ПРАКТИЧЕСКАЯ РАБОТА №1

1.1. Написание простого приложения на Kotlin с несколькими экранами

Код, связанный с созданием и управлением активностями в Android-приложении, обычно размещается в папке src внутри проектного пакета. Для каждого класса представления и активности в инструментах разработчика Android предусмотрено текстовое описание объектов этого класса на языке XML с использованием предопределенных тегов для экземпляров класса и их свойств. Это XML-описание позволяет создать визуальный макет активности.

Для указания свойств, принадлежащих объектам из пакета Android, используется префикс "android:". Значения свойств присваиваются с использованием стандартного синтаксиса XML, где значение свойства указывается после знака "=" в виде текста.

Одними из обязательных свойств всех представлений (Views) являются размеры области экрана, выделенные для отображения представления. Эти размеры включают ширину и высоту представления (View).

На активности в Android объекты обычно размещаются в контейнерах, называемых ViewGroup. Каждая активность должна иметь ровно один корневой элемент View, который может быть контейнером или отдельным виджетом. Для каждого контейнера, а также для каждого виджета, существует соответствующий класс на языке Kotlin, который определяет структуру этого представления.

Макет активности можно описать в файле разметки, который размещается в папке layout внутри папки res, или можно создать его программно. Основным контейнером или разметкой в макете активности служит layout, в которой размещаются представления. В зависимости от количества и расположения размещаемых объектов, выбирается подходящая разметка в качестве корневого элемента. Разметка может также быть вложенной, то есть дочерним элементом другой разметки, что позволяет создавать сложные иерархии слоёв для размещения представлений.

В файле разметки XML объекты описываются с использованием парных тегов <Тип объекта> </Тип объекта>. В Java-файле необходимо импортировать стандартный класс из пакета android.widget.Тип_объекта, а затем создать константу этого типа. В программном коде доступ к объекту осуществляется через его идентификатор, который задается с помощью атрибута android:id. Поэтому все создаваемые представления должны иметь заданный этот атрибут. Чтобы создать объект на основе макета представления, используется метод

findViewById(), которому передается идентификатор создаваемого объекта в качестве входного параметра.

Листинг 1.1. Описание объекта в макете

```
< TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/sample"/>
```

Листинг 1.2. Описание объекта в коде

```
var textView: TextView = findViewById(R.id.textView)
```

Размещение элементов View на экране зависит от контейнера ViewGroup (Layout), в котором они находятся.

1. LinearLayout — отображает элементы View в виде одной строки (если ориентация горизонтальная) или одного столбца (если ориентация вертикальная).
2. TableLayout — отображает элементы в виде таблицы, расположенной по строкам и столбцам.
3. RelativeLayout — каждый элемент можно расположить относительно других элементов, устанавливая их позиции относительно друг друга.
4. AbsoluteLayout — это контейнерный макет, который позволяет размещать элементы пользовательского интерфейса внутри себя в виде слоев или кадров. Он располагает все свои дочерние элементы в левом верхнем углу и перекрывает их друг друга. При этом, каждый последующий дочерний элемент добавляется поверх предыдущего.

Демонстрация расположения элементов на экране в представленных контейнерах представлена на Рисунке 1.1.

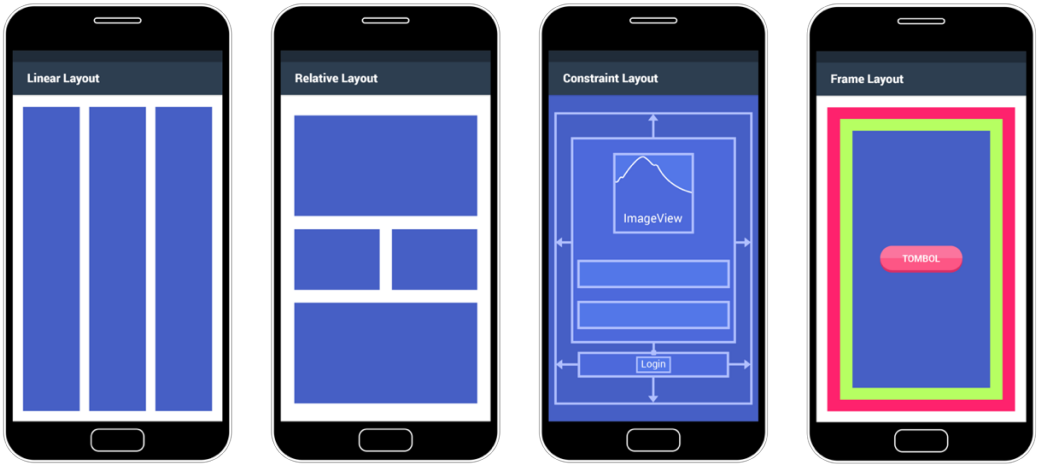

Рисунок 1.1. Расположение элементов на экране в разных контейнерах

На одном контейнере может быть размещено сколько угодно элементов View(в зависимости от их размера). Однако, размещать и связывать элементы макетов с кодом используя findViewById() является достаточно проблематичным и устаревшим методом. Для облегчения связывания макетов с кодом компанией Google был создан инструмент ViewBinding.

Для его активации необходимо установить параметр сборки viewBinding в true в файле build.gradle на уровне модуля, как показано в следующем примере:

Листинг 1.3. Активация viewBinding

```
android {
    ...
    buildFeatures {
        viewBinding = true
    }
}
```

Далее, необходимо в коде нужного Activity прописать следующее:

Листинг 1.4. Использование viewBinding в коде

```
// Объявление переменной для привязки к разметке
private lateinit var binding: ActivityBinding

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    // Создание объектов разметки с использованием класса
ActivityBinding
    binding = ActivityBinding.inflate(layoutInflater)
```

```
    // Получение корневого представления разметки
    val view = binding.root

    // Установка корневого представления разметки в качестве
контента активности
    setContentView(view)
    binding.button.setOnClickListener { // Установка слушателя
нажатий на кнопку
       binding.name.text = "Новое имя"
    }
}
```

После активации ViewBinding в файле build.gradle и создания объектов разметки с использованием класса ActivityBinding, вы можете легко связывать элементы макета с кодом вашей активности.

1.2. Навигация в Kotlin

Навигация — важная и неотъемлемая часть разработки приложений для платформы Android. Она позволяет пользователям перемещаться между экранами и обеспечивает удобство использования приложения. В языке программирования Kotlin существует несколько способов реализации навигации в Android-приложения. Давайте рассмотрим несколько основных способов навигации на Kotlin.

1.2.1. Intents (Интенты)

Интенты — это основной механизм навигации в Android. С их помощью можно запускать активности (Activity) и передавать данные между ними. Для перехода на другой экран с помощью интента нужно создать объект Intent, указать текущий контекст и целевую активность, а затем выполнить метод startActivity(). Например, следующий код отвечает за переход на другую активность:

Листинг 1.5. Переход на другую активность

```
val intent = Intent(this, TargetActivity::class.java)
startActivity(intent)
```

Интенты также позволяют передавать дополнительные данные или параметры между активностями. Для этого используются методы putExtra() и getExtra(). Например:

Листинг 1.6. Передача и получение данных между активностями

```
// Передача данных между активностями
val intent = Intent(this, TargetActivity::class.java)
intent.putExtra("key", value)
startActivity(intent)

// Получение данных между активностями
val receivedValue = intent.getStringExtra("key")
```

В приведенном примере, данные передаются из одной активности в другую. Сначала создается объект Intent с указанием текущей активности (this) и целевой активности (TargetActivity::class.java). Затем с помощью метода putExtra() в объект Intent добавляется пара ключ-значение, где "key" - это ключ, по которому данные будут доступны в целевой активности, а value - значение, которое нужно передать.

1.2.2. Методы внутри фрагментов

Ручное управление транзакциями фрагментов: Вы можете использовать методы FragmentManager для добавления, удаления и замены фрагментов в вашей активности или контейнере фрагментов. Например, чтобы заменить текущий фрагмент другим фрагментом, вы можете использовать следующий код:

Листинг 1.7. Перемещение между фрагментами

```
val fragmentManager = supportFragmentManager
val transaction = fragmentManager.beginTransaction()
transaction.replace(R.id.container, newFragment)
transaction.addToBackStack(null)     // Добавление транзакции в
стек возврата
transaction.commit()
```

В приведенном примере демонстрируется ручное управление транзакциями фрагментов в Android. Для этого используются методы FragmentManager.

Сначала получаем экземпляр FragmentManager с помощью метода supportFragmentManager. Затем создаем новую транзакцию с помощью метода beginTransaction().

Далее, с помощью метода replace() указываем контейнер (например, R.id.container), в котором будет размещен новый фрагмент, и новый фрагмент (newFragment), который будет отображен вместо текущего фрагмента.

Метод addToBackStack(null) добавляет текущую транзакцию в стек возврата, что позволяет пользователю вернуться к предыдущему фрагменту при нажатии кнопки «Назад».

Наконец, вызываем метод commit() для применения транзакции и применения изменений.

1.2.3. Navigation API

API навигации — это компонент AndroidX (Android JetPack), который упрощает управление и реализацию переходов между различными компонентами вашего приложения, такими как активности и фрагменты. Для использования API навигации вам необходимо определить маршруты вашего приложения в виде графа навигации. Для этого необходимо создать файл ресурсов навигации в папке res/navigation вашего проекта, а также добавить зависимости в файл build.gradle. Про последнее стоит упомянуть подробнее.

Зависимости в Android — это внешние библиотеки или модули, которые используются в проекте для добавления дополнительных функций или возможностей. Они представляют собой код, написанный другими разработчиками, который вы можете включить в свой проект.

Объявление зависимостей происходит в файле build.gradle вашего проекта. В Android-проекте обычно есть два уровня build.gradle файлов:

– Project-level build.gradle: В этом файле объявляются зависимости, относящиеся к всему проекту, включая плагины и классы сборки. Обычно это файл с верхним уровнем проекта.

– App-level build.gradle: В этом файле объявляются зависимости, специфичные для модуля приложения. Здесь вы можете указать зависимости для библиотек Android, таких как библиотеки поддержки, библиотеки для работы с сетью, базами данных и другие.

Объявление зависимостей в правильном месте позволяет системе сборки (например, Gradle) автоматически загрузить и включить эти зависимости в проект при сборке приложения.

Листинг 1.8. Зависимости для использования Navigation API

```
implementation("androidx.navigation:navigation-fragment-ktx: 2.7.0
")
implementation("androidx.navigation:navigation-ui-ktx: 2.7.0")
```

Что касается файла навигации, он может называться nav_graph.xml. Далее его необходимо заполнить либо вручную, прописав все возможные пути

навигации, или использовать редактор навигации. В конечном итоге файл nav_graph.xml может представлять из себя следующее.

Листинг 1.9. Перемещения между фрагментами в файле nav_graph.xml

```
<navigation
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <fragment
        android:id="@+id/fragment1"
        android:name="com.example.Fragment1"

        android:label="Fragment 1">
        <action
            android:id="@+id/action_fragment1_to_fragment2"
            app:destination="@id/fragment2" />
    </fragment>

    <fragment
        android:id="@+id/fragment2"
        android:name="com.example.Fragment2"
        android:label="Fragment 2" />
</navigation>
```

Затем необходимо настроить навигацию в коде самого фрагмента, а также в разметке главной активности.

Листинг 1.10. Перемещения между фрагментами в коде

```
class Fragment1 : Fragment() {
    private lateinit var binding: Fragment1Binding

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {

        // Создание экземпляра класса Fragment1Binding и
связывание его с разметкой фрагмента
        binding = Fragment1Binding.inflate(inflater, container,
false)

        binding.buttonGotoScreen2.setOnClickListener {
```

```
            // Навигация к другому фрагменту с помощью
NavController и указанием ID действия
            findNavController().navigate(R.id.
action_fragment1_to_fragment2)
        }
        // Возвращение корневого View разметки фрагмента
        return binding.root
    }
}
```

Листинг 1.11. Разметка MainActivity

```
<?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=".MainActivity">

    <androidx.fragment.app.FragmentContainerView
        android:id="@+id/nav_host_fragment"



android:name="androidx.navigation.fragment.NavHostFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:defaultNavHost="true"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:navGraph="@navigation/nav_graph" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Итоговый вид файла навигации представлен на Рисунке 1.2.

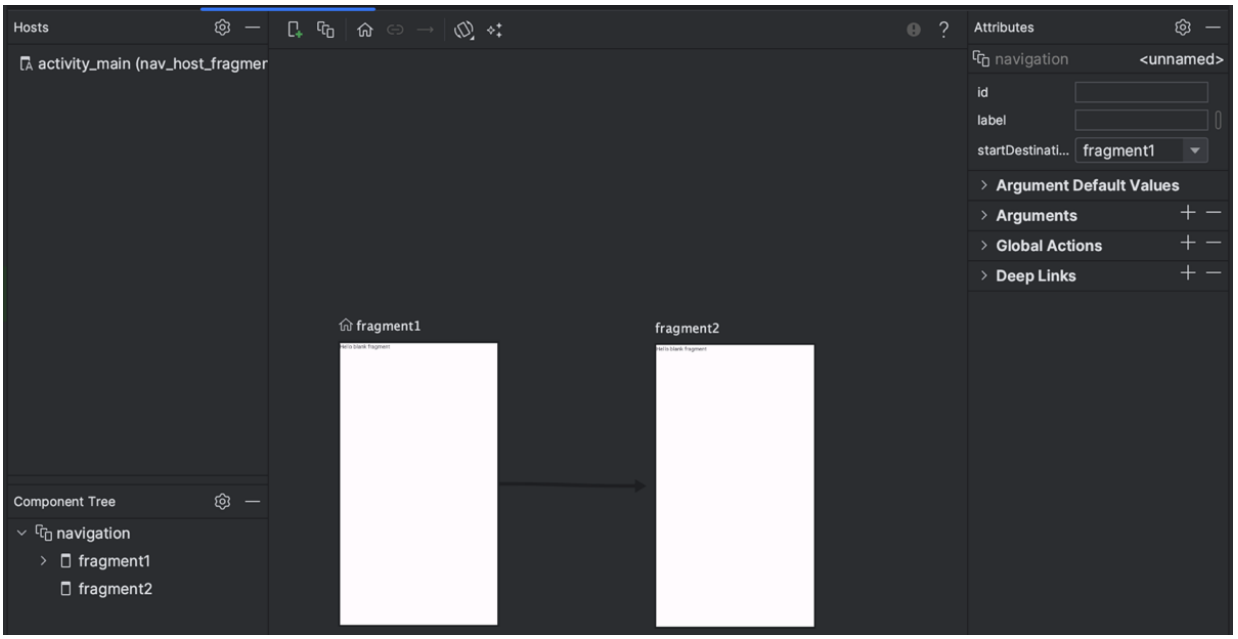

Рисунок 1.2. Настроенная навигация в nav_graph.xml

Вот и все! Теперь вы можете осуществлять навигацию между фрагментами с использованием Navigation API. API автоматически обрабатывает транзакции фрагментов и управляет стеком возврата, обеспечивая согласованность навигации.

Задание

1. Реализовать приложение согласно выбранной тематике, состоящее из трех фрагментов. Фрагменты должны иметь разное наполнение, а также минимальный функционал для возможности их идентификации по внешнему виду.
2. Реализовать навигацию между созданными фрагментами двумя способами:
    a. Ручное управление транзакциями фрагментов;
    b. Navigation API.
3. Обе реализации навигации должны иметь возможность возвращений к предыдущему фрагменту.