Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерный наук и технологий

Высшая школа интеллектуальных систем и суперкомпьютерных технологий.

**Отчёт по лабораторной работе №3**

**Дисциплина**: Проектирование мобильных приложений

**Тема**: Lifecycle компоненты. Навигация в приложении. Вариант 13

Выполнил студент гр. 3530901/80201: ­­­­­­­­­­­­­ ­ В.Д. Петров

(подпись)

Преподаватель: ­А.Н. Кузнецов

(подпись)

Санкт-Петербург

2020

**СОДЕРЖАНИЕ**

Введение…………………………………………………………………………….3

Задача №1. Обработка жизненного цикла с помощью Lifecycle-Aware компонентов………………………………………………………………..………4

Задача №2. Навигация (startActivityForResult)……..……………………………..6

Задача №3. Навигация (флаги Intent/атрибуты Activity)………..…………..…..24

Задача №4. Навигация (флаги Intent/атрибуты Activity)………………..……....28

Задача №5. Навигация (Fragments, Navigation Graph)………………..…………34

Заключение……………………………………………………………..…………40

Список использованных источников……………………………………….…...41

**ВВЕДЕНИЕ**

Целями данной работы является знакомство с методом обработки жизненного цикла activity/fragment с помощью Lifecycle-Aware компонентов, а также изучение основных возможностей навигации внутри приложения: создание новых activity, navigation graph.

**ЗАДАЧА №1. ОБРАБОТКА ЖИЗНЕННОГО ЦИКЛА С ПОМОЩЬЮ LIFECYCLE-AWARE КОМПОНЕНТОВ**

Lifecycle-aware компоненты выполняют действия в ответ на изменение состояния жизненного цикла другого компонента (activity или фрагмента). Такие компоненты помогают создавать лучше организованный и более легкий для чтения код, который будет легче поддерживать. Типичным приемом является реализация действий нужных компонентов внутри методов жизненного цикла activity или фрагмента. Однако такой прием ведет к плохо организованному коду и увеличению числа ошибок. Используя lifecycle-aware компоненты, можно переместить код действий нужных компонентов из методов жизненного цикла в сам компонент. Lifecycle-aware компоненты могут автоматически настраивать свое поведение в зависимости от текущего состояния жизненного цикла activity или фрагмента.

Lifecycle – это класс, который хранит информацию о текущем состоянии жизненного цикла компонента и позволяет другим объектам просматривать это состояние. Lifecycle класс использует два перечисления для того, чтобы следить за текущим состоянием жизненного цикла компонента: event и state. Event – события жизненного цикла, отправляемые фреймворком и классом Lifecycle. Эти события являются отражением на события обратных вызовов в activity или фрагменте. State – текущее состояние компонента, которое отслеживает Lifecycle класс.

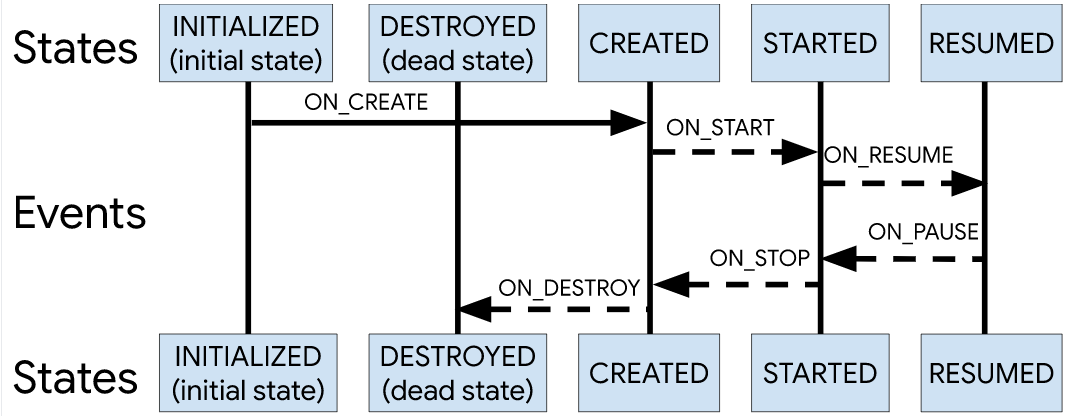


Рис.1 Графическое изображение изменений в перечислениях State и Event

LifecycleOwner – интерфейс с одним методом, который указывает на то, что класс имеет объект класса Lifecycle. Он имеет только метод getLifecycle(), который должен быть переопределен. Этот интерфейс абстрагирует владение объектом Lifecycle от отдельных классов activity и фрагментов и позволяет писать компоненты, которые работают с ними. Компоненты, переопределяющие LifecycleObserver работают с компонентами, переопределяющими LifecycleOwner, владелец предоставляет жизненный цикл, наблюдать за которым может зарегистрироваться наблюдатель.

После прохождения codelabs были лучше изучены и опробованы на практике классы ViewModel и LiveData. ViewModel предоставляет способ создавать и извлекать объекты, привязанные к определенному жизненному циклу. ViewModel обычно хранит данные view и взаимодействует с другими компонентами. ViewModel переживает configuration change события, как, например, поворот экрана. Фреймворк оставляет ViewModel до тех пор, пока существует LifecycleOwner. ViewModel не уничтожается, если ее владелец уничтожается при configuration change событиях, вместо этого вновь созданный владелец переподключается к созданной ViewModel (Рис.2)

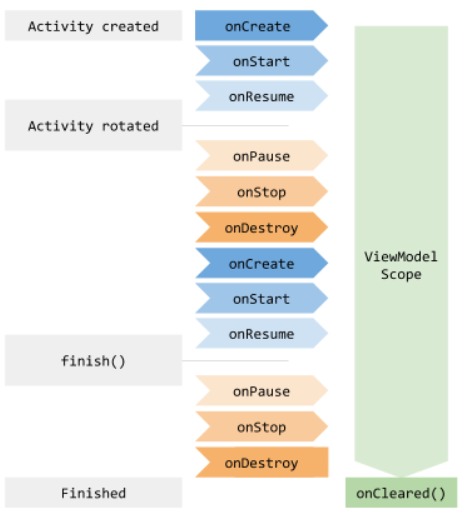


Рис. 2 Жизненный цикл ViewModel

LiveData позволяет наблюдать за изменениями данных в нескольких компонентах без создания явных путей зависимостей между ними. LiveData управляет подписками наблюдателей, ставит на паузу подписки у остановившегося компонента и закрывает подписки у компонентов, завершивших исполнение.

**ЗАДАЧА №2. НАВИГАЦИЯ (startActivityForResult)**

Необходимо реализовать навигацию между экранами одного приложения согласно изображению (Рис. 3) с помощью Activity, Intent и метода startActivityForResult. Исходный код для данного задания представлен по ссылке: <https://github.com/ADsty/labs-android_labs/tree/master/lab3.2>

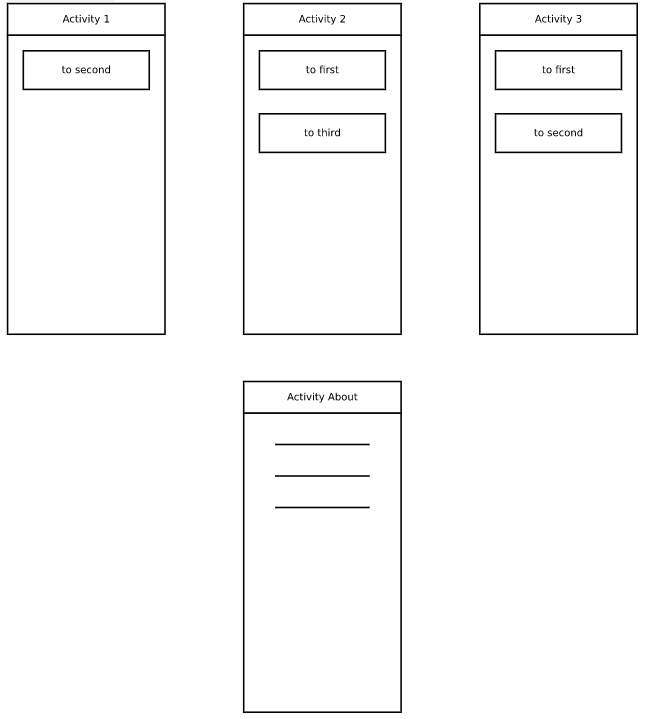


Рис. 3 Изображение навигации между экранами реализуемого приложения

Данное приложение начинает работу с Activity 1 (Рис. 4).



Рис. 4 Итоговый вариант Activity 1

В ней есть кнопка go\_to\_second, которая отображает Activity 2, используя метод startActivity.

Листинг 1 – исходный код для кнопки go\_to\_second

val button1: Button = findViewById(R.id.*from\_1\_to\_2*)  
button1.setOnClickListener**{** val intent = Intent(this, Activity2::class.*java*)  
 startActivity(intent)  
**}**

А также, согласно варианту, есть “шторка” Navigation Drawer, в которой есть кнопка About (Рис.5)

Листинг 2 – исходный код для открытия activity About

private val mOnNavigationItemSelectedListener = NavigationView.OnNavigationItemSelectedListener **{** item **->** if(item.*itemId* == R.id.*about*) {  
 val intent = Intent(this, ActivityAbout::class.*java*)  
 startActivity(intent)  
 }  
 return@OnNavigationItemSelectedListener true  
**}**



Рис. 5 Шторка Navigation Drawer

При нажатии на кнопку отображается Activity About (Рис. 6).



Рис. 6 Результат нажатия на кнопку About

При нажатии на кнопку go\_to\_second, отображается Activity 2 (Рис. 7)

  
Рис. 7 Отображение Activity 2 на экране

В ней находится идентичная шторка Navigation Drawer и две кнопки- go\_to\_third и go\_to\_first. Первая кнопка отображает Activity 3 и использует метод startActivityForResult.

Листинг 3 – исходный код для кнопки go\_to\_third

val button2: Button = findViewById(R.id.*from\_2\_to\_3*)  
button2.setOnClickListener**{** val intent = Intent(this, Activity3::class.*java*)  
 startActivityForResult(intent, 0)  
**}**

Также в Activity 2 переопределен метод onActivityResult, который при получении кода “1” завершает работу Activity 2. Данный код будет возвращать Activity 3 методом setResult при нажатии кнопки go\_to\_first в нем.

Листинг 4 – метод onActivityResult

override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {  
 super.onActivityResult(requestCode, resultCode, data)  
 if (resultCode == 1) {  
 finish()  
 }  
}

Вторая кнопка завершает Activity 2 методом finish, после чего пользователь возвращается в Activity 1.

Листинг 5 – исходный код для кнопки go\_to\_first

val button1: Button = findViewById(R.id.*from\_2\_to\_1*)  
button1.setOnClickListener **{** finish() **}**

Проверим что шторка появляется в Activity 2 (Рис. 8).



Рис. 8 Отображение шторки Navigation Drawer в Activity 2

Теперь проверим работу второй кнопки (Рис. 9).



Рис. 9 Результат нажатия кнопки go\_to\_first

Как можно видеть, Activity 2 было закрыто и приложение вернулось к Activity1. Для того, чтобы удостовериться, что в backstack лежит только Activity 1, нажмем кнопку back один раз, ожидается что после нажатия приложение остановится (Рис. 10).



Рис. 10 Результат нажатия кнопки back

Как можно видеть, в backstack хранилось лишь Activity 1, Activity 2 было завершено и по нажатию кнопки back мы остановили приложение. Вернемся в приложение и нажмем кнопку go\_to\_third (Рис. 11).



Рис. 11 Результат нажатия кнопки go\_to\_third

На экране отобразилось Activity 3, в нем также есть шторка Navigation Drawer, а также две кнопки – go\_to\_second и go\_to\_first. Первая кнопка завершает работу Activity 3 методом finish, а вторая кнопка помимо этого методом setResult возвращает код “1”, который помимо этого завершит и работу Activity 2, из-за чего на экране отобразится Activity 1, а в backstack кроме нее не будет ничего.

Листинг 6 – исходный код для кнопки go\_to\_first

val button1: Button = findViewById(R.id.*from\_3\_to\_1*)  
button1.setOnClickListener **{** setResult(1)  
 finish()  
**}**

Для начала проверим, на месте ли шторка (Рис. 12).



Рис. 12 Проверка шторки в Activity 3

Затем нажмем на кнопку go\_to\_second (Рис. 13).



Рис. 13 Результат нажатия кнопки go\_to\_second

Нажмем кнопку back один раз, мы должны будем вернуться в Activity 1 (Рис.14).



Рис. 14 Результат одного нажатия кнопки back

Значит, Activity 3 не хранится более в backstack. Теперь вернемся в Activity 3 и нажмем кнопку go\_to\_first (Рис. 15).



Рис. 15 Результат нажатия кнопки go\_to\_first в Activity 3

Мы вернулись к Activity 1, для проверки backstack нажмем один раз кнопку back – как и ранее, мы должны будем выйти из приложения (Рис. 16).



Рис. 16. Результат нажатия кнопки back

Значит в backstack не хранилось более ни Activity 3, ни Activity 2, оба были завершены после нажатия кнопки go\_to\_first. Наконец, чтобы убедиться, что все Activity находятся всегда в одном порядке, зайдем в Activity 3 и будем постепенно нажимать кнопку back (Рис. 17).

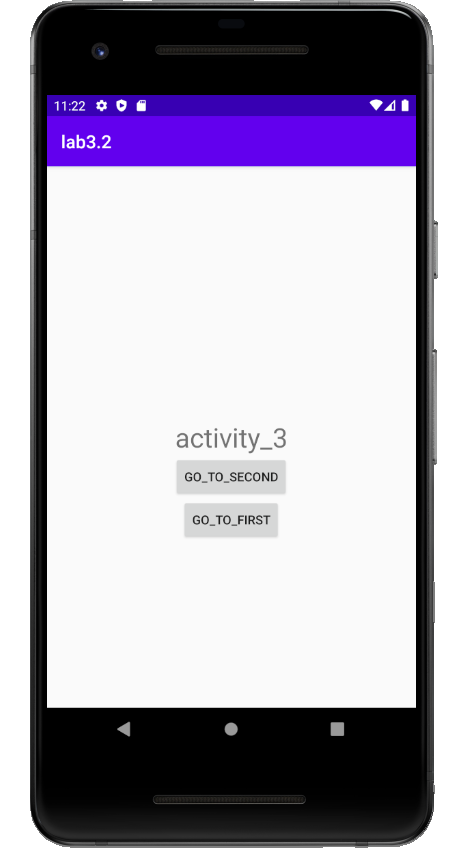


Рис. 17 Начало проверки порядка Activity

После первого нажатия возвращаемся в Activity 2 (Рис. 18).



Рис. 18 Продолжение проверки порядка

После еще одного нажатия, отображается Activity 1 (Рис. 19).



Рис. 19 Продолжение проверки

После последнего нажатия выходим из приложения (Рис. 20).



Рис. 20 Результат проверки порядка Activity

В результате было выяснено, что Activity хранятся в одном и том же порядке 1-2-3. Все Activity были зарегистрированы в файле AndroidManifest, если забыть зарегистрировать какую-либо из Activity, то приложение будет падать с ошибкой android.content.ActivityNotFoundException.

**ЗАДАЧА №3. НАВИГАЦИЯ (ФЛАГИ INTENT/АТРИБУТЫ ACTIVITY)**

Исходный код для данного задания представлен по ссылке:

<https://github.com/ADsty/labs-android_labs/tree/master/lab3.3>

Для данного задания был незначительно изменен исходный код Activity 2 – теперь вместо метода startActivityForResult на кнопке go\_to\_third, используется метод startActivity.

Листинг 7 – измененный код для кнопки go\_to\_third

button2.setOnClickListener**{** val intent = Intent(this, Activity3::class.java)  
 startActivity(intent)  
**}**

Также был изменен код Activity 3, в ней вместо использования метода setResult и finish на кнопке go\_to\_first используется метод startActivity с флагом Intent FLAG\_ACTIVITY\_CLEAR\_TOP.

Листинг 8 – измененный код для кнопки go\_to\_first

button1.setOnClickListener **{** val intent = Intent(this, Activity1::class.*java*).addFlags(Intent.*FLAG\_ACTIVITY\_CLEAR\_TOP*)  
 startActivity(intent)  
**}**

Если данный флаг установлен и activity, которая должна запускаться, уже работает в текущем task-е, то вместо того, чтобы запускать новую activity, все activity выше нужной нам завершат свою работу и нужное нам activity будет на вершине стека. Таким образом, при вызове Activity 1 из Activity 3, Activity 2 и 3 завершат свою работу и в стеке останется только Activity 1. Проверим работу данного флага (Рис. 21).



Рис. 21 Начало проверки флага Intent

Теперь нажмем на кнопку go\_to\_first (Рис. 22).



Рис. 22 Отображение Activity 1 на экране

Как видим, Activity 1 действительно отобразилось на экране. Для того, чтобы убедиться в правильности работы флага, нажмем кнопку back один раз, ожидается, что мы выйдем из приложения, так как в backstack находится только Activity 1 (Рис. 23).



Рис. 23 Результат нажатия кнопки back

Как мы видим, в backstack действительно хранился только экземпляр Activity 1. Поскольку никаких изменений кроме данного не было внесено в исходный код, то ожидается, что во всех остальных ситуациях программа будет вести себя идентично предыдущему заданию.

**ЗАДАЧА №4. НАВИГАЦИЯ (ФЛАГИ INTENT/АТРИБУТЫ ACTIVITY)**

Исходный код для данного задания представлен по ссылке:

<https://github.com/ADsty/labs-android_labs/tree/master/lab3.4>

В данной задаче необходимо дополнить граф навигации новым переходом для демонстрации какого-либо флага Intent, который не использовался ранее. Для данного задания в шторке Navigation Drawer создадим еще одну Activity для демонстрации флагов FLAG\_ACTIVITY\_CLEAR\_TASK и FLAG\_ACTIVITY\_NEW\_TASK.

Листинг 9 – изменения в Navigation Drawer

<item  
 android:id="@+id/clear"  
 android:icon="@drawable/ic\_menu\_camera"  
 android:title="@string/menu\_gallery" />

Листинг 10 – изменения в activity

else if (item.*itemId* == R.id.*clear*){  
 val intent = Intent(this, ActivityClear::class.*java*)  
 intent.addFlags(Intent.*FLAG\_ACTIVITY\_NEW\_TASK*)  
 intent.addFlags(Intent.*FLAG\_ACTIVITY\_CLEAR\_TASK*)  
 startActivity(intent)  
}

Первый флаг можно использовать только в связке со вторым, поскольку он очищает любой task, связанный с данным приложением до того, как activity будет запущена, все старые activity будут завершены. Таким образом, данные флаги можно использовать для очищения backstack, если при переходе в нужное activity ни одно из прошлых activity нам не понадобится. Проверим работу данного флага на следующем примере – зайдем в Activity 3, оттуда из шторки вызовем Activity Clear, после чего нажмем кнопку back – ожидается что приложение будет остановлено, так как вся история до Activity Clear будет удалена. Начнем с того, что зайдем в Activity 3 (Рис. 24).

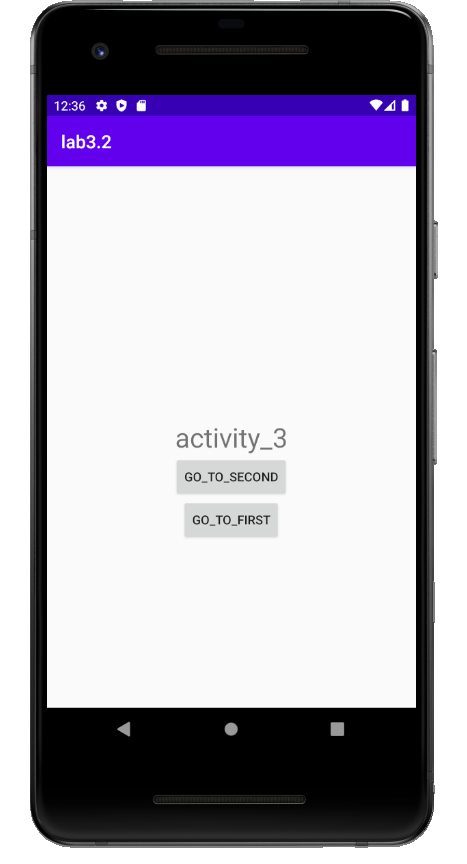


Рис. 24 Начало проверки работы флагов

Затем вызовем шторку Navigation Drawer (Рис. 25).



Рис. 25 Измененная шторка Navigation Drawer

Теперь нажмем на элемент Clear (Рис. 26).

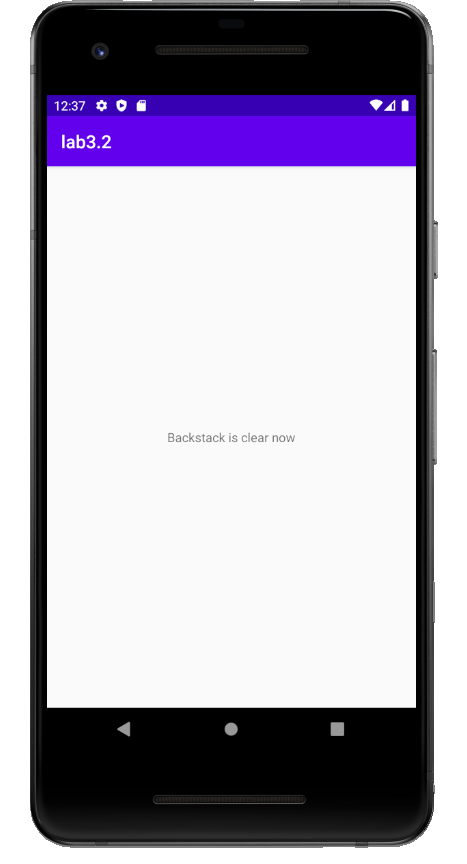


Рис. 26 Результат нажатия кнопки Clear

Как можно видеть, было вызвано Activity Clear. После нажатия кнопки back ожидается, что приложение будет остановлено (Рис. 27).



Рис. 27 Результат нажатия кнопки back

После повторного запуска приложение начинает свою работу с Activity 1 (Рис.28).



Рис. 28 Повторный запуск приложения

Для остальных activity ожидается аналогичная работа данных флагов.

**ЗАДАЧА №5. НАВИГАЦИЯ (FRAGMENTS, NAVIGATION GRAPH)**

Исходный код для данного задания можно найти по ссылке:

<https://github.com/ADsty/labs-android_labs/tree/master/lab3.5>

В данном задании нужно решить предыдущую задачу с помощью navigation graph. Все Activity должны быть заменены на фрагменты, но Activity About должна остаться самостоятельной Activity. Для этого сделаем директорию navigation в директории res, после чего создадим navigation graph и соединим наши фрагменты и Activity About переходами (Рис. 29).

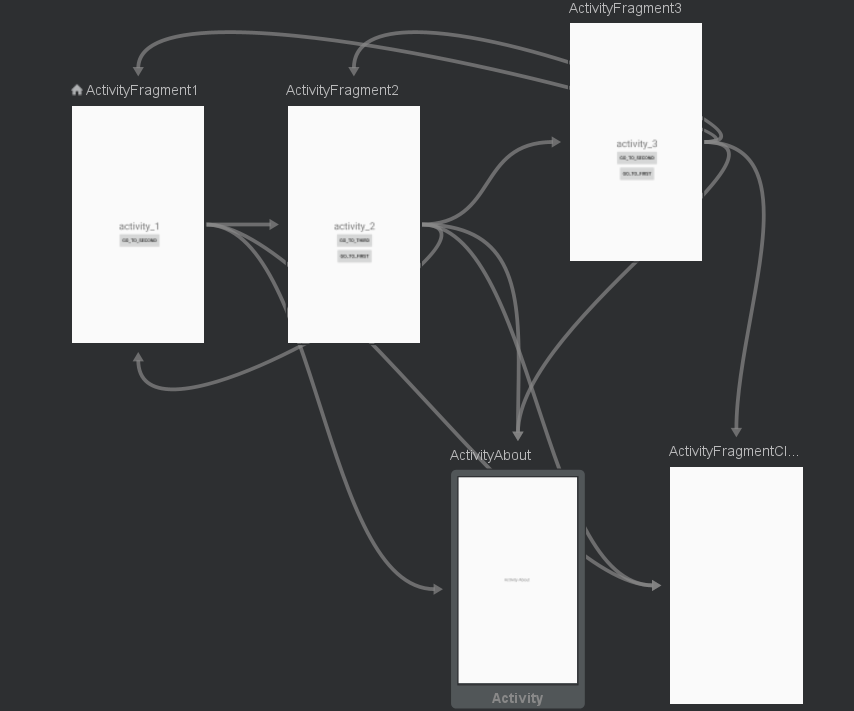


Рис. 29 Граф навигации между фрагментами и Activity About

Добавленное в прошлом задании Activity Clear также сделано с помощью фрагмента, но оставлено в шторке Navigation Drawer. Внутри фрагментов, в методе onCreateView мы устанавливаем содержимое данного фрагмента, в методе onViewCreated мы добавляем методы для обработки нажатия на кнопки внутри фрагмента, но теперь используется NavHost Fragment, который с помощью метода navigate меняет содержимое в соответствии с графом навигации.

Листинг 11 – исходный код первого фрагмента (для примера)

class Fragment1: Fragment() {  
  
 override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {  
 return inflater.inflate(R.layout.*fragment1*, container, false)  
 }  
  
 override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
 val button1: Button = view.findViewById(R.id.*from\_1\_to\_2*)  
 button1.setOnClickListener**{** NavHostFragment.findNavController(this).navigate(R.id.*from\_first\_to\_second*)  
 **}** }  
  
}

Внутри Activity main был несколько изменен принцип работы шторки Navigation Drawer. Теперь, при нажатии на кнопки about и clear просматривается, какой из фрагментов сейчас на экране (с помощью id фрагмента), и в соответствии с этим осуществляется переход по нужному пути в графе навигации.

Листинг 12 – измененный код для шторки Navigation Drawer

private val mOnNavigationItemSelectedListener = NavigationView.OnNavigationItemSelectedListener **{** item **->** val o = *supportFragmentManager*.*fragments*[0]  
 val i = NavHostFragment.findNavController(o).*currentDestination*?.*id* if(item.*itemId* == R.id.*about*) {  
 when (i) {  
 R.id.*ActivityFragment1* -> NavHostFragment.findNavController(o)  
 .navigate(R.id.*from\_first\_to\_about*)  
 R.id.*ActivityFragment2* -> NavHostFragment.findNavController(o)  
 .navigate(R.id.*from\_second\_to\_about*)  
 R.id.*ActivityFragment3* -> NavHostFragment.findNavController(o)  
 .navigate(R.id.*from\_third\_to\_about*)  
 }  
 }  
 else if (item.*itemId* == R.id.*clear*){  
 when (i) {  
 R.id.*ActivityFragment1* -> NavHostFragment.findNavController(o)  
 .navigate(R.id.*from\_first\_to\_clear*)  
 R.id.*ActivityFragment2* -> NavHostFragment.findNavController(o)  
 .navigate(R.id.*from\_second\_to\_clear*)  
 R.id.*ActivityFragment3* -> NavHostFragment.findNavController(o)  
 .navigate(R.id.*from\_third\_to\_clear*)  
 }  
 }  
 return@OnNavigationItemSelectedListener true  
**}**

Для начала зайдем в приложение и пройдем от первой activity, до третьей (Рис. 30).

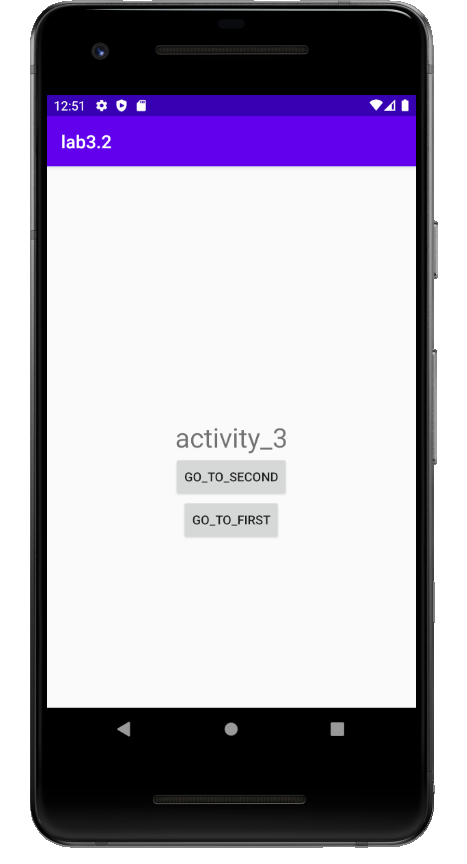


Рис. 30 Результат прохода до activity 3

Теперь нажмем на кнопку go\_to\_first (Рис. 31).



Рис. 31 Результат нажатия кнопки go\_to\_first

И теперь нажмем кнопку back, в данном случае мы не остановим приложение, а вернемся обратно в activity 3 (Рис.32).

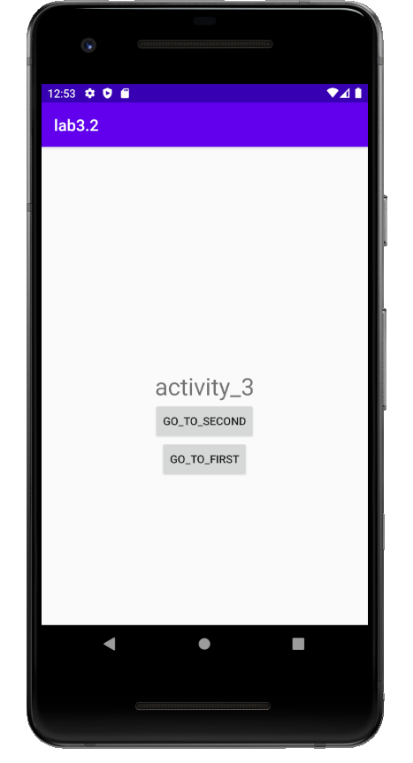


Рис. 32 Результат нажатия кнопки back

На экране меняется лишь содержимое фрагмента, без каких-либо видимых переходов, как в случае с activity. Теперь откроем шторку Navigation Drawer и нажмем кнопку clear, в данном случае это будет полностью пустым фрагментом без каких-либо переходов или текста (Рис. 33).



Рис. 33 Результат нажатия кнопки clear в меню Navigation Drawer

Теперь вернемся кнопкой back в Activity 3 и откроем Activity about (Рис. 34).



Рис. 34 Открытие Activity About

С характерной анимацией при нажатии кнопки открывается новое activity – Activity about. Как можно видеть, приложение работает корректно, за исключением ограничения на backstack, которое не учитывается в данном задании.

**ЗАКЛЮЧЕНИЕ**

В данной работе были изучены Lifecycle-aware компоненты, которые выполняют действия в ответ на изменение состояния жизненного цикла другого компонента (activity или фрагмента). Были изучены различные подходы к организации навигации внутри приложения – с помощью метода startActivityForResult получается наиболее легко читаемый код, логика приложения легче всего отслеживается, в то же время с помощью флагов Intent легче организовать работу программы в нестандартных ситуациях, где будет недостаточно одного метода finish, как в случае с Activity clear, которая очищает полностью текущий task приложения и создает новый. С помощью фрагментов получается код, который легче всего поддерживать, вместо того чтобы переписывать большие куски кода в activity, достаточно изменить код небольшого фрагмента или же изменить код общей для фрагментов activity, вместо изменения кода для каждой отдельной activity из первого варианта.

Исходный код для лабораторных работ находится на [github](https://github.com/ADsty/labs-android_labs) (<https://github.com/ADsty/labs-android_labs> )

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

<https://developer.android.com/topic/libraries/architecture/lifecycle>

<https://codelabs.developers.google.com/codelabs/android-lifecycles/index.html?index=..%2F..%2Findex#0>

<https://developer.android.com/reference/android/app/Activity.html#starting-activities-and-getting-results>

<https://developer.android.com/guide/components/activities/tasks-and-back-stack>

<https://developer.android.com/reference/android/content/Intent#FLAG_ACTIVITY_CLEAR_TOP>

<https://developer.android.com/reference/android/content/Intent#FLAG_ACTIVITY_CLEAR_TASK>

<https://developer.android.com/reference/android/content/Intent#FLAG_ACTIVITY_NEW_TASK>

<https://developer.android.com/guide/navigation/navigation-getting-started>

<https://developer.android.com/guide/fragments#UI>