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»
Вариант 18

выполнил:
Пименов С. В.
группа: 3530901/90201
преподаватель:
Кузнецов А. Н.

Санкт-Петербург
2021

Оглавление

Цели

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

Задачи

- Использование Jetpack Compose для верстки layouts
  Познакомьтесь с содержимым курса Jetpack Compose и выполните codelab "Jetpack Compose basics"
- Навигация (startActivityForResult)
  Реализуйте навигацию между экранами одного приложения согласно изображению ниже с помощью Activity, Intent и метода startActivityForResult.
- Навигация (флаги Intent/атрибуты Activity)
  Решите предыдущую задачу с помощью Activity, Intent и флагов Intent либо атрибутов Activity.
- Навигация (флаги Intent/атрибуты Activity)
  Дополните 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.
- Навигация (Fragments, Navigation Graph)
  Решите 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.

Jetpack Compose

Мной был изучен инструмент Google Codelabs для быстрого изучения новых инструментов и фреймворков. С помощью этого инструмента я ознакомился с Jetpack Compose – набором инструментов для создания нативного пользовательского интерфейса для Android.

При использовании этого инструмента количество строк кода заметно сокращается, а весь процесс написания программы состоит из написания compose (составных) функций. Внутри самих функций вызываются необходимые методы и описываются их параметры. Сам Jetpack Compose позволяет делать множество различных вещей, а главное – делать их удобно.

Мной были изучены:

- Базовые компоненты Compose
- Реализация состояний UI (mutableState)
- Базовые контейнеры (Column, Row)
- Анимация (animateColorAsState)
- lazyRow, lazyColumn

В целом могу сказать, что изучать Jetpack Compose было интересно, а писать что-то на нем – поразительно просто.

Решение задачи при помощи метода startActivityForResult

Небходимо реализовать навигацию между экранами одного приложения согласно изображению ниже с помощью Activity, Intent и метода startActivityForResult. Также Activity About должна быть доступна из любой другой Activity через Options Menu (вариант 18).

Рис. 1 Задание

Рассмотрим предложенные нам методы:

• startActivity(Intent) – создаёт новую Activity, которая будет положена наверх стека. Имеет один аргумент Intent, который описывает в какую Activity будет совершён переход.

Intent – описание операции, которая должна быть исполнена.

• startActivityForResult(Intent, int) – создаёт новую Activity с ожиданием возврата результата по заданному коду.

• onActivityResult(int, int, Intent) –обрабатывает результат работы Activity.

• setResult(int) – возвращает данные родителю.

• finish – завершение работы Activity.

По заданию необходимо, чтобы в backstack не было разных сущностей одной Activity. Составим таблицу переходов, чтобы выполнить это условие и ничего не пропустить.

From, Activity	To, Activity	Backstack before	Backstack after	Transition
1	2	1	12	1 -> 2
2	3	12	123	2 -> 3
2	1	12	1	Close 2
3	1	123	1	Close 3, it closes 2
3	2	123	12	Close 3

Ход решения:
Для начала объявим в Manifest наши activity, иначе будем получать ActivityNotFoundExeption при попытке перейти к любому из них.

```
<activity
    android:name=".SecondActivity"
    android:label="Second"
    android:parentActivityName=".MainActivity"
    />
<activity
    android:name=".ThirdActivity"
    android:label="Third"
    android:parentActivityName=".SecondActivity"
    />
<activity
    android:name=".AboutActivity"
    android:label="About"
    android:parentActivityName=".MainActivity"
    />
<activity
    android:name=".MainActivity"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

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

Рис. 2 AndroidManifest.xml

Для начала были созданы все активити и их .xml файлы. Также была создана активити под названием “Optioned Activity”. Это было необходимо, чтобы был возможен переход к этой активити из всех прочих. Для того, чтобы не писать одинаковый код во всех прочих классах активити, было сделано так, что они все наследуют класс “Optioned Activity”, который, в свою очередь наследует “AppCompatActivity”.

Листинг 1. Optioned Activity

```
abstract class OptionedActivity: AppCompatActivity() {
    override fun onCreateOptionsMenu(menu: Menu?): Boolean {
        val inflater: MenuInflater = menuInflater
        inflater.inflate(R.menu.option_menu, menu)
        return true
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        return if (item.itemId == R.id.about_item) {
            startActivity(Intent(this, AboutActivity::class.java))
            true
        } else {
            super.onOptionsItemSelected(item)
        }
    }
}
```

Займемся переходами между экранами. В первой активити была написана функция toSecond(), которая осуществляет сам переход.

Листинг 2 MainActivity

```
class MainActivity : OptionedActivity() {
    private lateinit var binding: ActivityMainBinding
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)
        binding.secondButton.setOnClickListener { toSecond() }
    }

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

В .xml-файле этой активити есть кнопка, по нажатию на которую будет осуществляться переход.
Далее рассмотрим SecondActivity.

Листинг 3. SecondActivity

```
class SecondActivity : OptionedActivity() {

    private lateinit var binding: SecondActivityBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = SecondActivityBinding.inflate(layoutInflater)
        setContentView(binding.root)
        binding.firstButton.setOnClickListener { toFirst() }
        binding.thirdButton.setOnClickListener { toThird() }
```

```
    }

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


    private fun toFirst() {
        finish()
    }

    private fun toThird() {
        startActivityForResult(Intent(this, ThirdActivity::class.java),
RESULT_CODE)

    }

    companion object {
        const val RESULT_CODE = 0
    }

}
```

Для перехода из второго к первому состоянию мы вызываем finish() (см. обоснование в таблице переходов), для перехода к третьему – используем startActivityForResult(), т.к. из 3 нужно будет переходить в 1, за этим должно последовать разрушение второй активити. Если результатом завершения третьей активити станет RESULT_OK, то мы завершим данную активити.

Рассмотрим ThirdActivity.

Листинг 4. ThirdActivity

```
class ThirdActivity : OptionedActivity() {

    private lateinit var binding : ThirdActivityBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ThirdActivityBinding.inflate(layoutInflater)
        setContentView(binding.root)
        binding.firstButton.setOnClickListener { toFirst() }
        binding.secondButton.setOnClickListener { toSecond() }
    }

    private fun toFirst() {
        this.setResult(RESULT_OK)
        finish()
    }

    private fun toSecond() {
        finish()
    }

}
```

Если переходим к 1 состоянию, то проходим через 3 и завершаем его. Если к 1 – то просто завершаем.
Посмотрим на Back Stack в ходе работы нашего приложения.

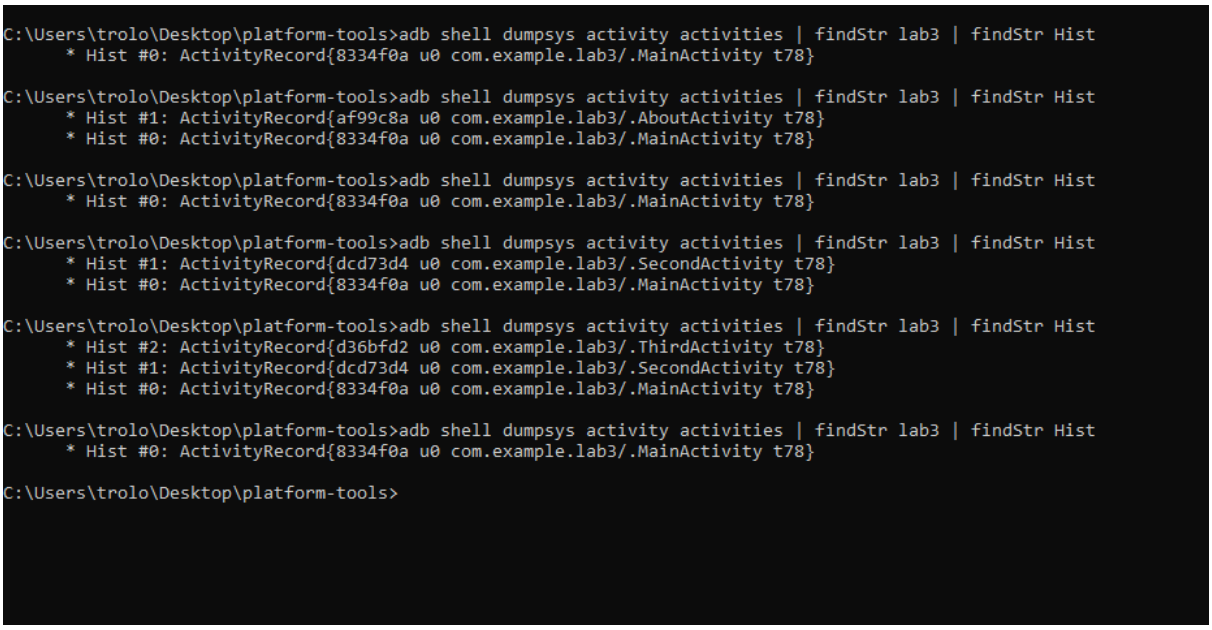

Рис 3. Проверка

В ходе проверки выяснилось, что разных сущностей одной активити не наблюдается.

Навигация (флаги Intent/атрибуты Activity)

Решим данную задачу при помощи startActivity и дополнительных флагов для Intent.

Листинг 5. Изменения в коде

SecondActivity

```
private fun toThird() {
    //with startActivityForResult() method
    //startActivityForResult(Intent(this, ThirdActivity::class.java),
RESULT_CODE)
    //without that method
    startActivity(Intent(this, ThirdActivity::class.java))

}
```

ThirdActivity

```
private fun toFirst() {
    //with startActivityForResult() method
    /*this.setResult(RESULT_OK)
    finish()*/
    //without that method
    val intent = Intent(this,
MainActivity::class.java).addFlags(FLAG_ACTIVITY_CLEAR_TOP)
```

```
    startActivity(intent)
}
```

Флаг *FLAG_ACTIVITY_CLEAR_TOP* позволяет переходить к Activity, если она уже лежит в стеке. Такие случаи возникали только в третьем переходе, поэтому больших изменений в коде не потребовалось.
Приложение работало как и предыдущее, за одним исключением: мне показалось, что переход 3 -> 1 стал чуть дольше.
В back stack не хранились дубликаты одной Activity, а когда мы переходили к другой Activity с флагом *FLAG_ACTIVITY_CLEAR_TOP*, то все Activity выше нее по стеку закрывались.

Дополнение графа навигации

Представим достаточно распространенную ситуацию:

Воскресенье 1960 года. Джош, примерный семьянин и просто славный малый, проживающий в Прайнвилле, округ Крук, штат Орегон, решил сходить в парикмахерскую. Он зашел в свой смартфон и перешел на страничку, чтобы выбрать себе новомодную прическу (телефон, ему, видимо подарили внеземные цивилизации, не об этом речь).

Так вот, он выбрал себе прическу и отправился в парикмахерскую. Все бы ничего, однако в Штате Орегон в то время действовал закон ORS 690.210, запрещающий ходить в парикмахерскую по воскресеньям.

Когда Джош почти было зашел в парикмахерскую с заднего входа, откуда ни возьмись появился не менее славный малый по имени Фред, с самого рождения поступивший в полицейскую академию штата Орегон, а ныне доблестно защищающий этот штат от правонарушителей.

После некоторого диалога на тему почему Джош идет в парикмахерскую в воскресенье, Фред быстро заподозрил неминуемое повышение за поимку особо опасного преступника. Фред взял телефон подозреваемого и смог по памяти и без ошибок ввести команду: “adb shell dumpsys activity activities | findStr lab3 | findStr Hist” (я же говорил, что он славный малый). Внимание вопрос: «Что же увидит Фред в логах?»

- Вариант первый
  Джош не знал о флаге FLAG_ACTIVITY_NO_HISTORY, позволяющем не сохранять в стеке активити, на которое был совершен переход (в случае, если с него были переходы).
  Тогда Фред увидит следующее:

```
* Hist #2: ActivityRecord{2095c56 u0 com.example.lab3_4/.ThirdActivity t81}
* Hist #1: ActivityRecord{32e1dda u0 com.example.lab3_4/.SecondActivity t81}
* Hist #0: ActivityRecord{b6b1fb9 u0 com.example.lab3_4/.MainActivity t81}
```

Рис 4. Мечта Фреда осуществилась

Фред увидит переход через второе активити (фотографии причесок), арестует преступника и получит повышение.

- Вариант второй
  Джош подсуетился и изменил исходный код следующим образом.

Листинг 5. Умный Джош

```
class MainActivity : OptionedActivity() {
    private lateinit var binding: ActivityMainBinding
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)
        binding.secondButton.setOnClickListener { toSecond() }
        binding.secondButtonSafe.setOnClickListener { toSecondSafely()
}
    }

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


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

Он добавил безопасный переход и теперь, по идее, если он посмотрит прически на втором активити и перейдет в 3, то в логах не будет видно его преступления.

```
* Hist #1: ActivityRecord{f30bcff u0 com.example.lab3_4/.ThirdActivity t81}
* Hist #0: ActivityRecord{ee81b57 u0 com.example.lab3_4/.MainActivity t81}
```

Рис. 4.1 Повышения не будет

Фред не сможет найти доказательств, поэтому отпустит Джоша. Все счастливы.

Решение с помощью Fragments

Исходную задачу можно решить, используя фрагменты и Arch Nav Component.

Все активити кроме About нужно было реализовать при помощи Fragment, переходы осуществляются через Navigation Graph.

Листинг 6.1 MainActivity

```
class MainActivity : AppCompatActivity() {

    private lateinit var binding: ActivityMainBinding
    private lateinit var appBarConfiguration: AppBarConfiguration
    private lateinit var navController : NavController

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)
        val navHostFragment =
supportFragmentManager.findFragmentById(R.id.fragmentContainerView) as
NavHostFragment
        navController = navHostFragment.navController
        appBarConfiguration =
AppBarConfiguration.Builder(navController.graph).build()
        setupActionBarWithNavController(
            navController,
            appBarConfiguration
        )
    }

    override fun onSupportNavigateUp(): Boolean {
        return navController.navigateUp(appBarConfiguration) ||
super.onSupportNavigateUp()
    }

    override fun onCreateOptionsMenu(menu: Menu): Boolean {
        menuInflater.inflate(R.menu.option_menu, menu)
        return true
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        return NavigationUI.onNavDestinationSelected(
            item, findNavController(R.id.fragmentContainerView)
        ) || super.onOptionsItemSelected(item)
    }
}
```

Листинг 6.2 BaseFragment

```
class Fragment1 : BasedFragment(R.layout.fragment1)
class Fragment2 : BasedFragment(R.layout.fragment2)
class Fragment3 : BasedFragment(R.layout.fragment3)

abstract class BasedFragment(private val res: Int) : Fragment() {

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        val layout = inflater.inflate(res, container, false)

        layout.findViewById<View>(R.id.firstButton)?.setOnClickListener {
            findNavController().navigate(R.id.action_to1)
        }

        layout.findViewById<View>(R.id.secondButton)?.setOnClickListener {
            findNavController().navigate(R.id.action_to2)
        }
```

```
        layout.findViewById<View>(R.id.thirdButton)?.setOnClickListener {
            findNavController().navigate(R.id.action_to3)
        }

        return layout
    }

}
```

В связи с тем, что из всех фрагментов можно осуществить переход в About, мной было принято решение соединить все в один класс, от которого могут наследоваться все фрагменты.

Листинг 6.3 Навигационный граф

```
<navigation xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/mobile_navigation"
    app:startDestination="@id/fragment1">

    <activity
        android:id="@+id/aboutActivity"
        android:name="com.example.lab3_5.AboutActivity"
        android:label="AboutActivity"
        app:action="@+id/action_about" />

    <fragment
        android:id="@+id/fragment1"
        android:name="com.example.lab3_5.Fragment1"
        android:label="Fragment1" >
            <action
                android:id="@+id/action_to2"
                app:destination="@id/fragment2" />
    </fragment>
    <fragment
        android:id="@+id/fragment2"
        android:name="com.example.lab3_5.Fragment2"
        android:label="Fragment2" >
            <action
                android:id="@+id/action_to3"
                app:destination="@id/fragment3" />
            <action
                android:id="@+id/action_to1"
                app:popUpTo="@id/fragment1" />
    </fragment>
    <fragment
        android:id="@+id/fragment3"
        android:name="com.example.lab3_5.Fragment3"
        android:label="Fragment3" >
            <action
                android:id="@+id/action_to2"
                app:popUpTo="@id/fragment2"/>
            <action
                android:id="@+id/action_to1"
                app:popUpTo="@id/fragment1" />
    </fragment>
</navigation>
```

Тут интерактивно можно указывать навигацию между всеми фрагментами, но мне было удобнее прописывать все руками. Из-за этого я словил багулю (указал неверный destination) и в упор не мог ее увидеть, потратив на

исправление одной строчки полтора часа. PopUpTo указывает, до какого фрагмента необходимо вытолкнуть другие элементы с верхушки стека – это необходимо для того, чтобы фрагменты не оставались в backStack.

Выводы

В первой части мной были изучены основы JetpackCompose. Мне понравился такой формат работы – я изучал документацию и параллельно с этим писал код. В целом, данный инструмент показался мне удобным.

В последующих частях работы было изучено понятие backStack, а также произведена навигация в приложении между разными активити и навигация через Navigation Graph и Fragment.

Мне кажется, что для более сложных приложений будет удобнее использовать Navigation Graph или его аналоги, потому что вся навигация, во-первых, интерактивно отображается, а во-вторых, находится в одном месте. Однако как бы это парадоксально не прозвучало, мне было проще разобраться с startActivityForResult и реализовывать работу в этом формате.

Время на выполнение ЛР

11 альбомов Оксимирона – Красота и уродство (один альбом идет 1.07 минут) на репите.

- 1 – Изучение документации + написание кода = 120 минут
- 2 – 180 минут
- 3 – 40 минут
- 4 – 50 минут (тут я больше гуглил веселые факты и думал, чего бы добавить)
- 5 – 270 минут
- Отчет писался параллельно с выполнением ЛР, поэтому в общем времени уже заложено время под отчет.

Github: github.com/pimenov01/lab3

Тестирование приложений

Проведем тестирование приложений из ЛР3 посредством инструментальных тестов. Инструментальные тесты – это тесты, запускающиеся на устройстве пользователя и позволяющие использовать все классы и методы Android. Это очень удобно, в отличие, от тестирования на компьютере, где ощущался бы недостаток во многих элементах приложения, которые мы хотим протестировать. В качестве фреймворка, с которым мы будем работать, выберем Espresso.

Сами тесты разбиты на 4 класса, в зависимости от того, что в них тестируется.

- AboutTest – проверяем поведения активити About, заходы и выходы из него
- BackstackTest – проверяем, что в Backstack не остается лишней информации об активити, после выхода из нее, через Lifecycle.State.DESTROYED
- NavigationTest – тестируем навигацию: заходы в разные активити и соответствие ожиданий с отображаемым на экране состоянием
- RecreateTest – проверяем, как ведут себя активити при повороте экрана

Также существует вспомогательный класс AboutUtils, который раньше был нужен для взаимодействия с AboutActivity, но в процессе решения в него добавились функции нажатая на кнопку и проверка отображаемого на экране. Обращаться к методам такого класса удобнее, чем каждый раз дубллировать код.

Первое и второе приложение прошли тесты без каких либо затруднений, они ведут себя также, как и референсное приложение. Для работы было необходимо добавить кнопку навигации вверх, потому что моя реализация этого не подразумевала. За этим небольшим исключением все прошло в соответствии с ожиданиями.

Приложение, в котором использовались фрагменты упорно не хотело проходить тесты AboutTest и NavigationTest из-за того, что при выходе из AboutActivity через кнопку вверх создавалась новая сущность и приложение переставало работать корректно.

Исправить это можно добавлением в манифест параметра запуска singleTop, означающим, что если уже есть экземпляр Activity с таким же типом наверху стека в вызывающей задаче, не будет создано никакого нового Activity, вместо этого Intent будет отправлен существующему экземпляру

Activity через метод onNewIntent(). После того, как выбран режим запуска, все тесты проходили без неудач.

Выводы по ЛР4

В ходе данной ЛР я ознакомился с UI тестами и с их преимуществами, относительно обычных тестов. Функциональность UI тестирования мне показалась очень удобной и наглядной, можно без проблем на экране смотреть, как ведет себя приложение в зависимости от написанных тестов и даже понять, на каком моменте оно ведет себя отличным образом.

Также тестирование помогло мне выявить ошибки и недочеты, которые имелись в прошлой реализации, и исправить их.