Android Debug Bridge) проследим за backstack, что в нем не создаются дубликаты одной Activity, а также порядок в стеке не меняется.

```
fsergey@Sasha MINGW64 ~
$ adb shell dumpsys activity activities | grep task2 | grep Hist
* Hist #0: ActivityRecord{a9926a u0 com.example.task2/.FirstActivity t35}

fsergey@Sasha MINGW64 ~
$ adb shell dumpsys activity activities | grep task2 | grep Hist
* Hist #1: ActivityRecord{6725b6c u0 com.example.task2/.SecondActivity t35}
* Hist #0: ActivityRecord{a9926a u0 com.example.task2/.FirstActivity t35}

fsergey@Sasha MINGW64 ~
$ adb shell dumpsys activity activities | grep task2 | grep Hist
* Hist #2: ActivityRecord{56f0c22 u0 com.example.task2/.ThirdActivity t35}
* Hist #1: ActivityRecord{6725b6c u0 com.example.task2/.SecondActivity t35}
* Hist #0: ActivityRecord{a9926a u0 com.example.task2/.FirstActivity t35}

fsergey@Sasha MINGW64 ~
$ adb shell dumpsys activity activities | grep task2 | grep Hist
* Hist #1: ActivityRecord{6725b6c u0 com.example.task2/.SecondActivity t35}
* Hist #0: ActivityRecord{a9926a u0 com.example.task2/.FirstActivity t35}

fsergey@Sasha MINGW64 ~
$ adb shell dumpsys activity activities | grep task2 | grep Hist
* Hist #0: ActivityRecord{a9926a u0 com.example.task2/.FirstActivity t35}

fsergey@Sasha MINGW64 ~
$ adb shell dumpsys activity activities | grep task2 | grep Hist
* Hist #2: ActivityRecord{b647c64 u0 com.example.task2/.ThirdActivity t35}
* Hist #1: ActivityRecord{cbb291e u0 com.example.task2/.SecondActivity t35}
* Hist #0: ActivityRecord{a9926a u0 com.example.task2/.FirstActivity t35}

fsergey@Sasha MINGW64 ~
$ adb shell dumpsys activity activities | grep task2 | grep Hist
* Hist #0: ActivityRecord{a9926a u0 com.example.task2/.FirstActivity t35}

fsergey@Sasha MINGW64 ~
$ adb shell dumpsys activity activities | grep task2 | grep Hist
* Hist #1: ActivityRecord{fb8dae7 u0 com.example.task2/.ActivityAbout t35}
* Hist #0: ActivityRecord{a9926a u0 com.example.task2/.FirstActivity t35}

fsergey@Sasha MINGW64 ~
$ adb shell dumpsys activity activities | grep task2 | grep Hist
* Hist #0: ActivityRecord{a9926a u0 com.example.task2/.FirstActivity t35}

fsergey@Sasha MINGW64 ~
$ adb shell dumpsys activity activities | grep task2 | grep Hist
* Hist #3: ActivityRecord{28b9092 u0 com.example.task2/.ActivityAbout t35}
* Hist #2: ActivityRecord{ebe205c u0 com.example.task2/.ThirdActivity t35}
* Hist #1: ActivityRecord{6caef56 u0 com.example.task2/.SecondActivity t35}
* Hist #0: ActivityRecord{a9926a u0 com.example.task2/.FirstActivity t35}
```

Рис.3 back stack для task2

Исходя из Рис.3 можно сделать вывод, что программа работает корректно (в соответствии с поставленным заданием: дубликаты не создаются и сохраняется порядок Activity в стеке).

Также была проверена ситуация, когда Activity не было зарегистрировано в AndroidManifest.xml. В этом случае при попытке запустить это Activity приложение вылетает.

### Задача 3. Решение при помощи флагов Intent

По заданию требуется решить предыдущую задачу с помощью флагов Intent или же атрибутов Activity. Выберем вариант реализации при помощи флагов Intent.

Для реализации можно оставить метод `startActivity`, а вот логику переходов, которую реализовывал метод `startActivityForResult` и вспомогательные ему (`setResult` и `onActivityResult`) заменим на реализацию при помощи флагов Intent.

Для решения задачи “проблемного” перехода от `ThirdActivity` к `FirstActivity` воспользуемся флагом `FLAG_ACTIVITY_CLEAR_TOP`.

Этот флаг ищет в `backstack` создаваемое Activity. Если находит, то открывает его, а все, что выше – закрывает. То есть позволяет переходить уже к существующей в стеке Activity, попутно закрывая все Activity, лежащие выше.

Листинг 7. Изменения в `SecondActivity`.

```
private fun moveToThird() {  
    startActivity(Intent(this, ThirdActivity::class.java))  
}
```

Листинг 8. Изменения в `ThirdActivity`.

```
private fun moveToFirst() {  
    val intent = Intent(this, FirstActivity::class.java)  
    intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP)  
    startActivity(intent)  
}
```

Теперь протестируем нашу реализацию при помощи флага Intent и проследим за backstack при помощи adb

```
fsergey@Sasha MINGW64 ~
$ adb shell dumpsys activity activities | grep task3 | grep Hist
* Hist #1: ActivityRecord{c289737 u0 com.example.task3/.SecondActivity t37}
* Hist #0: ActivityRecord{46ccac0 u0 com.example.task3/.FirstActivity t37}

fsergey@Sasha MINGW64 ~
$ adb shell dumpsys activity activities | grep task3 | grep Hist
* Hist #2: ActivityRecord{d1d95c5 u0 com.example.task3/.ThirdActivity t37}
* Hist #1: ActivityRecord{c289737 u0 com.example.task3/.SecondActivity t37}
* Hist #0: ActivityRecord{46ccac0 u0 com.example.task3/.FirstActivity t37}

fsergey@Sasha MINGW64 ~
$ adb shell dumpsys activity activities | grep task3 | grep Hist
* Hist #0: ActivityRecord{f39f3c3 u0 com.example.task3/.FirstActivity t37}

fsergey@Sasha MINGW64 ~
$ adb shell dumpsys activity activities | grep task3 | grep Hist
* Hist #3: ActivityRecord{5d395d u0 com.example.task3/.ActivityAbout t37}
* Hist #2: ActivityRecord{201780f u0 com.example.task3/.ThirdActivity t37}
* Hist #1: ActivityRecord{4b384b1 u0 com.example.task3/.SecondActivity t37}
* Hist #0: ActivityRecord{f39f3c3 u0 com.example.task3/.FirstActivity t37}

fsergey@Sasha MINGW64 ~
$ adb shell dumpsys activity activities | grep task3 | grep Hist
* Hist #2: ActivityRecord{201780f u0 com.example.task3/.ThirdActivity t37}
* Hist #1: ActivityRecord{4b384b1 u0 com.example.task3/.SecondActivity t37}
* Hist #0: ActivityRecord{f39f3c3 u0 com.example.task3/.FirstActivity t37}

fsergey@Sasha MINGW64 ~
$ adb shell dumpsys activity activities | grep task3 | grep Hist
* Hist #1: ActivityRecord{4b384b1 u0 com.example.task3/.SecondActivity t37}
* Hist #0: ActivityRecord{f39f3c3 u0 com.example.task3/.FirstActivity t37}
```

Рис.4 back stack для task3

Исходя из Рис.4 можно сделать вывод, что программа работает корректно, так как в backstack нет дубликатов Activity и Activity располагаются в правильном (по заданию) порядке. Также можно увидеть, что при переходе из ThirdActivity в FirstActivity при помощи флага FLAG\_ACTIVITY\_CLEAR\_TOP все Activity, лежащие по стеку выше FirstActivity были завершены.



## Задача 4. Дополнение графа навигации новым переходом

По заданию требуется дополнить граф навигации еще одним переходом и продемонстрировать какой-либо атрибут Activity или же флаг Intent, который еще не использовался для решения задачи.

Для выполнения этого задания дополним граф навигации переходом из ActivityAbout в ThirdActivity.

### Листинг 9. ActivityAbout. Переход к ThirdActivity

```
private fun moveToThird() {  
    startActivity(Intent(this, ThirdActivity::class.java))  
}
```

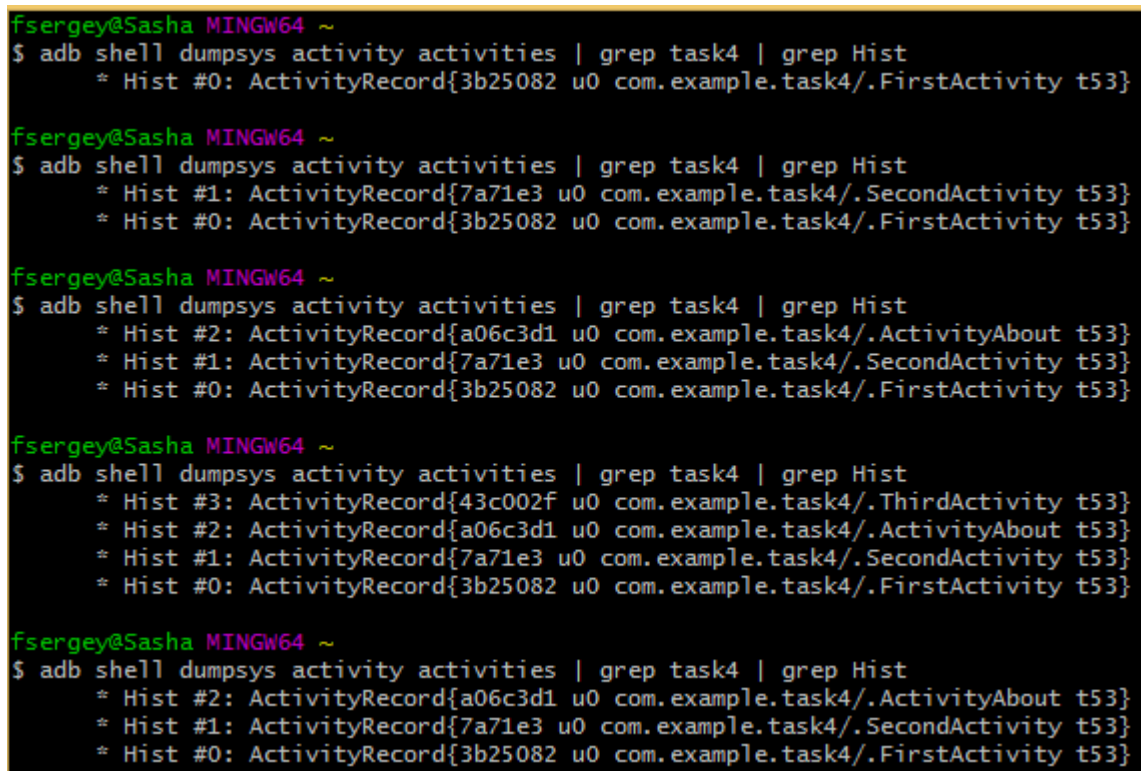
Проследим за backstack при помощи adb:

```
fsergey@Sasha MINGW64 ~  
$ adb shell dumpsys activity activities | grep task4 | grep Hist  
* Hist #0: ActivityRecord{118be5f u0 com.example.task4/.FirstActivity t54}  
  
fsergey@Sasha MINGW64 ~  
$ adb shell dumpsys activity activities | grep task4 | grep Hist  
* Hist #1: ActivityRecord{7d2506a u0 com.example.task4/.SecondActivity t54}  
* Hist #0: ActivityRecord{118be5f u0 com.example.task4/.FirstActivity t54}  
  
fsergey@Sasha MINGW64 ~  
$ adb shell dumpsys activity activities | grep task4 | grep Hist  
* Hist #2: ActivityRecord{b866c10 u0 com.example.task4/.ActivityAbout t54}  
* Hist #1: ActivityRecord{7d2506a u0 com.example.task4/.SecondActivity t54}  
* Hist #0: ActivityRecord{118be5f u0 com.example.task4/.FirstActivity t54}  
  
fsergey@Sasha MINGW64 ~  
$ adb shell dumpsys activity activities | grep task4 | grep Hist  
* Hist #3: ActivityRecord{9382be6 u0 com.example.task4/.ThirdActivity t54}  
* Hist #2: ActivityRecord{b866c10 u0 com.example.task4/.ActivityAbout t54}  
* Hist #1: ActivityRecord{7d2506a u0 com.example.task4/.SecondActivity t54}  
* Hist #0: ActivityRecord{118be5f u0 com.example.task4/.FirstActivity t54}  
  
fsergey@Sasha MINGW64 ~  
$ adb shell dumpsys activity activities | grep task4 | grep Hist  
* Hist #4: ActivityRecord{730696c u0 com.example.task4/.ActivityAbout t54}  
* Hist #3: ActivityRecord{9382be6 u0 com.example.task4/.ThirdActivity t54}  
* Hist #2: ActivityRecord{b866c10 u0 com.example.task4/.ActivityAbout t54}  
* Hist #1: ActivityRecord{7d2506a u0 com.example.task4/.SecondActivity t54}  
* Hist #0: ActivityRecord{118be5f u0 com.example.task4/.FirstActivity t54}  
  
fsergey@Sasha MINGW64 ~  
$ adb shell dumpsys activity activities | grep task4 | grep Hist  
* Hist #5: ActivityRecord{c488a22 u0 com.example.task4/.ThirdActivity t54}  
* Hist #4: ActivityRecord{730696c u0 com.example.task4/.ActivityAbout t54}  
* Hist #3: ActivityRecord{9382be6 u0 com.example.task4/.ThirdActivity t54}  
* Hist #2: ActivityRecord{b866c10 u0 com.example.task4/.ActivityAbout t54}  
* Hist #1: ActivityRecord{7d2506a u0 com.example.task4/.SecondActivity t54}  
* Hist #0: ActivityRecord{118be5f u0 com.example.task4/.FirstActivity t54}
```

Рис.5 back stack для task4 (с ошибками дубликатов)

Исходя из Рис.5 можно сделать вывод, пользователя осуществил переход из FirstActivity -> SecondActivity -> ActivityAbout, а затем напрямую из ActivityAbout -> ThirdActivity, после уже из ThirdActivity -> ActivityAbout, и опять напрямую из ActivityAbout -> ThirdActivity. Этими действиями в backstack появились дубликаты ActivityAbout и ThirdActivity, что говорит об ошибке.

Проверим еще несколько переходов:



```
fsergey@Sasha MINGW64 ~
$ adb shell dumpsys activity activities | grep task4 | grep Hist
* Hist #0: ActivityRecord{3b25082 u0 com.example.task4/.FirstActivity t53}

fsergey@Sasha MINGW64 ~
$ adb shell dumpsys activity activities | grep task4 | grep Hist
* Hist #1: ActivityRecord{7a71e3 u0 com.example.task4/.SecondActivity t53}
* Hist #0: ActivityRecord{3b25082 u0 com.example.task4/.FirstActivity t53}

fsergey@Sasha MINGW64 ~
$ adb shell dumpsys activity activities | grep task4 | grep Hist
* Hist #2: ActivityRecord{a06c3d1 u0 com.example.task4/.ActivityAbout t53}
* Hist #1: ActivityRecord{7a71e3 u0 com.example.task4/.SecondActivity t53}
* Hist #0: ActivityRecord{3b25082 u0 com.example.task4/.FirstActivity t53}

fsergey@Sasha MINGW64 ~
$ adb shell dumpsys activity activities | grep task4 | grep Hist
* Hist #3: ActivityRecord{43c002f u0 com.example.task4/.ThirdActivity t53}
* Hist #2: ActivityRecord{a06c3d1 u0 com.example.task4/.ActivityAbout t53}
* Hist #1: ActivityRecord{7a71e3 u0 com.example.task4/.SecondActivity t53}
* Hist #0: ActivityRecord{3b25082 u0 com.example.task4/.FirstActivity t53}

fsergey@Sasha MINGW64 ~
$ adb shell dumpsys activity activities | grep task4 | grep Hist
* Hist #2: ActivityRecord{a06c3d1 u0 com.example.task4/.ActivityAbout t53}
* Hist #1: ActivityRecord{7a71e3 u0 com.example.task4/.SecondActivity t53}
* Hist #0: ActivityRecord{3b25082 u0 com.example.task4/.FirstActivity t53}
```

Рис.6 back stack для task4 (с ошибками неправильных переходов)

Исходя из Рис.6 можно сделать вывод, пользователя осуществил переход из FirstActivity -> SecondActivity -> ActivityAbout, а затем напрямую из ActivityAbout -> ThirdActivity, после уже из ThirdActivity -> SecondActivity. Судя по backstack последнее действие перехода осуществилось неправильно, так как при переходе из ThirdActivity в SecondActivity мы оказались в ActivityAbout.

Ошибки, представленные на Рис.5 и Рис.6 возникли из-за того, что при дополнении графа навигации новым переходом мы не позаботились о закрытии ActivityAbout при переходе.

Для устранения ошибок воспользуемся атрибутом Activity android:launchMode="singleTask", который задается в AndroidManifest.xml. С этим значением Activity разрешено иметь только один экземпляр в системе, а все существующие activity над singleTask Activity будут завершены.

Листинг 10. AndroidManifest.xml.

```
<activity
    android:name=".ThirddActivity"
    android:exported="true"
    android:launchMode="singleTask">
</activity>
```

Проследим за новой реализацией в backstack при помощи adb:

```
fsergey@Sasha MINGW64 ~
$ adb shell dumpsys activity activities | grep task4 | grep Hist
* Hist #0: ActivityRecord{712f141 u0 com.example.task4/.FirstActivity t55}

fsergey@Sasha MINGW64 ~
$ adb shell dumpsys activity activities | grep task4 | grep Hist
* Hist #1: ActivityRecord{fe33b42 u0 com.example.task4/.SecondActivity t55}
* Hist #0: ActivityRecord{712f141 u0 com.example.task4/.FirstActivity t55}

fsergey@Sasha MINGW64 ~
$ adb shell dumpsys activity activities | grep task4 | grep Hist
* Hist #2: ActivityRecord{a6f69a8 u0 com.example.task4/.ActivityAbout t55}
* Hist #1: ActivityRecord{fe33b42 u0 com.example.task4/.SecondActivity t55}
* Hist #0: ActivityRecord{712f141 u0 com.example.task4/.FirstActivity t55}

fsergey@Sasha MINGW64 ~
$ adb shell dumpsys activity activities | grep task4 | grep Hist
* Hist #3: ActivityRecord{5a5b23e u0 com.example.task4/.ThirdActivity t55}
* Hist #2: ActivityRecord{a6f69a8 u0 com.example.task4/.ActivityAbout t55}
* Hist #1: ActivityRecord{fe33b42 u0 com.example.task4/.SecondActivity t55}
* Hist #0: ActivityRecord{712f141 u0 com.example.task4/.FirstActivity t55}

fsergey@Sasha MINGW64 ~
$ adb shell dumpsys activity activities | grep task4 | grep Hist
* Hist #2: ActivityRecord{a6f69a8 u0 com.example.task4/.ActivityAbout t55}
* Hist #1: ActivityRecord{fe33b42 u0 com.example.task4/.SecondActivity t55}
* Hist #0: ActivityRecord{712f141 u0 com.example.task4/.FirstActivity t55}

fsergey@Sasha MINGW64 ~
$ adb shell dumpsys activity activities | grep task4 | grep Hist
* Hist #3: ActivityRecord{ec126ee u0 com.example.task4/.ThirdActivity t55}
* Hist #2: ActivityRecord{a6f69a8 u0 com.example.task4/.ActivityAbout t55}
* Hist #1: ActivityRecord{fe33b42 u0 com.example.task4/.SecondActivity t55}
* Hist #0: ActivityRecord{712f141 u0 com.example.task4/.FirstActivity t55}

fsergey@Sasha MINGW64 ~
$ adb shell dumpsys activity activities | grep task4 | grep Hist
* Hist #2: ActivityRecord{a6f69a8 u0 com.example.task4/.ActivityAbout t55}
* Hist #1: ActivityRecord{fe33b42 u0 com.example.task4/.SecondActivity t55}
* Hist #0: ActivityRecord{712f141 u0 com.example.task4/.FirstActivity t55}
```

Рис.7 back stack для task4 (исправление ошибки появления дубликатов)

Исходя из Рис.7 можно сделать вывод, что атрибут Activity android:launchMode= "singleTask" для ActivityAbout решил проблему появления дубликатов (первый фрагмент), но не исправил ошибку неправильных переходов (фрагмент 2: при переходе ThirdActivity -> SecondActivity мы оказались в ActivityAbout).

Для исправления ошибки неправильных переходов теперь надо учитывать переход из ActivityAbout -> ThirdActivity. Теперь алгоритм перехода ThirdActivity -> SecondActivity при помощи finish не будет работать, так как метод finish завершает ThirdActivity и на вершине стека оказывается ActivityAbout. Поэтому изменим реализацию перехода ThirdActivity -> SecondActivity при помощи флага FLAG\_ACTIVITY\_CLEAR\_TOP, который закроет ActivityAbout, так как это activity по стеку находится выше, чем SecondActivity.

Листинг 11. ThirdActivity. Переход к SecondActivity
---

<pre>private fun moveToSecond() {     val intent = Intent(this, SecondActivity::class.java)     intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP)     startActivity(intent) }</pre>
---

Проверим backstack с такими же переходами, что и на Рис.7

```
fsergey@Sasha MINGW64 ~
$ adb shell dumpsys activity activities | grep task4 | grep Hist
* Hist #0: ActivityRecord{73c2007 u0 com.example.task4/.FirstActivity t57}

fsergey@Sasha MINGW64 ~
$ adb shell dumpsys activity activities | grep task4 | grep Hist
* Hist #1: ActivityRecord{7929878 u0 com.example.task4/.SecondActivity t57}
* Hist #0: ActivityRecord{73c2007 u0 com.example.task4/.FirstActivity t57}

fsergey@Sasha MINGW64 ~
$ adb shell dumpsys activity activities | grep task4 | grep Hist
* Hist #2: ActivityRecord{76a1f8e u0 com.example.task4/.ActivityAbout t57}
* Hist #1: ActivityRecord{7929878 u0 com.example.task4/.SecondActivity t57}
* Hist #0: ActivityRecord{73c2007 u0 com.example.task4/.FirstActivity t57}

fsergey@Sasha MINGW64 ~
$ adb shell dumpsys activity activities | grep task4 | grep Hist
* Hist #3: ActivityRecord{5c37e54 u0 com.example.task4/.ThirdActivity t57}
* Hist #2: ActivityRecord{76a1f8e u0 com.example.task4/.ActivityAbout t57}
* Hist #1: ActivityRecord{7929878 u0 com.example.task4/.SecondActivity t57}
* Hist #0: ActivityRecord{73c2007 u0 com.example.task4/.FirstActivity t57}

fsergey@Sasha MINGW64 ~
$ adb shell dumpsys activity activities | grep task4 | grep Hist
* Hist #2: ActivityRecord{76a1f8e u0 com.example.task4/.ActivityAbout t57}
* Hist #1: ActivityRecord{7929878 u0 com.example.task4/.SecondActivity t57}
* Hist #0: ActivityRecord{73c2007 u0 com.example.task4/.FirstActivity t57}

fsergey@Sasha MINGW64 ~
$ adb shell dumpsys activity activities | grep task4 | grep Hist
* Hist #3: ActivityRecord{bed2484 u0 com.example.task4/.ThirdActivity t57}
* Hist #2: ActivityRecord{76a1f8e u0 com.example.task4/.ActivityAbout t57}
* Hist #1: ActivityRecord{7929878 u0 com.example.task4/.SecondActivity t57}
* Hist #0: ActivityRecord{73c2007 u0 com.example.task4/.FirstActivity t57}

fsergey@Sasha MINGW64 ~
$ adb shell dumpsys activity activities | grep task4 | grep Hist
* Hist #1: ActivityRecord{9b79fa u0 com.example.task4/.SecondActivity t57}
* Hist #0: ActivityRecord{73c2007 u0 com.example.task4/.FirstActivity t57}
```

Рис.8 back stack для task4 (исправление ошибки неправильных переходов)

Исходя из Рис.8 можно сделать вывод, что все ошибки при дополнении графа навигации новым переходом были устранены. Для этого использовался флаг `FLAG_ACTIVITY_CLEAR_TOP` и ранее не использованный атрибут Activity `android:launchMode="singleTask"`.

## Задача 5. Решение при помощи Fragments, Navigation Graph

По заданию требуется решить исходной задачу (см [Задача 2](#)) с использованием navigation graph. Также все Activity должны быть заменены на Fragment, кроме ActivityAbout, которая должна остаться самостоятельной Activity.

Fragment представляет собой часть пользовательского интерфейса, которую можно многократно использовать. Фрагмент определяет и управляет своим собственным макетом, имеет свой собственный жизненный цикл и может обрабатывать свои собственные входные события. Фрагменты не могут жить сами по себе - они должны размещаться в Activity или другом фрагменте. Поэтому в AndroidManifest.xml фрагменты прописывать не нужно

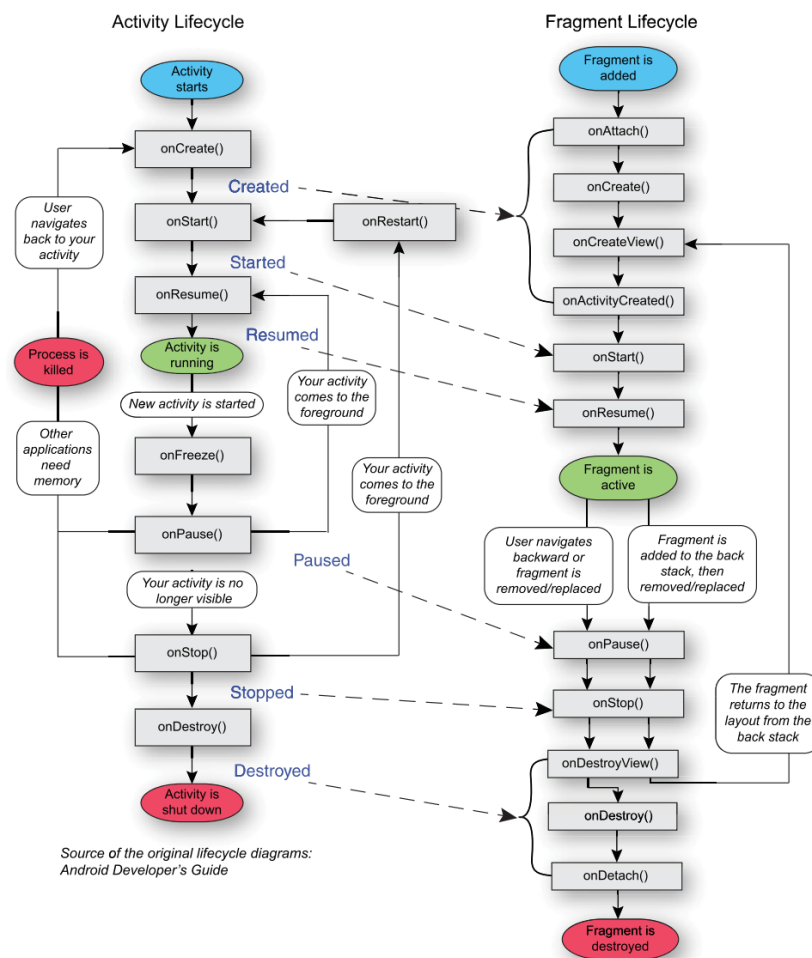


Рис.9 Жизненный цикл для Activity и Fragment



Navigation Graph – это ресурс, который определяет пути, доступные пользователю в приложении. Он показывает визуально все пункты назначения, которые могут быть достигнуты из данного пункта назначения. Редактор навигации в Android Studio отображает Navigation Graph наглядно:

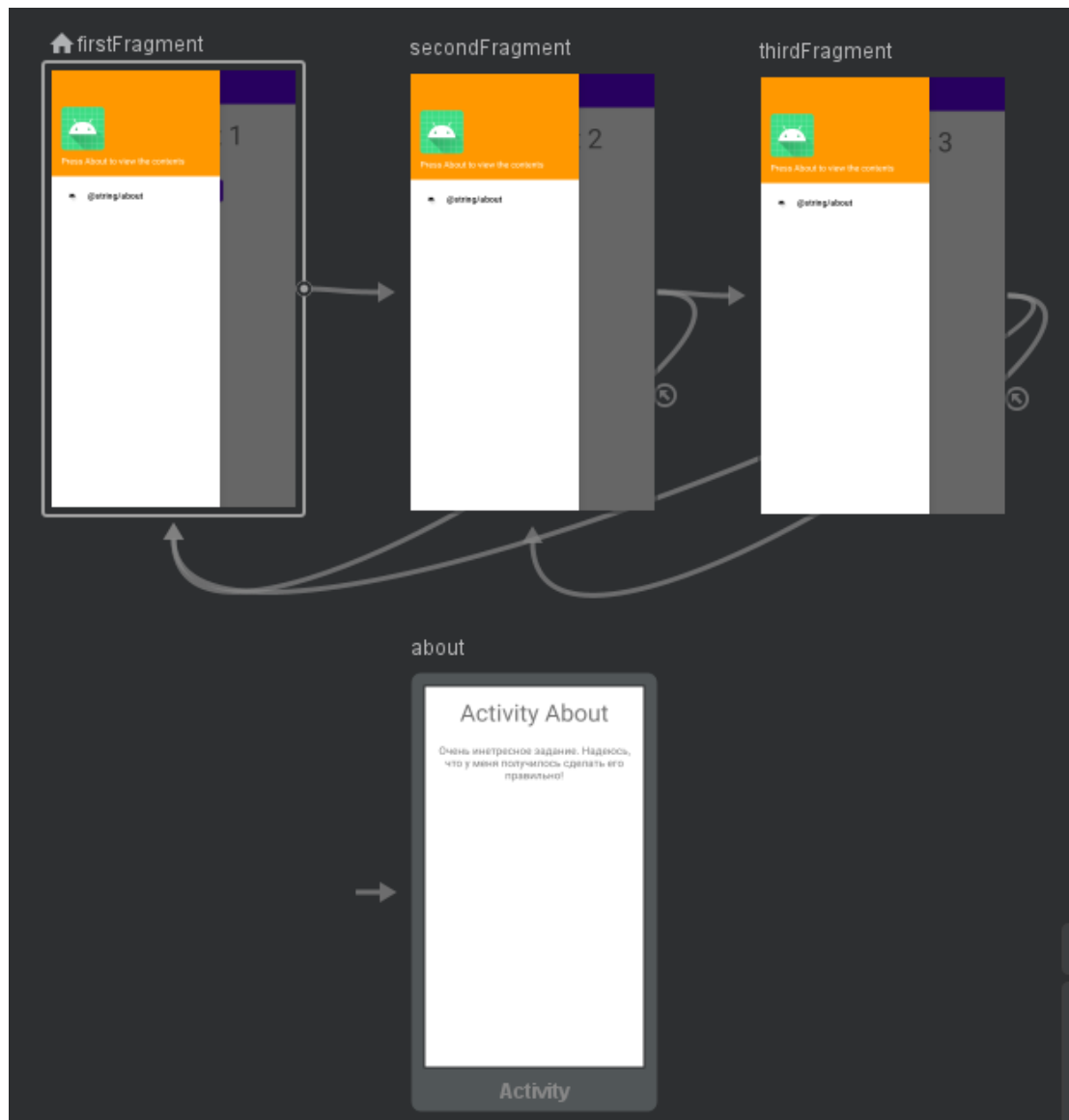


Рис.9 Navigation Graph для задачи 5

Создание navigation Graph:

- Создаем фрагмент и далее указываем несколько параметров:
  - android:id – чтобы использовать этот фрагмент при создании переходов (action)
  - android:name – связываем фрагмент с его java/kotlin классом

- android:label – подписываем наш fragment в окне Navigation Graph (для визуального удобства)
- tools: layout – связываем fragment с layout (визуальным представлением, которое создано в соответствующем layout.xml)
- Также внутри фрагмента можно создать action, благодаря которому можно осуществить переход из одного fragment в другой
  - android:id – идентификатор этого перехода
  - app:destination – место назначения перехода (тут указываем android:id нужного fragment)
  - app:popUpTo – “чистит” backstack до заданного фрагмента, выталкивая при этом фрагменты с верхушки стека, пока не будет достигнут указанный fragment
  - app:popUpToInclusive – требуется указывать значение “true”, так как принцип работы popUpTo заключается в том, что он завершает все значения до заданного. То есть, если не использовать app:popUpToInclusive или же указать значение “false”, то в back stack окажутся дубликаты одного и того fragment, к которому мы хотели перейти в данном action. Если же указать “true”, то это будет сигнализировать о том, что надо исключить дубликаты из back stack.
- Исходя из Рис.9 можно увидеть, что About имеет специфическую стрелку и является Activity. По заданию требуется осуществлять переход к ActivityAbout из любого Fragment, следовательно, этот переход можно указать глобальным. Для этого требуется вынести action из тела fragment/activity.



## Листинг 12. Navigation Graph

```
<?xml version="1.0" encoding="utf-8"?>
<navigation 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_graph"
    app:startDestination="@id/firstFragment">

    <fragment
        android:id="@+id/firstFragment"
        android:name="com.example.task5.FirstFragment"
        android:label="first_fragment"
        tools:layout="@layout/first_fragment" >
        <action
            android:id="@+id/action_firstFragment_to_secondFragment"
            app:destination="@id/secondFragment" />
    </fragment>

    <fragment
        android:id="@+id/secondFragment"
        android:name="com.example.task5.SecondFragment"
        android:label="second_fragment"
        tools:layout="@layout/second_fragment" >
        <action
            android:id="@+id/action_secondFragment_to_thirdFragment"
            app:destination="@id/thirdFragment" />
        <action
            android:id="@+id/action_secondFragment_to_firstFragment"
            app:destination="@id/firstFragment"
            app:popUpTo="@id/firstFragment"
            app:popUpToInclusive="true"/>
    </fragment>

    <fragment
        android:id="@+id/thirdFragment"
        android:name="com.example.task5.ThirdFragment"
        android:label="third_fragment"
        tools:layout="@layout/third_fragment" >
        <action
            android:id="@+id/action_thirdFragment_to_secondFragment"
            app:destination="@id/secondFragment"
            app:popUpTo="@id/secondFragment"
            app:popUpToInclusive="true"/>
        <action
            android:id="@+id/action_thirdFragment_to_firstFragment"
            app:destination="@id/firstFragment"
            app:popUpTo="@id/firstFragment"
            app:popUpToInclusive="true"/>
    </fragment>

    <activity
        android:id="@+id/about"
        android:name="com.example.task5.ActivityAbout"
        android:label="About"
        tools:layout="@layout/about" />

    <action
        android:id="@+id/global_about"
        app:destination="@id/about" />
</navigation>
```

Теперь можно перейти к реализации фрагментов. Для этого требуется:

- Указать наследование от класса `Fragment`
- Переопределить метод `onCreateView`, который является аналогом метода `setContentView()` для `Activity`. Этот метод будет возвращать `View`.
  - Для осуществления переходов по нажатию кнопок будем использовать метод `findNavController()`, чтобы по нему делать навигацию и указывать соответствующие `android:id` для `action`

#### Листинг 13. Пример класса `Fragment` (`SecondFragment`)

```
class SecondFragment : Fragment() {  
  
    override fun onCreateView(  
        inflater: LayoutInflater,  
        container: ViewGroup?,  
        savedInstanceState: Bundle?  
    ): View {  
        super.onCreate(savedInstanceState)  
        val binding = SecondFragmentBinding.inflate(inflater)  
        val navController = findNavController()  
  
        binding.buttonSecondFirst.setOnClickListener {  
            navController.navigate(R.id.action_secondFragment_to_firstFragment)  
        }  
  
        binding.buttonSecondThird.setOnClickListener {  
            navController.navigate(R.id.action_secondFragment_to_thirdFragment)  
        }  
  
        binding.navView.setNavigationItemSelectedListener { moveToAbout(it) }  
  
        return binding.root  
    }  
  
    private fun moveToAbout(item: MenuItem) : Boolean {  
  
        return if (item.itemId == R.id.nav_message) {  
            findNavController().navigate(R.id.global_about)  
            true  
        } else {  
            false  
        }  
    }  
}
```

## Выводы

- В ходе выполнения данной лабораторной работы (задача 1) был выполнен codelabs по Lifecycle-Aware Components
  - Здесь я изучил ViewModel, которая служит для получения и сохранения информации, необходимой для activity или fragment
  - Изучил LiveData – это обертка для ViewModel, которая создает наблюдателя, который будет уведомлен об изменениях обернутых данных, когда владелец будет в состоянии STARTED или RESUMED. Класс LiveData служит для хранения отдельных полей ViewModel
  - Изучил подписку на события жизненного цикла, а также интерфейс LifecycleOwner
  - При помощи вышеописанных компонентов различными способами была решена задача счетчика времени работы приложения
- Была решена задача переключения между Activity в рамках одного приложения. При решении таких задач требуется следить за back stack, чтобы не появлялись дубликаты одной activity. Для решения проблемы с появлением одинаковых сущностей использовалось два подхода:
  - Использование метода startActivityForResult(), чтобы при закрытии созданной Activity получить код завершения. При завершении созданной Activity использовался метод setResult, чтобы выставить результат, который обрабатывается в onActivityResult (в Activity, из которой создалась эта), чтобы при соответствующем переходе завершить и данную Activity.
  - Использование атрибутов Activity, чтобы данная Activity могла иметь только одну сущность и завершать дубликаты. Также использование флагов Intent, чтобы искать в стеке уже открытые Activity, и при переходе завершать те, что лежат выше по стеку

- В последнем задании функционал перехода между экранами был реализован при помощи Navigation Graph.
  - Все activity были заменены на fragment, кроме ActivityAbout.
  - При использовании fragment потребовалось использовать MainActivity, так как fragment не может быть сам по себе. MainActivity – это стартовый activity для запуска NavHostFragment – пустого контейнера для выполнения автономной навигации
- Использование Navigation Graph дает большей наглядности при реализации переходов и уменьшает время написания кода, поэтому лучше использовать его, чем переходы с помощью activity, для реализации сложных и больших проектов

Ссылка на github: [https://github.com/sergeyfedorov02/Android\\_labs.git](https://github.com/sergeyfedorov02/Android_labs.git)

## Список источников

<https://developer.android.com/>

<https://github.com/andrei-kuznetsov/android-lectures>

Видео-гайд для создания Navigation Drawer:

<https://www.youtube.com/watch?v=fGcMLu1GJEc&t=4s>

## Время выполнения лабораторной работы

- Задача 1 – 120 минут
- Задача 2 – 240 минут
- Задача 3 – 30 минут
- Задача 4 – 30 минут
- Задача 5 – 260 минут
- Заполнение отчета – 680 минут (отчет заполнялся по мере решения задач, поэтому время решения задачи = отчет + написание кода)

## Лабораторная работа №4

### Задача 2. Тестирование навигации

#### Задание

Возьмите референсную реализацию приложения из Лаб №3 о навигации (решение с помощью navgraph). Напишите UI тесты, проверяющие навигацию между 4мя исходными Fragments/Activity (1-2-3-About).

#### Решение

Для проверки навигации были написаны тесты:

- Проверки фрагментов (fragmentFirst, fragmentSecond, fragmentThird) в которых осуществлялась проверка
  - Наличия данного фрагмента на экране
  - Проверка содержимого фрагмента
  - Проверка сохранения содержимого после поворота экрана
  - Тестирование кнопок переходов к другим фрагментам
- Проверка ActivityAbout при открытии из любого фрагмента (aboutFirstFragment, aboutSecondFragment, aboutThirdFragment)
  - Проверка наличия на экране ActivityAbout после перехода к нему
  - Проверка содержимого About
  - Проверка сохранения содержимого после поворота экрана
  - Возвращение назад при помощи кнопки с экрана самого устройства
- Тестирование содержимого стека при различных переходах между фрагментами и открытиями ActivityAbout (пример имени теста: chaeckBackStack\_1\_2\_1)
- Тестирование навигации при помощи кнопки, которая была создана в самом приложении. То есть, возвращение назад осуществляется при помощи кнопки ActionBar (пример имени теста: navigationBackFirst)

## Задача 3. Рефакторинг

### Задание

После того, как в результате решения задачи появилась 2 "достаточно хорошая" реализация тестов, перенесены эти тесты в проект из Лаб. 3.

### Решение

Для решения данной задачи была взята реализация приложения с навигационным графом из lab3/Task5 и реализована вместе с тестами в lab4/Task3 (lab3/Task5 осталось без изменений).

После того, как тесты были вставлены и был произведен рефакторинг, выяснилось, что большая часть тестов не проходит. В результате было выявлено несколько ошибок:

- Тесты, которые содержали переход к ActivityAbout и возвращение обратно, а потом еще какой-нибудь переход не проходили, так как боковое меню не убиралось и перекрывало кнопки. Кнопки можно было использовать, но переход к другому фрагменту не осуществлялся, и следующая проверка на наличие фрагмента экране не проходила.
  - Для решения данной проблемы в код перехода к ActivityAbout из fragment была добавлена проверка на то, открыто ли боковое меню, и если открыто, то закрываем его

Листинг 14. Изменения перехода к ActivityAbout из fragment (FirstFragment)

```
private fun moveToAbout(item: MenuItem) : Boolean {  
  
    if (binding.drawer.isDrawerOpen(GravityCompat.START)) {  
        binding.drawer.closeDrawer(GravityCompat.START)  
    }  
  
    return if (item.itemId == R.id.aboutActivity) {  
        findNavController().navigate(R.id.global_about)  
        true  
    } else {  
        false  
    }  
}
```

- Также не проходили тесты, которые осуществляли обратную навигацию при помощи кнопки ActionBar, так как эта кнопка не была создана
  - Для решения этой проблемы в класс MainActivity были внесены изменения, чтобы добавить эту кнопку

#### Листинг 15. Изменения в MainActivity

```
class MainActivity : AppCompatActivity() {  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        val binding = ActivityMainBinding.inflate(layoutInflater)  
        setContentView(binding.root)  
        supportActionBar?.setDisplayHomeAsUpEnabled(true)  
    }  
  
    override fun onSupportNavigateUp(): Boolean {  
        onBackPressed()  
        return super.onSupportNavigateUp()  
    }  
}
```

- Теперь, чтобы все кнопка отображалась на экране в AndroidManifest.xml для ActivityAbout надо указать, что она является дочерней от MainActivity
- Следующая проблема заключается в том, что при переходе из ActivityAbout мы всегда попадаем в FirstFragment. Чтобы решить эту проблему, также в AndroidManifest.xml для MainActivity добавим атрибут android:launchMode="singleTop"

Теперь все тесты проходят успешно.



## Задача 4. Исправление ошибок

### Задание

Примените разработанные тесты к каждой из 3х задач Лаб.3 о навигации (кроме задачи с демонстрацией флага/атрибута): `startActivityForResult`, `startActivity+Intent.flags`, `navgraph`.

### Решение

Для решения данной задачи были скопированы реализации `lab3/Task2` и `lab3/Task3` в соответствующие проекты `lab4/Task4/Task2_lab3` и `lab4/Task4/Task3_lab3`, куда и были внесены изменения для исправления ошибок.

Все тесты и рефакторинг были выполнены также, как и в предыдущем пункте. Также была добавлена проверка на наличие открытого бокового меню при переходе к `ActivityAbout` (реализация такая же, как и в предыдущем пункте).

Отличие от предыдущего пункта заключается в том, что:

- Чтобы добавить кнопку навигации вверх потребовалось в каждом `Activity` внести изменения (`AndroidManifest.xml` остался без изменений).

#### Листинг 16. Изменения для навигации вверх (FirstActivity)

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding = FirstBinding.inflate(layoutInflater)
    setContentView(binding.root)
    binding.bnToSecond.setOnClickListener { moveToSecond() }

    supportActionBar?.setDisplayHomeAsUpEnabled(true)

    binding.drawerNavView.setNavigationItemSelectedListener { moveToAbout(it) }
}

override fun onSupportNavigateUp(): Boolean {
    onBackPressed()
    return super.onSupportNavigateUp()
}
```

То есть был переопределен метод `onSupportNavigateUp()`, где мы возвращаемся назад, если кнопка была нажата, а также в методе `onCreate` мы добавляем строчку `“supportActionBar?.setDisplayHomeAsUpEnabled(true)”`, чтобы отобразить эту кнопку на экране.

## Выводы

- В ходе выполнения данной лабораторной работы я на практике ознакомился с основами UI тестов
- Также был выполнен рефакторинг реализаций задачи по навигации из lab3 (Task2, Task3, Task5). Модифицированный код был помещен в новый проект lab4 (Task3 для lab/Task5, Task4 для lab3/Task2 и lab3/Task3)
- При добавлении тестов были выявлено и исправлено несколько ошибок
  - Ошибка наличия фрагмента на экране при переходе к нему из другого фрагмента, в котором до этого был открыт `ActivityAbout`. Так как боковое меню не закрывалось, то и новый фрагмент не появлялся на экране
    - В `moveToAbout` добавлена проверка на наличие открытого бокового меню и закрытие его
  - Ошибка отсутствия кнопки навигации вверх из `ActionBar`
    - Для Task5 были внесены изменения в `MainActivity` и `AndroidManifest.xml`
    - Для Task2 и Task3 изменения были внесены в сами `Activity`, где была добавлена сама кнопка в методе `onCreate()` и для возвращения назад переопределен метод `onSupportNavigateUp()`