Федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский политехнический университет Петра Великого»

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

**Отчёт по лабораторной работе №3**

**Дисциплина:** Проектирование мобильных приложений

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

**Вариант №14**

Выполнил студент гр. 3530901/70202

Д.В.Павлов

*(подпись)*

Преподаватель

А.Н.Кузнецов

*(подпись)*

Санкт-Петербург

2019

1. **Цели работы**

* Ознакомиться с методом обработки жизненного цикла activity/fragment при помощи Lifecycle-Aware компонентов;
* Изучить основные возможности навигации внутри приложения: создание новых *activity, navigation graph*.

**2. Выполнение работы**

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

***Шаг 1.***

Первоначально запущенный вариант приложения не предусматривает обработку жизненного цикла *activity*, вследствие чего при изменении конфигурации (поворот экрана) таймер сбрасывается и начинает отсчет с нуля.

***Шаг 2.***

Далее был запущен второй вариант приложения*.* В данном случае класс использует *ViewModel*, который сохраняется до тех пор, пока *LifecycleOwner* не *finished,* что отличается от состояния *destroyed*. Теперь смена конфигурации не уничтожает *ViewModel*, а новый *activity instance* подключается к существующему экземпляру *ViewModel*. Таймер сбрасывается лишь при полном закрытии приложения.

***Шаг 3.***

Теперь строковое значение таймера устанавливается в текстовое поле *TextvVIew* *ChronoActivity3* каждую секунду. Данное действие логируется на *debug* уровне.

*ChronoActivity3* обновляет *UI* каждый раз, когда получает соответствующее уведомление от таймера. Для избегания утечек памяти во *ViewModel* не следует указывать никаких отсылок к *activity*. Если *ViewModel* содержит такие отсылки, то они могут остаться к той версии *activity*, которая была до изменения конфигурации и должна быть собрана сборщиком мусора. Система будет хранить экземпляры *ViewModel*, пока соответствующее *activity* не будет терминировано.

Поэтому, вместо изменений полей прямо в *ViewModel,* *activity* конфигурируется так, что оно «наблюдает» за источником данных, получая эти данные при каждом их изменении. На примере нашего проекта в метод *subscribe* *ChronoActivity3* была добавлена строка, в которой данные о прошедшем времени получаются из экземпляра *LiveDataTimerViewModel,* после чего применяется метод *observe*, аргументами на вход которого будет нынешний экземпляр *LifecycleOwner (this)* и *observer*, устанавливающий новое время при его изменении.

Также в таймер было добавлено установление нового значение времени, которого отсутствовало ранее (*mElapsedTime.postValue(newValue)*).

После внесения указанных модификаций в *logcat* можно видеть, что каждую секунду обновляются записи – происходит обновление *UI*. Логирование не совершается, если перейти в другое приложение, свернуть приложение *Step3,* перевести экран в режим блокировки. Поворот экрана не оказывает никакого влияния на работу приложения. Из этого можно сделать вывод, что *LiveData* объекты обновляются только тогда, когда *activity* или *LifecycleOwner* активны.

***Шаг 4.***

На этом этапе речь пойдет о *Lifecycle Events*. Многие компоненты и библиотеки *Android* требуют того, чтобы они были инициализированы или остановлены в определенный момент жизни *activity*. Несоблюдение данного правила влечет за собой утечки памяти и баги.

Экземпляр *LifecycleOwner* может быть передан в новые экземпляры ­*lifecycle-aware* компонентов, чтобы сообщить им о нынешнем состоянии жизненного цикла.

Нынешнее состояние жизненного цикла можно запросить следующим способом: **lifecycleOwner.getLifecycle().getCurrentState().**

*­Lifecycle-aware* объект, который реализовывает *LifecycleObserver*, может также следить за изменениями состояния *LifecycleOwner*: **lifecycleOwnder.getLifecycle().addObserver(this).**

Можно также уведомить объект, во время какого состояния жизненного цикла выполнять те или иные действия (Рис.2.1.1).



Рис.2.1.1.

На этом шаге был создан компонент, реагирующий на смену состояний жизненного цикла *LifecycleOwner*. С помощью *LocationManager* мы получаем ширину и долготу, на которой находится устройство, и отображаем их на экране. При изменении локации пользователя происходит обновление *UI.* Когда *activity* приостановлено, слушатель не обрабатывает никакой информации и очищается. Если *activity* активно, будет происходить связь с системными сервисами, *GPS* модулем. Для этого были добавлены @OnLifecycleEvent аннотации ON\_RESUME и ON\_PAUSE к методам addLocationListener() и removeLocationListener() соответственно.

Для того, чтобы класс *BoundLocationManager* мог следить за жизненным циклом *activity*, в конструктор класса был добавлен *observer*.

Приложение было запущено на эмуляторе. При повороте экрана, его блокировке, а также при сворачивании приложения в логе показывается информации о том, что слушатель удален или добавлен, в зависимости от вызова методов onPause или onResume.

Результат работы представлен на Рис.2.1.2. Далее были изменены координаты расположения устройства. Как и следовало ожидать, при изменении расположения, *UI* обновляется и в *TextView* будут выведены новые значения (Рис.2.1.3-4).



Рис.2.1.2.



Рис.2.1.3.



Рис.2.1.4.

***Шаг 5.***

На данном этапе предлагается создать приложение, состоящие из одного *activity* и включающее в себя два фрагмента, в каждом из которых содержится по одному *SeekBar.* Также имеется *ViewModel* с полем *LiveData*, с помощью которого можно реализовать синхронное изменения значения на *SeekBar*.

В качестве *LifecycleOwner* используется *activity* (для его получения использован метод *getActivity()*), так как жизненный цикл каждого фрагмента является независимым. При изменение пользователем значения прогресса в *SeekBar*, его значение на данный момент устанавливается в *ViewModel* с помощью метода *setValue()*, а также производится запись в лог, говорящая о том, что прогресс изменился. Последний шагом во внесение изменений является обновление *SeekBar* с помощью метода *setProgress()*, когда появляются изменения в *ViewModel*.

После внесения данных поправок при изменении значения прогресса в одном из *SeekBar* оно меняется и во втором.

***Шаг 6.***

На данном этапе требуется сохранять состояние *ViewModel* при пересоздании процесса. Когда пользователи переключается между приложениями, *Android* хранит те из них, которые не находятся на данный момент на первом плане (например, не видимы сейчас для пользователя).

Когда пользователь впервые запускает приложение, для этого создается процесс. Но когда пользователь покидает это приложение, процесс не заканчивается. Система хранит его в кэше. Если пользователь позже возвращается к приложению, систему снова начинает использовать сохранившийся процесс, загружая, таким образом, приложение быстрее.

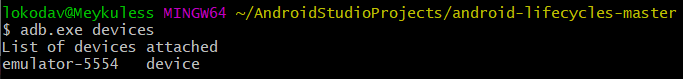
Если слишком много памяти занято, система убивает процессы, которые были использованы наиболее давно, и, если пользователь снова переключается в приложение, система запустит его в новом процессе.

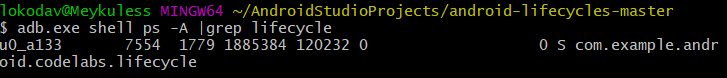
Поскольку этот сценарий имеет место лишь в том случае, когда пользователь не взаимодействовал с приложением в течение долгого времени, то тот факт, что он столкнется со стартовым экраном, является в некоторых случаях допустимым. Однако, в некоторых случаях, необходимо сохранять состояние всего приложения или какой-то его части, чтобы информации не была потеряна при уничтожении процесса.

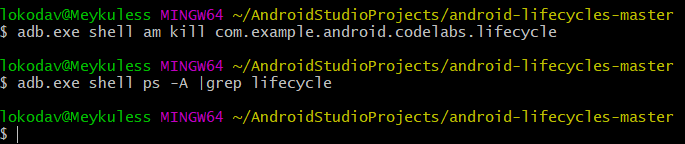
Модуль *lifecycle-viewmodel-savedstate* позволяет сохранить состояние в *ViewModels*. В проект требовалось ввести два изменения, чтобы можно было реализовать доступ к сохраненным данным:

1. Добавить конструктор в *ViewModel*, принимающий *SavedStateHandle*.
2. Передавать *SavedStateVMFactory* (делает *ViewModel* с *SavedStateHandles*)в *ViewModelProvider*;

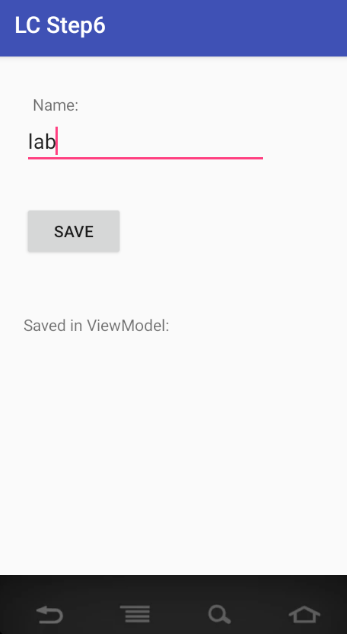
В начале было запущено приложение без каких-либо изменений и проведен эксперимент (смена имени и последующее его сохранение). Далее мы вышли на стартовый экран и программно убили наш процесс (проверив его наличие до убийства и после). Для таких действия потребовался эмулятор с API Pie и выше.





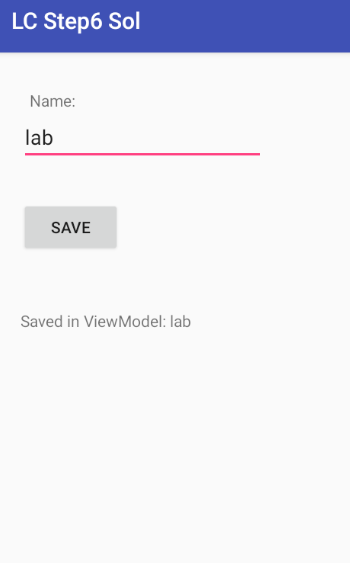


После приложение было открыто снова. Имя осталось в EditText, но во ViewModel оно не сохранилось. Некоторые элементы пользовательского интерфейса, включая EditText, сохраняют свое состояние, используя собственную реализацию onSaveInstanceState. Это состояние восстанавливается после того, как процесс завершается так же, как восстанавливается после изменения конфигурации.



В файл SavedStateViewModel.java был добавлен новый конструктор, который принимает SavedStateHandle и сохранить состояние в закрытом поле. Теперь будет использовать поддержка модуля LiveData, чтобы нам больше не нужно было хранить и отображать LiveData в ViewModel (соответствующим образом были изменены методы get и saveNewName.

После данных манипуляций приложение стало работать корректно (имя сохранилось во ViewModel).



**2.2. Навигация (*startActivityForResult*)**

Необходимо реализовать навигацию между экранами одного приложения согласно Рис.2.2.1 с помощью *Activity*, *Intent* и метода *startActivityForResult*. Приложение при этом начинает работу с отображения *Activity 1*. Кнопки *to first, to second* и *to third* отображают переходы к соответствующим *Activity*.

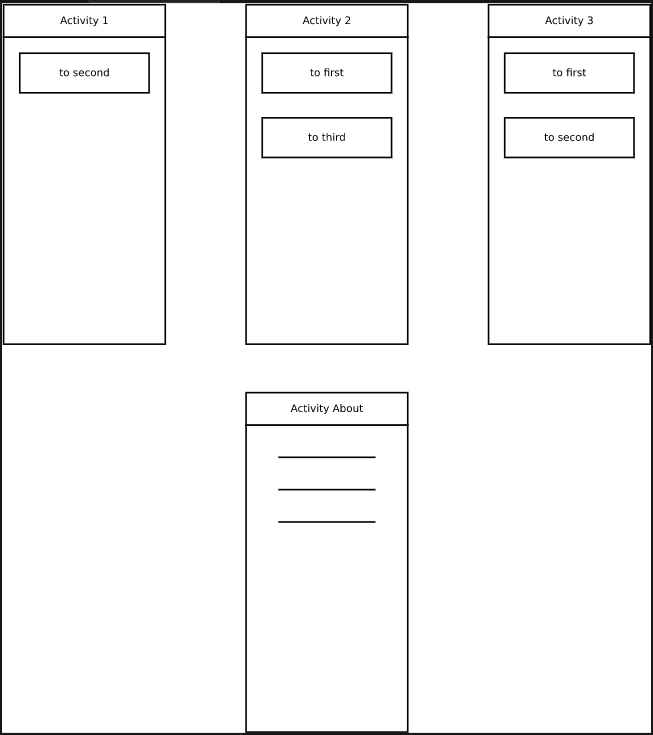


Рис.2.2.1.

Навигацию необходимо реализовать так, чтобы в любой момент времени в *backstack* находилось не более 4-х *Activity* (каждое *Activity* должен быть не более чем в одном экземпляре). *Activity ‘About’* должно быть доступно из любого другого *Activity*.

В начале выполнения данного пункта были созданы все 4 *Activity*, каждое из которых было зарегистрировано в *AndroidManifest*. Так приложение узнает о существовании *Activity*, и над ними можно выполнять различные действия. Соответственно, если их не указать в манифесте, работа приложения будет нарушена.

Для поддержки навигации с использованием *Bottom* *Navigation* в приложении, в *gradle* файл был добавлен следующий модуль (Листинг 2.2.1):

Листинг 2.2.1.

implementation 'com.google.android.material:material:1.1.0-alpha10'

Для *Bottom* *Navigation* было создано меню, содержащее всего один *item* (т. к. будет осуществляться навигация только к одному *Activity* – *About*)(Листинг 2.2.2).

Листинг 2.2.2.

<item  
 android:id="@+id/about\_item"  
 android:title="@string/about" />

Вид *xml* файла с применением *Bottom* *Navigation View* можно видеть в Листинге 2.2.3. *xml файлы остальных Activity,* из которых будет осуществляться навигация в *About*, выполнены по аналогии.

Листинг 2.2.3.

<?xml version="1.0" encoding="utf-8"?>  
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context=".Second">  
  
  
 <Button  
 android:id="@+id/firstFromSecond"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginTop="10dp"  
 android:text="@string/toFirst"  
 app:layout\_constraintLeft\_toLeftOf="parent"  
 app:layout\_constraintRight\_toRightOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent" />  
  
 <Button  
 android:id="@+id/thirdFromSecond"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginTop="10dp"  
 android:text="@string/toThird"  
 app:layout\_constraintLeft\_toLeftOf="parent"  
 app:layout\_constraintRight\_toRightOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@id/firstFromSecond" />  
  
  
 <com.google.android.material.bottomnavigation.BottomNavigationView  
 android:id="@+id/navigation"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:background="?android:attr/windowBackground"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintLeft\_toLeftOf="parent"  
 app:layout\_constraintRight\_toRightOf="parent"  
 app:menu="@menu/menu" />  
  
  
</androidx.constraintlayout.widget.ConstraintLayout>

Был задан слушатель *BottomNavigationVIew*, с той целью, чтобы предпринимать действия при нажатии пользователя на элемент меню (в данном случае, осуществление перехода в *Activity About*).

Переход в остальные *Activity* будет осуществляться с помощью кнопок. Поскольку нельзя сохранять лишь один *instance* of *Activity* в *backstack* с помощью флагов *Activity* или их атрибутов, для каждого из трех *Activity* будет использоваться функция, проверяющая, существует ли уже экземпляр этого *Activity* или нет. Если результат положительный, данная функция прекратит его работу, иначе можно производить переход к этому *Activity* сразу, так как в *backstack* его еще нет.

Сам переход осуществляется с помощью *startActivityForResult*. Аргументами будет являться *intent* между текущим *Activity* и *Activity*, к которому необходимо перейти, и целочисленный код запроса (*REQUEST\_CODE*), используемый для того, чтобы отличать друг от друга пришедшие результаты. А определить, успешно ли прошел вызов *Activity* или нет, помогает *resultCode* (*RESULT\_OK-успешный вызов*)*.*

Второе *Activity* представлено в Листинге 2.2.4.

Листинг 2.2.4.

import android.app.Activity;  
import android.content.Intent;  
import android.os.Bundle;  
import android.view.MenuItem;  
import android.view.View;  
import android.widget.Button;  
  
import androidx.annotation.NonNull;  
import androidx.appcompat.app.AppCompatActivity;  
  
import com.google.android.material.bottomnavigation.BottomNavigationView;  
  
public class Second extends AppCompatActivity{  
 private final int *THIRD\_ACTIVITY\_REQUEST\_CODE* = 3;  
 private final int *ABOUT\_ACTIVITY\_REQUEST\_CODE* = 0;  
 Second *instance* = null;  
  
 void finishWhenExist() {  
 if (*instance* != null)  
 *instance*.finish();  
 }  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*second*);  
  
 *instance* = this;  
 Intent intent = new Intent();  
 setResult(Activity.*RESULT\_OK*, intent);  
  
 Button butToFirst = findViewById(R.id.*firstFromSecond*);  
 butToFirst.setOnClickListener(  
 new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 finish();  
 }  
 }  
 );  
 Button butToThird = findViewById(R.id.*thirdFromSecond*);  
 butToThird.setOnClickListener(  
 new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 Third.*finishWhenExist*();  
 startActivityForResult(new Intent(Second.this,  
 Third.class),*THIRD\_ACTIVITY\_REQUEST\_CODE*);  
 }  
 }  
 );  
  
 BottomNavigationView navigation = findViewById(R.id.*navigation*);  
 navigation.setOnNavigationItemSelectedListener(new BottomNavigationView.OnNavigationItemSelectedListener() {  
 @Override  
 public boolean onNavigationItemSelected(@NonNull MenuItem item) {  
 Intent toMenu = new Intent(Second.this,About.class);  
 startActivity(toMenu);  
 return false;  
 }  
 });  
 }

@Override  
protected void onActivityResult(int requestCode, int resultCode, Intent data) {  
 super.onActivityResult(requestCode, resultCode, data)  
 if (requestCode == THIRD\_ACTIVITY\_REQUEST\_CODE)  
 if (resultCode == 33)  
 finish();  
 }

}

Несколько частей кода, иллюстрирующих навигацию приложения из 3 Activity:

1)Для перехода в 1 Activity.

Используем setResult(), чтобы известить 2 Activity о своих намерениях перейти в первую. В представленном выше коде для этого мы изменили onActivityResult добавив соответствующие проверки условий.

setResult(SECOND\_ACTIVITY\_FINISH\_CODE) //равен 33  
 finish();

2)Для перехода во 2 Activity достаточно использовать finish().

В первом Activity для перехода во вторую используется startActivity.

Работа приложения была проверена. Действительно, из каждого из трех *Activity* можно попасть в *Activity About* с помощью нижнего меню. Возврат осуществляется с помощью кнопки *back*, при нажатии которой *About* завершает работу благодаря *finish()* (Листинг 2.2.5).

Листинг 2.2.5.

Button back = findViewById(R.id.*back*);  
back.setOnClickListener(  
 new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 About.this.finish();  
 }  
 }  
);

Также, сколько бы много переходов между тремя *Activity* ни совершалось, в *backstack* никогда не было открыто несколько их экземпляров (что проверяется нажатием «назад»).

**2.3. Навигация (флаги *Intent* / атрибуты *Activity*)**

Отличие от предыдущего пункта - хранение экземпляров *Activity* в единственном числе осуществляется с помощью флагов *Intent* либо атрибутов *Activity*.

Вид *Activity* в данном случае будет выглядеть следующим образом (Листинг 2.3.1):

Листинг 2.3.1.

import android.content.Intent;  
import android.os.Bundle;  
import android.view.MenuItem;  
import android.view.View;  
import android.widget.Button;  
  
import androidx.annotation.NonNull;  
import androidx.appcompat.app.AppCompatActivity;  
  
import com.google.android.material.bottomnavigation.BottomNavigationView;  
  
public class Second extends AppCompatActivity{  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*second*);  
 Button butToFirst = findViewById(R.id.*firstFromSecond*);  
 butToFirst.setOnClickListener(  
 new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 Intent toFirst;  
 toFirst = new Intent(Second.this, First.class).  
 setFlags(Intent.*FLAG\_ACTIVITY\_REORDER\_TO\_FRONT*);  
 startActivity(toFirst);  
 }  
 }  
 );  
 Button butToThird = findViewById(R.id.*thirdFromSecond*);  
 butToThird.setOnClickListener(  
 new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 Intent toThird;  
 toThird = new Intent(Second.this, Third.class).  
 setFlags(Intent.*FLAG\_ACTIVITY\_REORDER\_TO\_FRONT*);  
 startActivity(toThird);  
 }  
 }  
 );  
  
 BottomNavigationView navigation = findViewById(R.id.*navigation*);  
 navigation.setOnNavigationItemSelectedListener(new BottomNavigationView.OnNavigationItemSelectedListener() {  
 @Override  
 public boolean onNavigationItemSelected(@NonNull MenuItem item) {  
 Intent toMenu = new Intent(Second.this,About.class);  
 startActivity(toMenu);  
 return false;  
 }  
 });  
 }  
}

Здесь *Intent* имеет флаг *FLAG\_ACTIVITY\_REORDER\_TO\_FRONT*. Если *Activity* уже есть в *backstack*, то оно перейдет наверх стека при вызове, в противном случае будет создан новый экземпляр.

Такого же эффекта можно было бы добиться с помощью атрибута *Activity android:launchMode= "singleTask "*. Как только *Activity* будетперезапущено, вызовется метод *onResume()*.

**2.4.** **Навигация (флаги *Intent* / атрибуты *Activity*)**

На данном этапе предлагается дополнить навигацию новыми переходами с целью демонстрации какого-нибудь атрибута *Activity* или флага *Intent*, который еще не использовался для решения задачи.

Для *Activity Second* была создана новая кнопка (Рис.2.4.1).

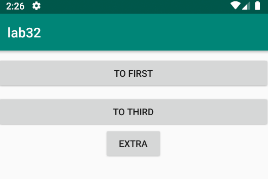


Рис.2.4.1.

При нажатии на эту кнопку запускается новое *Activity*, а у *Intent* используется флаг *FLAG\_ACTIVITY\_SINGLE\_TOP*. При использовании данного флага, если

*Activity* запускает сама себя, т.е. она находится в вершине стека, то вместо создания нового экземпляра в стеке вызывается метод onNewIntent().

Соответствующий код можно видеть в Листинге 2.4.1.

Листинг 2.4.1.

Button extraToFirst = findViewById(R.id.*exFirstFromSecond*);  
extraToFirst.setOnClickListener(  
 new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 Intent toFirst;  
 toFirst = new Intent(Second.this, Second.class).  
 setFlags(Intent.*FLAG\_ACTIVITY\_SINGLE\_TOP*);  
 startActivity(toFirst);  
 }  
 }  
);

Далее был проверен атрибут *Activity launchMode* со значением *SingleInstance*. *Activity* с установленным атрибутом не позволяет другим *Activity* быть частью её *Task*. Это единственная *Activity* в *Task*. Если она запускает другое *Activity*, то эта *Activity* назначается другому *Task*, как если бы FLAG\_ACTIVITY\_NEW\_TASK был в *Intent.*

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

Теперь необходимо решить предыдущую задачу с использованием *navigation graph.* Все *Activity* должны быть заменены на фрагменты, кроме *Activity* *'About'*, которая должна остаться самостоятельной.

Пример фрагмента можно видеть в Листинге 2.5.1.

Листинг 2.5.1.

import android.os.Bundle  
import android.view.LayoutInflater  
import android.view.View  
import android.view.ViewGroup  
import androidx.fragment.app.Fragment  
import androidx.navigation.fragment.findNavController  
import kotlinx.android.synthetic.main.fragment\_second.\*  
  
class SecondFragment : Fragment() {  
  
 override fun onCreateView(  
 inflater: LayoutInflater, container: ViewGroup?,  
 savedInstanceState: Bundle?  
 ): View? =  
 inflater.inflate(R.layout.*fragment\_second*, container, false)  
  
 override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
 super.onViewCreated(view, savedInstanceState)  
  
 val navController = *findNavController*()  
 button1.setOnClickListener **{** navController.popBackStack()  
 **}** button2.setOnClickListener **{** navController.navigate(R.id.*action\_secondFragment\_to\_thirdFragment*)  
 **}** }  
}

На кнопку установлен слушатель – при ее нажатии с помощью *Navigation controller (NavController)* производится переход, который содержится в ресурсе *navigation* – (либо со второго фрагмента на третий, либо на первый).

Все переходы меду фрагментами устанавливаются с помощью *Navigation Graph*. При этом все взаимосвязи наглядно показаны на Рис.2.5.1.

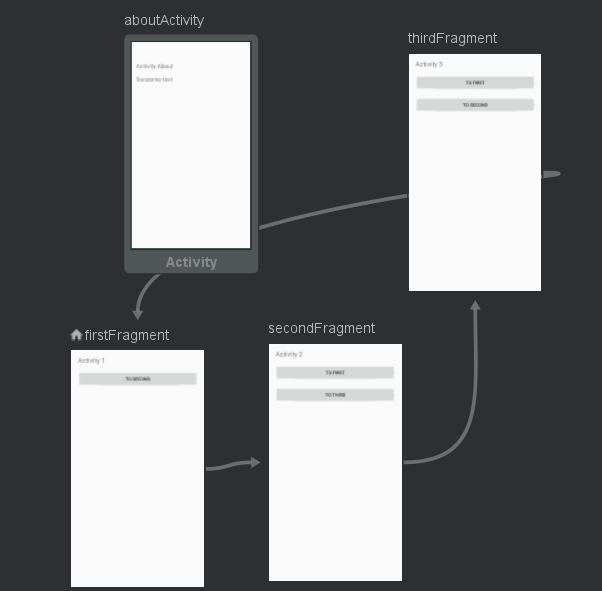


Рис.2.5.1.

*Xml* файл *navigation* представлен в Листинге 2.5.2.

Листинг 2.5.2.

<?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/mobile\_navigation"  
 app:startDestination="@id/firstFragment">  
 <fragment  
 android:id="@+id/firstFragment"  
 android:name="com.dao.lab3\_5.FirstFragment"  
 android:label="FirstFragment"  
 tools:layout="@layout/fragment\_first">  
 <action  
 android:id="@+id/action\_firstFragment\_to\_secondFragment"  
 app:destination="@id/secondFragment" />  
 </fragment>  
 <fragment  
 android:id="@+id/secondFragment"  
 android:name="com.dao.lab3\_5.SecondFragment"  
 android:label="SecondFragment"  
 tools:layout="@layout/fragment\_second">  
 <action  
 android:id="@+id/action\_secondFragment\_to\_thirdFragment"  
 app:destination="@id/thirdFragment" />  
 </fragment>  
 <fragment  
 android:id="@+id/thirdFragment"  
 android:name="com.dao.lab3\_5.ThirdFragment"  
 android:label="ThirdFragment"  
 tools:layout="@layout/fragment\_third">  
 <action  
 android:id="@+id/action\_thirdFragment\_to\_firstFragment2"  
 app:destination="@id/firstFragment"  
 app:popUpTo="@+id/firstFragment"  
 app:popUpToInclusive="true" />  
 </fragment>  
 <activity  
 android:id="@+id/aboutActivity"  
 android:name="com.dao.lab3\_5.AboutActivity"  
 android:label="AboutActivity"  
 tools:layout="@layout/activity\_about"/>  
</navigation>

У каждого *action* есть идентификатор, по которому переход упоминается во фрагментах (было показано в Листинге 2.5.1) и *destination* (куда ведет переход). Появляются атрибуты *popUpTo* (все фрагменты, находящиеся в *backstack* до запуска указанного фрагмента, закрываются) и *popUpToInclusive* (если *true*, то фрагмент, являющийся назначением атрибута *popUpTo*, тоже будет закрыт, и, перед только что открытым фрагментом, в *backstack* будет содержаться тот фрагмент, который был указан в назначении *popUpTo*).

Данные атрибуты были указаны, поскольку в *backstack* никогда не должно быть повторяющихся фрагментов и должна иметь место удобная навигация приложению: если пользователь находится в первом (стартовом) фрагменте, то переход по кнопке «назад» приводит к закрытию приложения; если пользователь находится во втором фрагменте, нажатие кнопки «назад» приводит к возвращению в первый (стартовый) фрагмент; если пользователь находится в третьем фрагменте, нажатие кнопки «назад» приводит к переходу во второй фрагмент.

*Activity About* может быть открыто через *Bottom Navigation*, как и в предыдущих пунктах. Реализация данного перехода, однако, также будет отличаться. *Activity*, как и фрагменты,был добавлен в *Navigation Graph*. Как и ранее, было создано меню, состоящее из одного *item* (Листинг 2.5.3).

Листинг 2.5.3.

<?xml version="1.0" encoding="utf-8"?>  
<menu xmlns:android="http://schemas.android.com/apk/res/android">  
 <item  
 android:id="@+id/navigation\_about"  
 android:icon="@drawable/ic\_notifications\_black\_24dp"  
 android:title="@string/activity\_about" />  
</menu>

Идентификатор *item* совпадает с идентификатором перехода в *Activity About* (Листинг 2.5.2).

Файлы *Activity About* представлены вЛистингах 2.5.4-5.

Листинг 2.5.4.

<?xml version="1.0" encoding="utf-8"?>  
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 android:id="@+id/container"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:paddingTop="?attr/actionBarSize">  
  
 <TextView  
 android:id="@+id/textView1"  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content"  
 android:text="@string/activity\_about"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent"  
 android:textSize="@dimen/standard\_text\_size"  
 android:layout\_margin="@dimen/standard\_margin" />  
  
 <TextView  
 android:id="@+id/textView2"  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content"  
 android:layout\_margin="@dimen/standard\_margin"  
 android:text="@string/simple\_text"  
 android:textSize="@dimen/standard\_text\_size"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@+id/textView1" />  
  
</androidx.constraintlayout.widget.ConstraintLayout>

Листинг 2.5.5.

import android.os.Bundle  
import androidx.appcompat.app.AppCompatActivity  
  
class AboutActivity : AppCompatActivity() {  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_about*)  
 }  
}

**3. Выводы**

В ходе выполнения лабораторной работы были достигнуты все поставленные цели:

- Получены навыки обработки жизненного цикла *Activity* с помощью *ViewModel* (существующий, все время, пока существует экземпляр *LifecycleOwner* и позволяющий сохранить состояние *Activity* при смене конфигурации)*, LifecycleOwner* (экземпляр которого позволяет нам работать с настоящей версией *Activity*)*, savedstate* (позволяющий хранить состояние долгое время, в том числе при *force stop* процесса);

- Приобретены умения по навигации в приложении между *Activities* или *Fragments* различными методами: переход к новому *Activity* с помощью *startActivityForResult* (при этом закрывать существующие экземпляры нужно альтернативными способами), *BottomNavigation* (представляющее собой нижнюю панель. Может быть доступен из разных участков приложения), *startActivity* (при этом для повышения функционала используются атрибуты *Activity* или флаги *Intent,* что, пожалуй, является самым простым способом при работе над навигацией приложения), а также с помощью *Navigation Graph* (при этом работа велась в основном с фрагментами, а не *Activity,* данный способ выделяется удобным графическим представлением переходов в приложении).