Санкт-Петербургский политехнический университет Петра Великого

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**Отчёт по лабораторной работе №3**

**Дисциплина**: Проектирование мобильных приложений

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

Выполнил студент гр. 3530901/80202 Д. Ю. Ткаченко

(подпись)

Преподаватель А. О. Алексюк

(подпись)

“\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

Санкт-Петербург

2020

**1. Цели** Ознакомиться с методом обработки жизненного цикла activity/fragment при помощи Lifecycle-Aware компонентов

Изучить основные возможности навигации внутри приложения: создание новых activity, navigation graph

Вариант 17.

**2. Программа работы**

**2.1. Задача 1. Обработка жизненного цикла с помощью Lifecycle-Aware компонентов**

Ознакомьтесь с Lifecycle-Aware Components по документации и выполните codelabs.

## Step 2 - Add a ViewModel

На этом шаге используется ViewModel для сохранения состояния при поворотах экрана. Таймер сбрасывается, когда происходит поворот экрана, который в свою очередь уничтожает Activity. Поэтому мы используем ViewModel, так как он не уничтожается, если его владелец уничтожается из-за изменения конфигурации (поворот экрана). Новый экземпляр владельца повторно подключается к существующей ViewModel. Если запустить код, то можно убедиться в этом: при повороте экрана или при переходе в другое в приложение и возвращении обратно таймер не сбрасывается.

## Step 3 - Wrap Data Using LiveData

На этом этапе заменяется таймер, который использовался в предыдущих шагах, на собственный, который использует таймер и обновляет пользовательский интерфейс каждую секунду.

Теперь при запуске приложения в Logcat журнал обновляется каждую секунду, если мы не перейдём в другое приложение, при этом таймер всё также будет работать и продолжать считать количество секунд.

## Step 4 - Subscribe to Lifecycle Events

На этом шаге нужно обновить класс под названием BoundLocationManager, чтобы он учитывал жизненный цикл: он будет связываться, наблюдать и реагировать на изменения в LifecycleOwner. Чтобы класс мог наблюдать за жизненным циклом Activity, мы должны добавить его в качестве наблюдателя. Для этого даём объекту BoundLocationManager команду наблюдать за жизненным циклом.

Теперь при запуске приложения в Logcat мы можем видеть, что наблюдатель добавлен или удалён.

## Step 5 - Share a ViewModel between Fragments

На данном этапе нужно разрешить обмен данными между фрагментами, используя ViewModel. Нашей задачей является соединение фрагментов с ViewModel, чтобы при изменении одного SeekBar обновлялся другой SeekBar.

После проделанных действий при изменении одного из SeekBar, своё значение меняет и другой SeekBar.

## Step 6 - Persist ViewModel state across process recreation (beta)

На заключительном шаге нашей задачей является сохранить состояние приложения, чтобы информация не была потеряна в случае остановки процесса. При запуске исходного кода приложения мы можем сохранить информацию в ViewModel, и если мы завершим процесс принудительно, то при перезапуске приложения мы увидим, что значение в ViewModel не сохранилось, однако EditText восстановил своё состояние. Это связано с тем, что некоторые элементы пользовательского интерфейса, включая EditText, сохраняют своё состояние, используя собственную реализацию onSaveInstanceState. Это состояние восстанавливается после завершения процесса так же, как оно восстанавливается после изменения конфигурации.

Для сохранения информации в ViewModel нам нужно в файле SavedStateViewModel.java добавить новый конструктор.

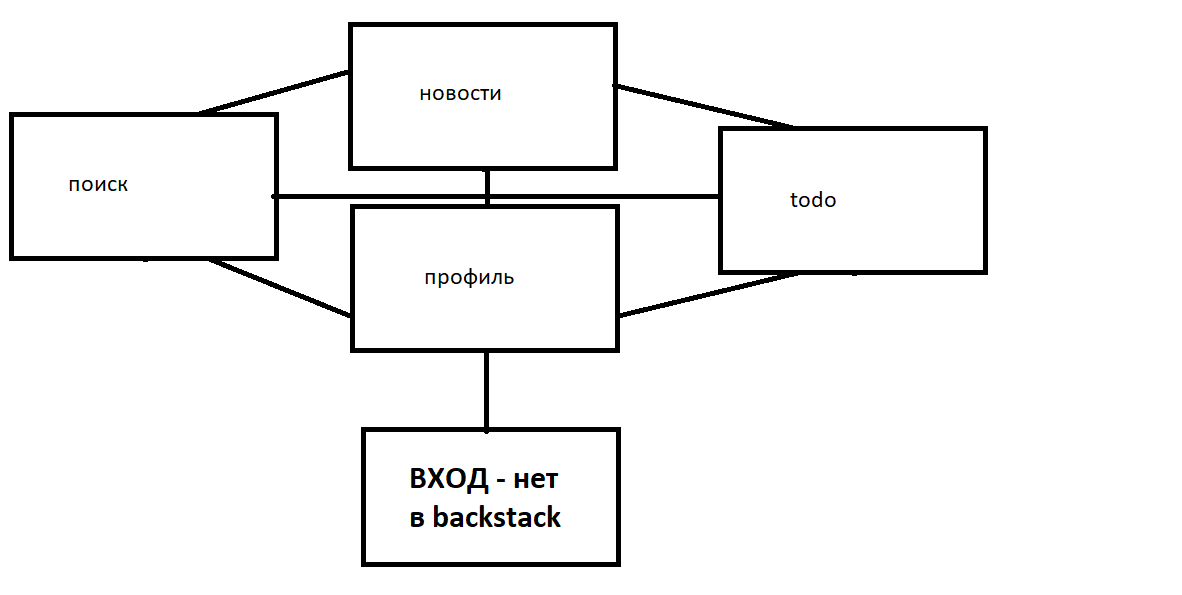
Теперь после проделанных шагов можно убедиться в том, что после повторного запуска приложения состояние в ViewModel сохранено.

После выполнения codelabs произведено знакомство со следующими компонентами жизненного цикла Activity:

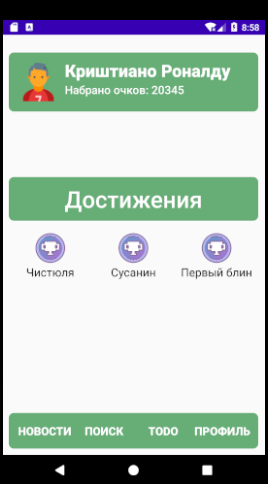
* ViewModel - предоставляет способ создавать и извлекать объекты, привязанные к определенному жизненному циклу.
* LifecycleOwner — это интерфейс, реализованный классами AppCompatActivity и Fragment. Мы можем подписать другие компоненты на объекты-владельцы, которые реализуют данный интерфейс, чтобы наблюдать за изменениями в жизненном цикле владельца.
* LiveData - позволяет наблюдать за изменениями данных в нескольких компонентах приложения, не создавая явных жёстких путей зависимости между ними. Он учитывает сложные жизненные циклы компонентов приложения, включая действия, фрагменты, службы или любой LifecycleOwner, определённый в нашем приложении. LiveData управляет подписками наблюдателей, приостанавливая подписки на остановленные объекты LifecycleOwner и отменяя подписки на завершённые объекты LifecycleOwner.

**2.2. Задача 2 + Задача 3 + Задача 4. Навигация**

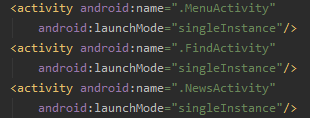
Задания 2, 3, 4 были объединены в одно для того, чтобы выбрать наиболее удобный способ реализации графа переходов для собственного приложения. Граф показан на рисунке ниже.



Испробовав startActivityForResult, было принято решение, что в данном графе его использование не имеет смысла, так как в каждой вкладке нет никаких вложенных вкладок. Более того, forResult очень плохо позволяет контролировать такой связанный граф, так как мы не знаем, на какой вкладке был пользователь до этого.



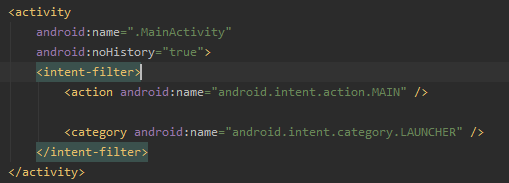
В манифесте созданы атрибуты, которые позволяют создавать Singleton Activity.

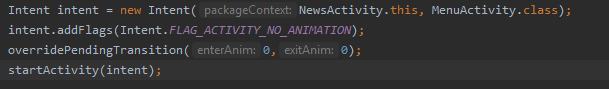


Благодаря этим атрибутам достигается ситуация, что каждая Activity может быть создана лишь один раз.

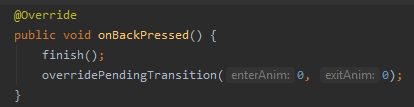
Этот режим очень похож на singleTask, где в системе мог существовать только один экземпляр Activity. Разница в том, что задача, которая располагает этим Activity, может иметь только одно Activity — то, у которого атрибут singleInstance. Если из этого вида Activity вызывается другое Activity, автоматически создается новое задание для размещения этого нового Activity. Аналогичным образом, если вызывается singleInstance Activity, будет создана новая задача для размещения этого Activity.

Для решения задачи «не сохранять экран входа в backstack» использован атрибут noHistory:



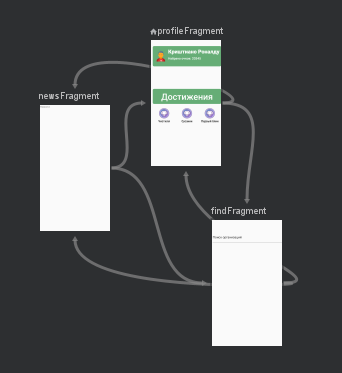
Для удаления анимации переходов между окнами был добавлен флаг в intent:

А при нажатии back вызывается такой код:



Теперь все работает, так как запланировано, 4 активити переключаются друг на друга без анимации, и мы легко нажатием back можем вернуться на предыдущий активити, при этом каждый активити может создаться лишь один раз.

**2.5. Задача 5. Навигация (Fragments, Navigation Graph)**



Решите предыдущую задачу (с расширенным графом) с использованием navigation graph.

Для решения данной задачи были созданы фрагменты, была создана Main activity, в которой находится NavController, который помогает переходить между activity. fragment является контейнером для фрагментов, и они переключаются внутри него.

Чтобы переключаться между фрагментами нужно заполнить nav\_graph, а также указать переходы между фрагментами.

Добавление фрагментов кардинально поменяло работу программы, теперь переключение между окнами происходит быстрее, код занимает меньше места и визуально программа выглядит куда более понятно.

**3. Выводы**

В ходе выполнения лабораторной работы были изучены: Lifecycle-Aware Components, навигация с помощью стандартных методов SDK и флагов Intent, навигация c использованием fragments + Navigation Component. Все эти способы навигации были продемонстрированы и протестированы на примере приложения. Для моего приложения наиболее удачным оказался вариант с использованием fragments.