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

Отчёт по лабораторной работе № 4

Дисциплина: Проектирование мобильных приложений

Тема: UI-Тесты, Рефакторинг и Исправление Ошибок.

Выполнил студент гр. 3530901/90201 _____З.А. Фрид
(подпись)

Принял преподаватель _____ А.Н. Кузнецов
(подпись)

“____” _____ 2021 г.

Санкт-Петербург

2021

Оглавление

Ссылка на проекты 3
Цели 3
Задачи 3
Задача 1. 3
Задача 2. Навигация (startActivityForResult) 3
Задача 3. Навигация (флаги Intent/атрибуты Activity) 4
Задача 4. Навигация (флаги Intent/атрибуты Activity) 4
Задача 5. Навигация (Fragments, Navigation Graph) 5
Задача 1 5
Задача 2. Навигация (startActivityForResult) 7
Задача 3. Навигация (флаги Intent/атрибуты Activity) 11
Задача 4. Навигация (флаги Intent/атрибуты Activity) 13
Задача 5. Навигация (Fragments, Navigation Graph) 15
Тестирование 17
Вывод 17
Затраченное время на лабораторную №3: 18
Затраченное время на лабораторную №4: 18
Список источников 19

Ссылка на проекты

https://github.com/zina-frid/AndroidLabs/tree/main/projects/lab4

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
- Познакомиться с принципами и получить практические навыки разработки UI тестов для Android приложений.
- Познакомиться с инструментами IDE для поддержки рефакторинга кода.

Задачи

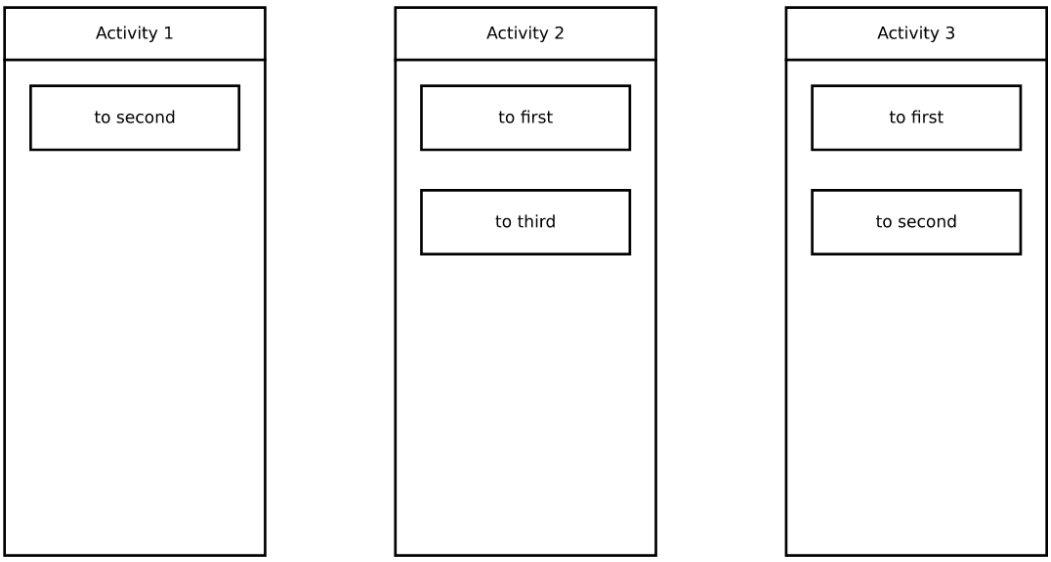
Задача 1.

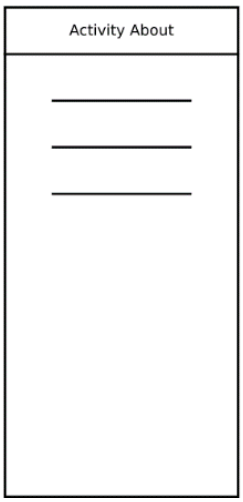
Задание 1.б. Использование Jetpack Compose для верстки layouts

Познакомьтесь с содержимым курса Jetpack Compose и выполните codelab "Jetpack Compose basics".

Задача 2. Навигация (startActivityForResult)

Реализуйте навигацию между экранами одного приложения согласно изображению ниже с помощью Activity, Intent и метода startActivityForResult.





Пояснения

- Приложение начинает работу с отображения Activity 1
- Кнопка 'to first' отображает на экране Activity 1
- Кнопка 'to second' отображает на экране Activity 2
- Кнопка 'to third' отображает на экране Activity 3
- В любой момент в backstack любого task приложения должно быть не более 4 activity
- Во всех вариантах Activity 'About' должна быть доступна из любой другой Activity одним из способов согласно варианту.
- Activity в BackStack (кроме About) всегда остаются в одном и том же порядке: 1-2-3

Вариант 23, следовательно, во всех вариантах Activity 'About' должна быть доступна из любой другой Activity с помощью Bottom Navigation.

Задача 3. Навигация (флаги Intent/атрибуты Activity)

Решите предыдущую задачу с помощью Activity, Intent и флагов Intent либо атрибутов Activity.

Задача 4. 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.
Ограничение на размер back stack к этому заданию не применяется.

Задача 5. 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.

Задача 1.
Jetpack Compose — это современный набор инструментов, предназначенный для упрощения разработки пользовательского интерфейса. Приложение Compose состоит из составных функций — обычных функций Kotlin, отмеченных @Composable, которые могут вызывать другие составные функции. Аннотация сообщает Compose о необходимости добавить в функцию особую поддержку для обновления и поддержки UI с течением времени. Compose позволяет структурировать код на небольшие фрагменты.

Создавая небольшие компоненты, которые используются повторно, легко создать библиотеку элементов UI, используемых в приложении. Каждый отвечает за одну часть экрана и может редактироваться независимо.

При создании проекта выберем Empty Compose Activity. Минимальная версия API для использования Compose – 21.

В заготовке проекта можно увидеть пример использования одной составной функции в другой. Например, в функции Greeting используется библиотечная функция Text(), библиотечная функция Text():

Листинг 1.1 Функция Greeting

```
@Composable
fun Greeting(name: String) {
    Text(text = "Hello $name!")
}
```

Листинг 1.2 Функция Text() в пакете android.compose.material

```
@Composable
fun Text(
    text: String,
    modifier: Modifier = Modifier,
    color: Color = Color.Unspecified,
    fontSize: TextUnit = TextUnit.Unspecified,
    fontStyle: FontStyle? = null,
    fontWeight: FontWeight? = null,
    fontFamily: FontFamily? = null,
    letterSpacing: TextUnit = TextUnit.Unspecified,
    textDecoration: TextDecoration? = null,
    textAlign: TextAlign? = null,
    lineHeight: TextUnit = TextUnit.Unspecified,
    overflow: TextOverflow = TextOverflow.Clip,
    softWrap: Boolean = true,
    maxLines: Int = Int.MAX_VALUE,
    onTextLayout: (TextLayoutResult) -> Unit = {},
    style: TextStyle = LocalTextStyle.current
) {
..........
```

Чтобы использовать предварительный просмотр Android Studio, нам просто нужно пометить любую compose функцию без параметров или функции с параметрами по умолчанию с помощью аннотации @Preview. Пример:

Листинг 1.3 Функция DefaultPreview

```
@Preview(showBackground = true, widthDp = 320)
@Composable
fun DefaultPreview() {
    Task1Theme {
        Greetings()
    }
}
```

Компонент, который представляет по сути то, что мы увидим на экране устройства, передается в метод setContent() класса MainActivity.

По ходу курса нам предлагается изучить некоторые функции и объекты. Surface фактически представляет промежуточный компонент, который задает дополнительное оформление в стиле Material Design. Объект Modifier предоставляет огромное количество встроенных функций-модификаторов, изменяющих отдельные аспекты компонентов.

Благодаря контейнерам компоновки Row, Column, Box мы можем различными способами размещать объекты на экране. Использование LazyColumn позволило реализовать прокручивающийся список. В ходе работы был добавлен новый экран, и реализован переход от него к основному. Для того, чтобы сохранить состояние UI были использованы remember и mutableStateOf. Также была добавлена анимация, были изменены цвета для ночной и дневной темы, а также были внесены некоторые другие изменения, для улучшения внешнего вида.

Задача 2. Навигация (startActivityForResult)

Необходимо реализовать навигацию между экранами одного приложения согласно изображению ниже с помощью Activity, Intent и метода startActivityForResult. Для решения задачи понадобились следующие методы Activity:

- startActivity(Intent) – метод, используемый, чтобы создать новую Activity, которая будет размещена в верхней части стека. Принимает один аргумент Intent, который описывает выполняемое действие.
- startActivityForResult(Intent, int) – метод, используемый, чтобы создать новую Activity, если нам необходимо получать результат от Activity, когда оно завершится.
- setResult(int) – метод, который возвращает данные родителю.
- onActivityResult(int, int, Intent) - метод, обрабатывающий результат работы Activity.
- finish – метод, завершающий работу Activity

Были созданы 3 основные Activity и одно для About Activity, в которое реализован переход из всех Activity при помощи Bottom Navigation. Были реализованы следующие переходы между Activity.

MainActivity

Листинг 2.1 Переход ко второму Activity

```
override fun onCreate(savedInstanceState: Bundle?) {
    . . . . . .
    binding.bnToSecond.setOnClickListener { toSecondAct() }
    . . . . . .
}

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


override fun onCreate(savedInstanceState: Bundle?) {
    . . . . . .
    binding.toSecond.setOnClickListener { toSecondAct() }
    . . . . . .
}
```

Переход к About Activity в MainActivity, SecondActivity и ThirdActivity реализованы идентично.

Листинг 2.2 Переход к About Activity

```
override fun onCreate(savedInstanceState: Bundle?) {
    . . . . . .
    binding.navView.setOnNavigationItemSelectedListener { toAboutAct(it) }
}

    . . . . . .
private fun toAboutAct(item: MenuItem): Boolean {
    when (item.itemId) {
        R.id.about -> {
            startActivity(Intent(this, AboutActivity::class.java))
        }
    }
    return false
}
```

SecondActivity

Для перехода к первому Activity мы просто вызываем finish(). Для перехода к третьему Activity используем startActivityForResult, т. к. из третьего нам нужно будет переходит к первому и для этого разрушать вторую Activity. Если третий экран завершиться с результатом RESULT_OK мы завершаем данную activity.

Листинг 2.3 Переход к первому и третьему Activity

```
override fun onCreate(savedInstanceState: Bundle?) {
    . . . . . .
```

```
    binding.bnToFirst.setOnClickListener { toFirstAct() }
    binding.bnToThird.setOnClickListener { toThirdAct() }
    . . . . . .
}

private fun toFirstAct() {
    finish()
}

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

private fun toThirdAct() {
    val intent = Intent(this, ThirdActivity::class.java)
    startActivityForResult(intent, REQUEST_FROM_THIRD)
}

. . . . . .

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

ThirdActivity

Если переходим к первому Activity, то проходим через второе Activity и завершаем его. Если переходим ко второму Activity, то просто завершаем его.

Листинг 2.4 Переход к первому и второму Activity

```
override fun onCreate(savedInstanceState: Bundle?) {
    . . . . . .
    binding.bnToFirst.setOnClickListener { toFirstAct() }
    binding.bnToSecond.setOnClickListener { toSecondAct() }
    . . . . . .
}

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

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

По заданию необходимо, чтобы в backstack не было разных сущностей одной Activity. Теперь проверим, что при работе приложения используется не более 4-х Activity. Для этого введем команды:

adb shell

dumpsys activity activities | grep 'Hist #' | grep 'com.zinafrid.task2'

Запуск приложения:

```
C:\Users\Z\AndroidStudioProjects\Labs\AndroidLabs\projects\lab3\task2> adb shell
generic_x86:/ $ dumpsys activity activities | grep 'Hist #' | grep 'com.zinafrid.task2'
      * Hist #0: ActivityRecord{bef6989 u0 com.zinafrid.task2/.FirstActivity t9}
```

Рис. 2-1 В стеке только FirstActivity

Перейдем в SecondActivity:

```
generic_x86:/ $ dumpsys activity activities | grep 'Hist #' | grep 'com.zinafrid.task2'
      * Hist #1: ActivityRecord{460a600 u0 com.zinafrid.task2/.SecondActivity t9}
      * Hist #0: ActivityRecord{bef6989 u0 com.zinafrid.task2/.FirstActivity t9}
```

Рис. 2-2 В стек добавилось SecondActivity

Перейдем в ThirdActivity:

```
generic_x86:/ $ dumpsys activity activities | grep 'Hist #' | grep 'com.zinafrid.task2'
      * Hist #2: ActivityRecord{88c388f u0 com.zinafrid.task2/.ThirdActivity t9}
      * Hist #1: ActivityRecord{22a8c43 u0 com.zinafrid.task2/.SecondActivity t9}
      * Hist #0: ActivityRecord{bef6989 u0 com.zinafrid.task2/.FirstActivity t9}
```

Рис. 2-3 В стек добавилось ThirdActivity

Откроем AboutActivity:

```
generic_x86:/ $ dumpsys activity activities | grep 'Hist #' | grep 'com.zinafrid.task2'
      * Hist #3: ActivityRecord{5f12a6f u0 com.zinafrid.task2/.AboutActivity t9}
      * Hist #2: ActivityRecord{88c388f u0 com.zinafrid.task2/.ThirdActivity t9}
      * Hist #1: ActivityRecord{22a8c43 u0 com.zinafrid.task2/.SecondActivity t9}
      * Hist #0: ActivityRecord{bef6989 u0 com.zinafrid.task2/.FirstActivity t9}
```

Рис. 2-4 В стек добавилось AboutActivity

Закроем AboutActivity, а затем вернемся из ThirdActivity в FirstActivity:

```
generic_x86:/ $ dumpsys activity activities | grep 'Hist #' | grep 'com.zinafrid.task2'
      * Hist #2: ActivityRecord{88c388f u0 com.zinafrid.task2/.ThirdActivity t9}
      * Hist #1: ActivityRecord{22a8c43 u0 com.zinafrid.task2/.SecondActivity t9}
      * Hist #0: ActivityRecord{bef6989 u0 com.zinafrid.task2/.FirstActivity t9}
generic_x86:/ $ dumpsys activity activities | grep 'Hist #' | grep 'com.zinafrid.task2'
      * Hist #0: ActivityRecord{bef6989 u0 com.zinafrid.task2/.FirstActivity t9}
```

Рис. 2-5 Состояние стека

Из приведённых выше результатов видно, что в стеке не хранятся разные сущности одной Activity.

Вопрос: что будет, если не зарегистрировать Activity в манифесте?

Ответ: если не указать все Activty в Манифесте, то приложение закроется при попытке перейти в незарегистрированное Activity.

Задача 3. Навигация (флаги Intent/атрибуты Activity)

Решим предыдущую задачу другим способом. Необходимо использовать Activity, Intent и флаги Intent либо атрибуты Activity. Зададим флаг FLAG_ACTIVITY_CLEAR_TOP, который позволяет переходить к Activity, если она уже есть в стеке, и уничтожать все Activity выше него в стеке.

Были изменены следующие листинги:

SecondActivity

Листинг 3.1 Переход к первому и третьему Activity

```
override fun onCreate(savedInstanceState: Bundle?) {
    . . . . . .
    binding.bnToFirst.setOnClickListener { toFirstAct() }
    binding.bnToThird.setOnClickListener { toThirdAct() }
    . . . . . .
}

private fun toFirstAct() {
    finish()
}

private fun toThirdAct() {
    val intent = Intent(this, ThirdActivity::class.java)
    startActivity(intent)
}
```

ThirdActivity

Листинг 3.2 Переход к первому и второму Activity

```
override fun onCreate(savedInstanceState: Bundle?) {
    . . . . . .
```

```
    binding.bnToFirst.setOnClickListener { toFirstAct() }
    binding.bnToSecond.setOnClickListener { toSecondAct() }
    . . . . . .
}

private fun toFirstAct() {
    val intent = Intent(this, MainActivity::class.java)
    intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP)
    startActivity(intent)
}

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

Тестирование было проведено, как и в предыдущем случае. Результат были такими же, однако переход от третьего Activity происходит заметно дольше.

Запуск приложения:

```
C:\Users\Z\AndroidStudioProjects\Labs\AndroidLabs\projects\lab3\task3>adb shell
generic_x86:/ $ dumpsys activity activities | grep 'Hist #' | grep 'com.zinafrid.task3'
      * Hist #0: ActivityRecord{b983b00 u0 com.zinafrid.task3/.FirstActivity t9}
```

Рис. 3-1 В стеке только FirstActivity

Перейдем в SecondActivity:

```
generic_x86:/ $ dumpsys activity activities | grep 'Hist #' | grep 'com.zinafrid.task3'
      * Hist #1: ActivityRecord{179f4ed u0 com.zinafrid.task3/.SecondActivity t9}
      * Hist #0: ActivityRecord{b983b00 u0 com.zinafrid.task3/.FirstActivity t9}
```

Рис. 3-2 В стек добавилось SecondActivity

Перейдем в ThirdActivity:

```
generic_x86:/ $ dumpsys activity activities | grep 'Hist #' | grep 'com.zinafrid.task3'
      * Hist #2: ActivityRecord{ad483d9 u0 com.zinafrid.task3/.ThirdActivity t9}
      * Hist #1: ActivityRecord{10daca2 u0 com.zinafrid.task3/.SecondActivity t9}
      * Hist #0: ActivityRecord{b983b00 u0 com.zinafrid.task3/.FirstActivity t9}
```

Рис. 3-3 В стек добавилось ThirdActivity

Откроем AboutActivity:

```
generic_x86:/ $ dumpsys activity activities | grep 'Hist #' | grep 'com.zinafrid.task3'
      * Hist #3: ActivityRecord{87550ac u0 com.zinafrid.task3/.AboutActivity t9}
      * Hist #2: ActivityRecord{ad483d9 u0 com.zinafrid.task3/.ThirdActivity t9}
      * Hist #1: ActivityRecord{10daca2 u0 com.zinafrid.task3/.SecondActivity t9}
      * Hist #0: ActivityRecord{b983b00 u0 com.zinafrid.task3/.FirstActivity t9}
```

Рис. 3-4 В стек добавилось AboutActivity

Закроем AboutActivity, а затем вернемся из ThirdActivity в FirstActivity:

```
generic_x86:/ $ dumpsys activity activities | grep 'Hist #' | grep 'com.zinafrid.task3'
      * Hist #2: ActivityRecord{ad483d9 u0 com.zinafrid.task3/.ThirdActivity t9}
      * Hist #1: ActivityRecord{10daca2 u0 com.zinafrid.task3/.SecondActivity t9}
      * Hist #0: ActivityRecord{b983b00 u0 com.zinafrid.task3/.FirstActivity t9}
generic_x86:/ $ dumpsys activity activities | grep 'Hist #' | grep 'com.zinafrid.task3'
      * Hist #0: ActivityRecord{fa431d7 u0 com.zinafrid.task3/.FirstActivity t9}
```

Рис. 3-5 Состояние стека

Из приведённых выше результатов видно, что в back stack не появляется дубликатов Activity, и когда мы переходили к другому Activity с флагом FLAG_ACTIVITY_CLEAR_TOP, то все Activity выше него в стеке закрывались.

Задача 4. Навигация (флаги Intent/атрибуты Activity)

Граф переходов был дополнен переходом из FirstActivity в ThirdActivity:

Листинг 4.1 Переход toThirdAct

```
override fun onCreate(savedInstanceState: Bundle?) {
    . . . . . .
    binding.toThird.setOnClickListener { toThirdAct() }
    . . . . . .
}
    . . . . . .

private fun toThirdAct() {
    val intent = Intent(this, ThirdActivity::class.java)
    startActivity(intent)
}
```

Для того, чтобы в дальнейшем лучше продемонстрировать работу выбранного флага, при переходе из ThirdActivity в SecondActivity не будем закрывать само ThirdActivity, а будем создавать новый экземпляр SecondActivity:

Листинг 4.2 Переход от третьего во второе Activity

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

Теперь перейдем из FirstActivity в ThirdActivity, затем перейдем в SecondActivity и посмотрим состояние стека.

```
C:\Users\Z\AndroidStudioProjects\Labs\AndroidLabs\projects\lab3\task4>adb shell
generic_x86:/ $ dumpsys activity activities | grep 'Hist #' | grep 'com.zinafrid.task4'
      * Hist #0: ActivityRecord{174e4a4 u0 com.zinafrid.task4/.FirstActivity t12}
generic_x86:/ $ dumpsys activity activities | grep 'Hist #' | grep 'com.zinafrid.task4'
      * Hist #1: ActivityRecord{5ae9945 u0 com.zinafrid.task4/.ThirdActivity t12}
      * Hist #0: ActivityRecord{174e4a4 u0 com.zinafrid.task4/.FirstActivity t12}
generic_x86:/ $ dumpsys activity activities | grep 'Hist #' | grep 'com.zinafrid.task4'
      * Hist #2: ActivityRecord{510a469 u0 com.zinafrid.task4/.SecondActivity t12}
      * Hist #1: ActivityRecord{5ae9945 u0 com.zinafrid.task4/.ThirdActivity t12}
      * Hist #0: ActivityRecord{174e4a4 u0 com.zinafrid.task4/.FirstActivity t12}
```

Рис. 4-1 Состояние стека до использования флага

Из приведенных выше результатов мы видим, что в стеке находятся экземпляр ThirdActivity.

Изменим переход от FirstActivity к ThirdActivity, добавив флаг FLAG_ACTIVITY_NO_HISTORY:

Листинг 4.3 Исправленный переход toThirdAct

```
private fun toThirdAct() {
    val intent = Intent(this, ThirdActivity::class.java)
    intent.addFlags(Intent.FLAG_ACTIVITY_NO_HISTORY)
    startActivity(intent)
}
```

Если данный флаг установлен, то Activity не сохраняется в backstack.

Снова перейдем из FirstActivity в ThirdActivity, затем перейдем в SecondActivity и посмотрим состояние стека.

```
generic_x86:/ $ dumpsys activity activities | grep 'Hist #' | grep 'com.zinafrid.task4'
      * Hist #0: ActivityRecord{d07b171 u0 com.zinafrid.task4/.FirstActivity t14}
generic_x86:/ $ dumpsys activity activities | grep 'Hist #' | grep 'com.zinafrid.task4'
      * Hist #1: ActivityRecord{721a6fa u0 com.zinafrid.task4/.ThirdActivity t14}
      * Hist #0: ActivityRecord{d07b171 u0 com.zinafrid.task4/.FirstActivity t14}
generic_x86:/ $ dumpsys activity activities | grep 'Hist #' | grep 'com.zinafrid.task4'
      * Hist #1: ActivityRecord{4339968 u0 com.zinafrid.task4/.SecondActivity t14}
      * Hist #0: ActivityRecord{d07b171 u0 com.zinafrid.task4/.FirstActivity t14}
```

Рис. 4-2 Состояние стека после использования флага

Из приведенных выше результатов мы видим, что при переходе к SecondActivity экземпляр ThirdActivity не сохраняется в стеке.

Задача 5. Навигация (Fragments, Navigation Graph)

Исходную задачу необходимо решить с помощью Fragments и Navigation Graph. Все переходы между окнами приложения были осуществлены при помощи Navigation Graph. Для этой цели использовался класс NavController и его метод navigate(), в котором можно использовать только переходы, заданные в Navigation Graph. Для получения экземпляр самого класса NavController использовался метод findNavController().

Реализация переходов во фрагментах с помощью навигационного графа.

Листинг 5.1 FirstFragment

```
val binding = FragmentFirstBinding.inflate(layoutInflater)
val navController = findNavController()
binding.toSecond.setOnClickListener {
    navController.navigate(R.id.first_frag_to_second_frag)
}

binding.bottomAbout.setOnNavigationItemSelectedListener {
    when (it.itemId) {
        R.id.about -> {
            navController.navigate(R.id.global_about)
        }
    }
    false
}
```

Листинг 5.2 SecondFragment

```
val binding = FragmentSecondBinding.inflate(layoutInflater)
val navController = findNavController()
binding.toFirst.setOnClickListener {
    navController.navigate(R.id.second_frag_to_first_frag)
}
binding.toThird.setOnClickListener {
    navController.navigate(R.id.second_frag_to_third_frag)
}

binding.bottomAbout.setOnNavigationItemSelectedListener {
    when (it.itemId) {
        R.id.about -> {
            navController.navigate(R.id.global_about)
        }
    }
    false
}
```

Листинг 5.3 ThirdFragment

```
val binding = FragmentThirdBinding.inflate(layoutInflater)
val navController = findNavController()

binding.toFirst.setOnClickListener {
    navController.navigate(R.id.third_frag_to_first_frag)
}
binding.toSecond.setOnClickListener {
    navController.navigate(R.id.third_frag_to_second_frag)
}

binding.bottomAbout.setOnNavigationItemSelectedListener {
    when (it.itemId) {
        R.id.about -> {
            navController.navigate(R.id.global_about)
        }
    }
    false
}
```

В рамках задания был разработан следующий граф переходов между фрагментами:

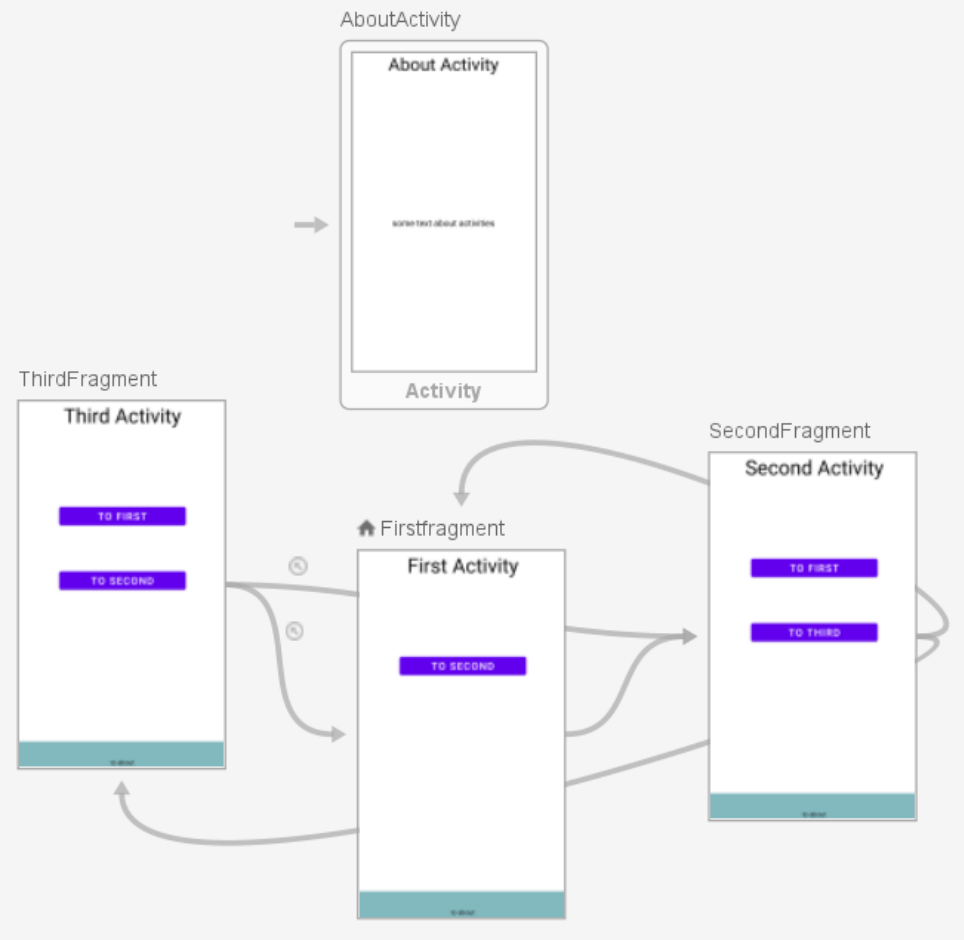


Рис. 5-1 Navigation Graph

Для того, чтобы лишние экземпляры фрагментов удалялись из backstack при переходах между фрагментами, использовались атрибуты popUpTo (удаляет из backstack все фрагменты выше указанного) и popUpToInclusive (в значении true обязывает popUpTo удалять из backstack все фрагменты, включая указанный).

В графе указано глобальное действие, которое может выполняться из любого activity или фрагмента в навигационном графе (переход к AboutActivity).

Тестирование

В ходе выполнения лабораторной работы №4 были написаны различные тесты для проверки референсного проекта. Затем на этих тестах были проверены ранее написанные проекты. Был осуществлен рефакторинг. В ходе тестирования ошибки выявлены не были.

Вопрос: объясните, кто и в какой момент сохраняет состояние EditText.

Ответ: состояние EditText сохраняется в методе onSaveInstanceState в Bundle. Этот метод вызывается, когда приложение останавливается (переходит в состояние stopped), а стандартная реализация метода сохраняет информацию о GUI (в нашем случае состояние EditText).

Вывод

В первом пункте лабораторной работы был выбран вариант с выполнением Codelab по Jetpack Compose. В целом мне понравился сам формат представленных заданий, когда сначала ставит задачу, рассказывают о различных функциях, которые можно использовать, про их тонкости, потом предлагают решить задачу самому, но в конце есть ответ, если вдруг что-то не сработало. К тому же я впервые узнала про Jetpack Compose и в ходе работы поняла, что это очень удобный инструмент для написания UI.

Во второй части работы мы познакомились с backstack. В ходе работы были выполнены задания по навигации в приложении и переходам между Activity. Так как в навигации присутствовал сложный переход от третьего экрана к первому, во избежание присутствия дубликатов одной Activity в backstack использовались startActivityForResult и флаги Intent.

В третьей части работы три основные Activity были заменены на Fragment’ы, для навигации между которыми был использован Navigation Graph.

Все способы решения поставленной задачи различаются. С одной стороны, реализовывать навигацию с помощью Navigation Graph удобнее для такого чтобы в дальнейшем расширять или модифицировать приложение, потому что сложные переходы создаются относительно легко, все они видны на графе, и к тому же переходы выполняются быстрее, чем переходы на Activity. С другой стороны, реализация из пунктов 2 и 3 намного понятнее, и код легко читается.

Затраченное время на лабораторную №3:

1-ый пункт ~ 2 часа;

2-ой пункт ~ 4,5 часа;

3-ий пункт ~ 1,5 часа;

4-ый пункт ~ 1 час;

5-ый пункт ~ 3,5 часа;

Затраченное время на лабораторную №4:

1-ый пункт ~ 2 часа;

2-ой пункт ~ 2 часа;

3-ий пункт ~ 1,5 часа;

4-ый пункт ~ 1 час;

Список источников

1. https://github.com/andrei-kuznetsov/android-lectures
2. https://developer.android.com/
3. Codelab по Jetpack Compose