Отчет по Android Studio

Содержание

[Практическая работа №1. Использование контейнеров Column и Row 2](#_Toc164284445)

[Практическая работа №2. Card, Box, Image 5](#_Toc164284446)

[Практическая работа №3. Модификаторы 7](#_Toc164284447)

[Практическая работа №4. Взаимодействие с кодом Kotlin и состояния 10](#_Toc164284448)

[Практическая работа №5. LazyColumn и LazyRow 12](#_Toc164284449)

[Практическая работа №6. Компонент Text 14](#_Toc164284450)

[Практическая работа №7. Поля ввода 18](#_Toc164284451)

[Практическая работа №8. Кнопки JetPack Compose 19](#_Toc164284452)

[Практическая работа №9. Scaffold и его друзья 22](#_Toc164284453)

[Практическая работа №11. Навигация в Jetpack Compose 24](#_Toc164284454)

# Практическая работа №1. Использование контейнеров Column и Row

Column позволяет размещать объекты друг под другом. Column можно задавать цвет, ширину, размер, позицию и так далее. (Рисунок 1, 2)



Рисунок 1 – Пример использования Column



Рисунок 2 – Кодовая часть контейнера Column

Row позволяет размещать объекты в одну горизонтальную линию. Им также можно задать цвет, размер и др. (Рисунок 3, 4)



Рисунок 3 – Пример использования Row



Рисунок 4 – Кодовая часть контейнера Row

Язык допускает комбинацию Column и Row, что позволяет размещать элементы в сетке таблицы. (Рисунок 5, 6)

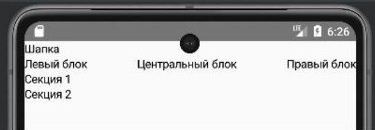


Рисунок 5 – Пример комбинации Column и Row

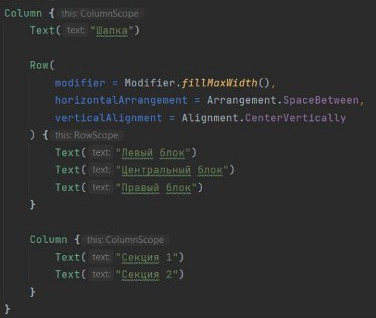


Рисунок 6 – Кодовая часть комбинации Column и Row

Также можно использовать модификаторы для различной компоновки и выравнивания элементов. (Рисунок 7)

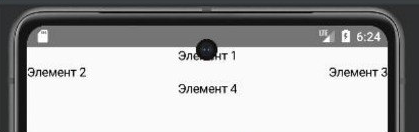


Рисунок 7 – Комбинация Column и Row с использованием модификаторов

# Практическая работа №2. Card, Box, Image

Card представляет собой контейнер, обычно используемый для размещения других элементов и создания карточек в пользовательском интерфейсе. (Рисунок 8, 9)

Box представляет собой простой контейнер для расположения и наложения дочерних элементов в определенном порядке. (Рисунок 10, 11)

Image представляет собой элемент для отображения изображений. (Рисунок 12, 13)

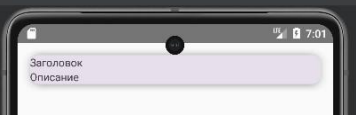


Рисунок 8 – Пример использования Card



Рисунок 9 – Кодовая часть Card

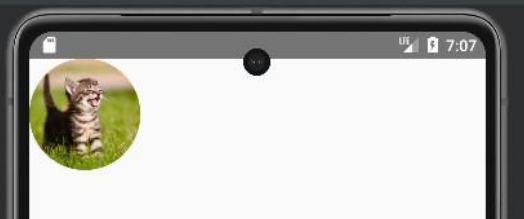


Рисунок 10 – Пример использования Image

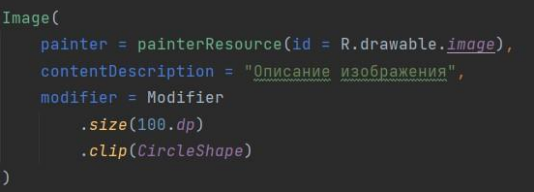


Рисунок 11 – Кодовая часть Image



Рисунок 12 – Пример использования Box



Рисунок 13 – Кодовая часть Box

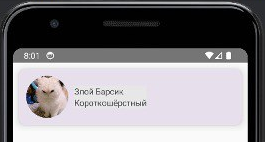


Рисунок 14 – Пример комбинации Card, Box и Image

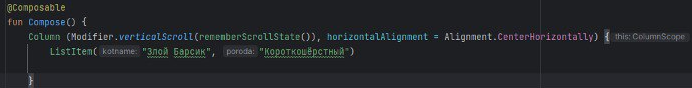


Рисунок 15 – Кодовая часть комбинации Card, Box и Image

# Практическая работа №3. Модификаторы

Модификаторы в Android представляют собой инструменты, позволяющие программистам изменять внешний вид и поведение элементов на экране. Они предоставляют гибкость в настройке компонентов пользовательского интерфейса, делая его более интуитивным и функциональным.

Модификатор offset позволяет менять координаты относительно исходного положения.

Модификатор clickable предоставляет возможность отслеживать и обрабатывать нажатия на элементы пользовательского интерфейса.

Модификатор verticalScroll добавляет возможность прокрутки вертикального содержимого экрана. (Рисунок 16, 17)

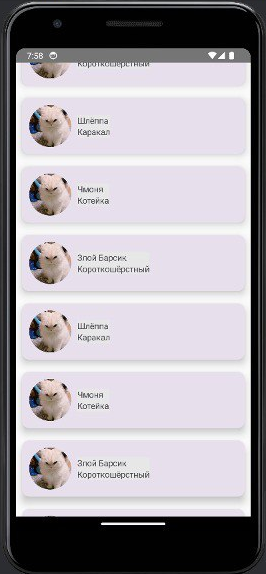


Рисунок 16 – Пример вертикальной прокрутки

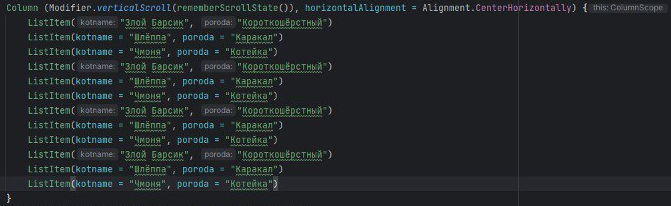


Рисунок 17 – Кодовая часть вертикальной прокрутки

Аналогично мы можем установить горизонтальную прокрутку с помощью модификатора horizontalScroll. (Рисунок 18, 19)

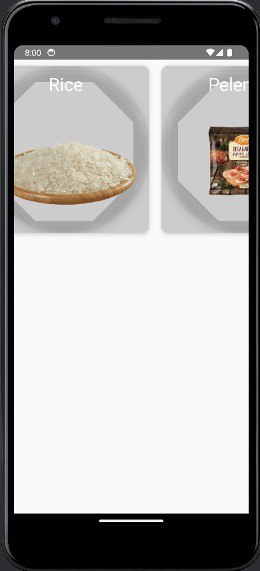


Рисунок 18 – Пример горизонтальной прокрутки

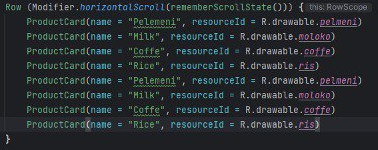


Рисунок 19 – Кодовая часть горизонтальной прокрутки

# Практическая работа №4. Взаимодействие с кодом Kotlin и состояния

В Jetpack Compose можно легко использовать условные конструкции внутри компонентов для динамического определения содержимого. (Рисунок 20)



Рисунок 20 – Пример использования условных конструкций if, else

В Jetpack Compose циклы можно использовать для создания серии однотипных компонентов, что делает удобным определение различающихся данных в виде списка и последующее создание компонентов в цикле. (Рисунок 21)



Рисунок 21 – Пример использования цикла for

В Jetpack Compose вы можете определять функции и использовать их в компонентах для динамического создания содержимого. (Рисунок 22)



Рисунок 22 – Пример определения функций и их последующее использование

В Jetpack Compose функция remember используется для сохранения объекта в памяти, как изменяемого (mutable), так и неизменяемого (immutable). Этот объект сохраняется при начальном построении интерфейса и продолжает храниться во время обновлений интерфейса. (Рисунок 23)

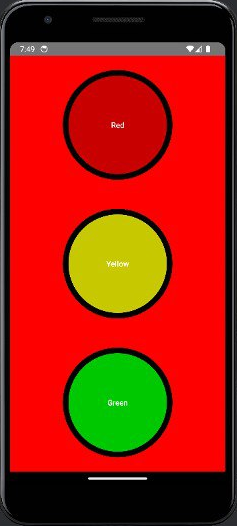


Рисунок 23 – Готовый светофор

# Практическая работа №5. LazyColumn и LazyRow

LazyColumn в Jetpack Compose это эффективный контейнер для отображения списков, который применяет ленивую загрузку элементов. LazyRow Аналогичным образом и отличается так же как Row отличается от Column. Для удобства наполнения LazyRow мы можем использовать модели, представленные через data class. (Рисунок 24)

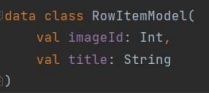


Рисунок 24 – Пример использования RowItemModel

Теперь мы можем создать шаблон для элемента нашего списка. Аналогично создадим пустой файл в папке проекта, назовем его RowItem, и в нем определим @Composable функцию RowItem, которая в качестве аргументы принимает нашу модель RowItemModel. (Рисунок 25)



Рисунок 25 – Пример готовой функции RowItem

Теперь, в MainActivity, мы можем создать LazyRow. (Рисунок 26)



Рисунок 26 – Пример готовой функции LazyRow

# Практическая работа №6. Компонент Text

Один из основных компонентов для работы с текстом - Text. Размер шрифта в Jetpack Compose определяется с использованием параметра fontSize. (Рисунок 27, 28)

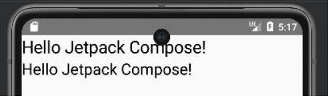


Рисунок 27 – Пример использования fontSize

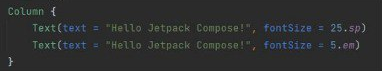


Рисунок 28 – Кодовая часть fontSize

Параметр color в Jetpack Compose определяет цвет шрифта и представляет объект Color. (Рисунок 29)

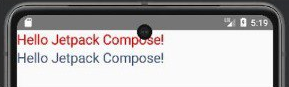


Рисунок 29 – Пример изменения цвета шрифта Color

Для изменения начертания текста необходимо установить параметр fontStyle (Рисунок 30, 31)

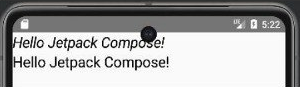


Рисунок 30 – Пример использования fontStyle для изменения начертания текста

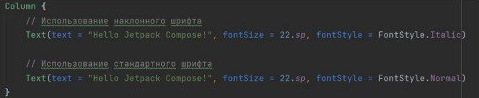


Рисунок 31 – Кодовая часть fontStyle

Параметр fontWeight в Jetpack Compose определяет толщину шрифта и представляет класс FontWeight. (Рисунок 32, 33)

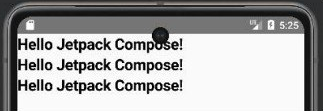


Рисунок 32 – Пример использования параметра fontWeight

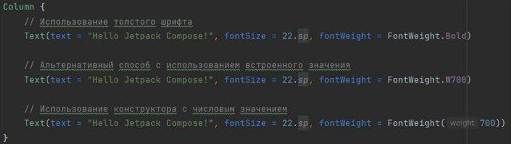


Рисунок 33 – Кодовая часть fontWeight

Параметр fontFamily в Jetpack Compose определяет тип или семейство шрифта и представляет объект FontFamily. (Рисунок 34, 35)



Рисунок 34 – Пример использования параметра fontFamily

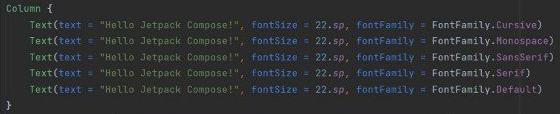


Рисунок 35 – Кодовая часть fontFamily

Параметр letterSpacing в Jetpack Compose задает расстояние между символами и представляет класс TextUnit. (Рисунок 36, 37)

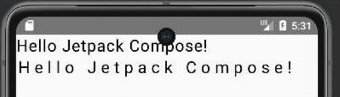


Рисунок 36 – Пример использования параметра letterSpacing



Рисунок 37 – Кодовая часть letterSpacing

Параметр textDecoration в Jetpack Compose позволяет задать декорации для текста. (Рисунок 38, 39)

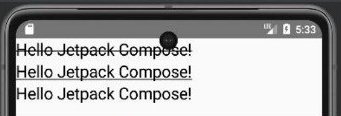


Рисунок 38 – Пример использования параметра textDecoration

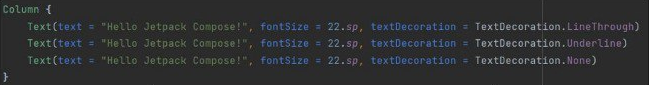


Рисунок 39 – Кодовая часть textDecoration

Параметр textAlign в Jetpack Compose управляет выравниванием текста и представляет объект класса TextAlign. (Рисунок 40, 41)

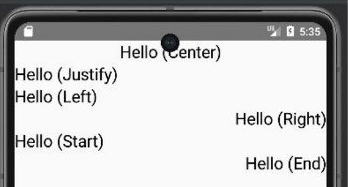


Рисунок 40 – Пример использования параметра textAlign

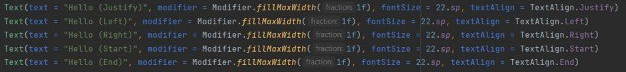


Рисунок 41 – Кодовая часть textAlign

# Практическая работа №7. Поля ввода

TextField - компонент, предназначенный для ввода текста в приложении. (Рисунок 42)



Рисунок 42 – Пример использования TextField

Параметр onValueChange принимает функцию обработки ввода текста. (Рисунок 43)

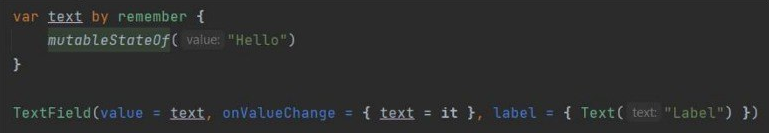


Рисунок 43 – Пример использования onValueChange для TextField

Параметр leadingIcon задает иконку перед текстом, а параметр trailingIcon - иконку после текста. (Рисунок 44, 45)

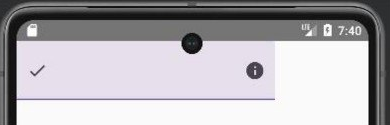


Рисунок 44 – Пример использования leadingIcon и trailingIcon



Рисунок 45 – Кодовая часть leadingIcon и trailingIcon

# Практическая работа №8. Кнопки JetPack Compose

Для создания кнопок в Jetpack Compose применяется компонент Button. (Рисунок 46)

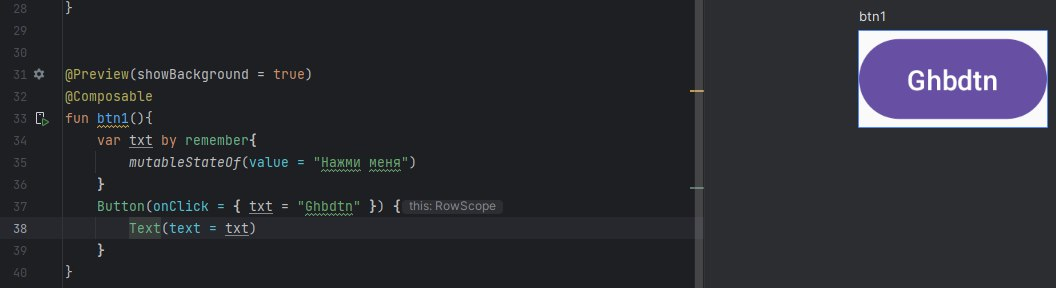


Рисунок 46 – Пример использования компонента Button

Кнопку можно использовать для размещения нескольких элементов внутри нее. (Рисунок 47)

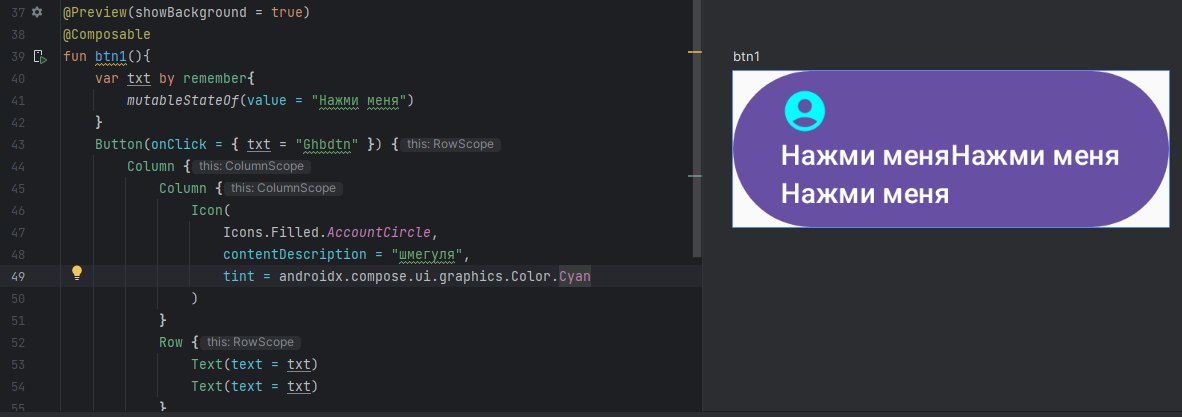


Рисунок 47 – Пример размещения нескольких элементов внутри Button

Компонент OutlinedButton также представляет кнопку и имеет тот же набор параметров. Он работает похожим образом. Главное отличие от стандартных кнопок - немного иная стилизация, которая по умолчанию добавляет ярко выраженную границу компонента и применяет иную цветовую гамму. (Рисунок 48)

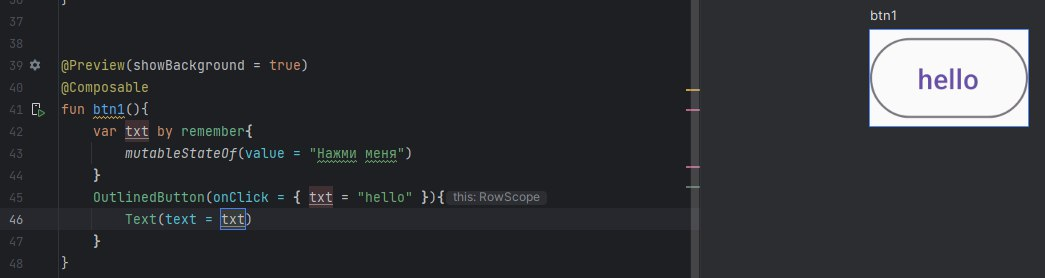


Рисунок 48 – Пример использования OutlinedButton

Компонент IconButton представляет кликабельную иконку, по нажатию на которую можно выполнить некоторые действия. (Рисунок 49)

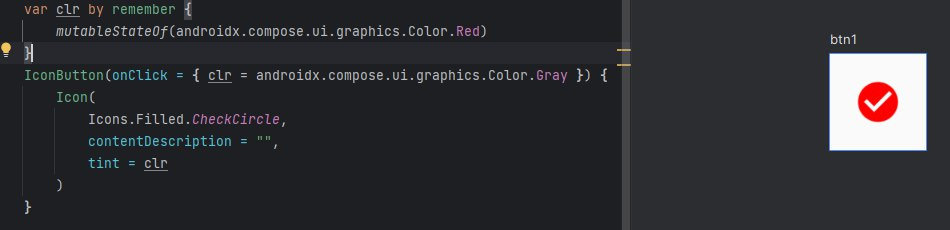


Рисунок 49 – Пример использования IconButton

Компонент FloatingActionButton представляет кнопку, которая обычно располагается внизу экрана поверх основного содержимого и приглашает к выполнению некоторого действия. (Рисунок 50)

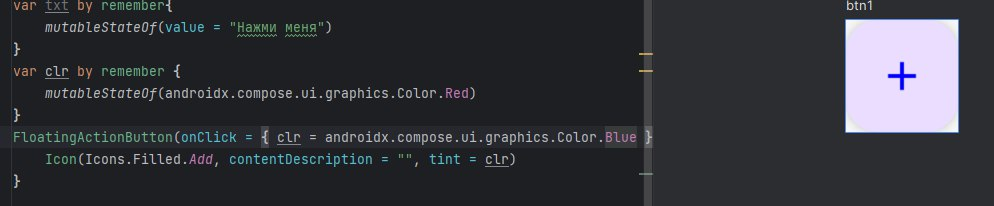


Рисунок 50 – Пример использования FloatingActionButton

Кнопка имеет множество функций компонента, например, Enabled, который устанавливает, доступна ли кнопка для нажатия. (Рисунок 51)

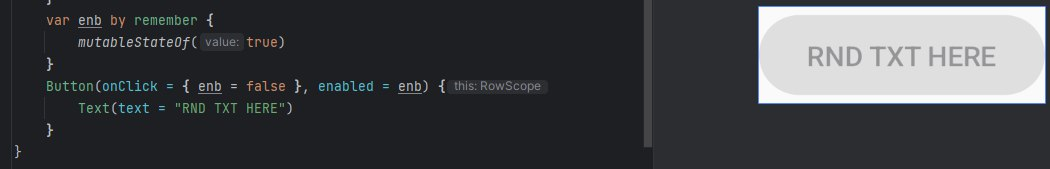


Рисунок 51 – Пример использования функции Enabled

# Практическая работа №9. Scaffold и его друзья

Scaffold в Jetpack Compose представляет собой фундаментальную структуру для построения сложных пользовательских интерфейсов в соответствии с рекомендациями Material Design. Он объединяет различные части пользовательского интерфейса, такие как верхняя и нижняя панели приложения, а также кнопка плавающего действия.

TopAppBar: Верхняя панель приложения, содержащая заголовок «Top app bar».

BottomAppBar: Нижняя панель приложения, содержащая текст «Bottom app bar».

FloatingActionButton: Кнопка плавающего действия, с иконкой добавления.

Content: Внутренний контент Scaffold, представленный текстовым блоком. Он передает innerPadding, которую затем можно использовать в дочерних составных объектах. (Рисунок 52, 53, 54)



Рисунок 52 – Пример использования TopAppBar, BottomAppBar и FloatingActionButton



Рисунок 53 – Пример реализации TopAppBar

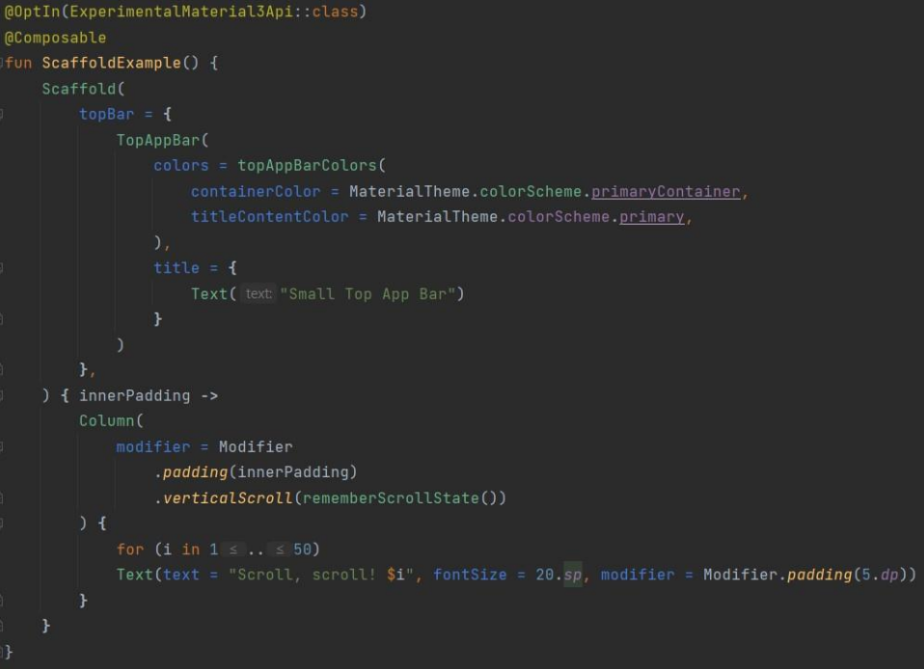


Рисунок 54 – Кодовая часть TopAppBar

# Практическая работа №11. Навигация в Jetpack Compose

Для использования для navController для смены экрана нам нужно воспользоваться NavHost, у которой есть два параметра:

* navController – используется для смены экрана.
* startDestination – используется для привязки определенного экрана по умолчанию.

Метод для отображения наших экранов называется «composable». Сами экраны следует создавать в отдельных файлах с конкретными именами для каждого экрана. Экраны создаем в месте, где по умолчанию создан ваш главный экран MainActivity.

При переходе на следующий экран, новый экран накладывается на предыдущий. Чтобы такого не было у свойства navigate есть свое свойство «popUpTo».

Теперь у нас в стеке храниться два одинаковых экрана. Первый вызвался изначально, при открытии приложения, второй при переходе с экрана. Эту проблемы может решить свойство inclusive свойства popUpTo. Оно удаляет из стека все повторяющиеся экраны оставляя один экземпляр. (Рисунок 55, 56, 57)

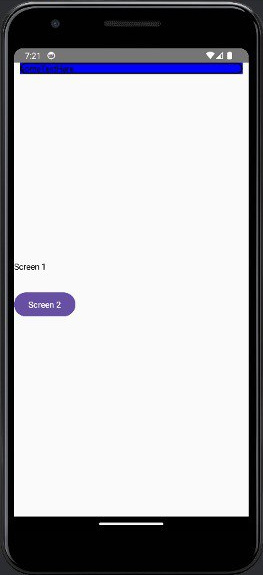


Рисунок 55 – Пример реализации StartDestination

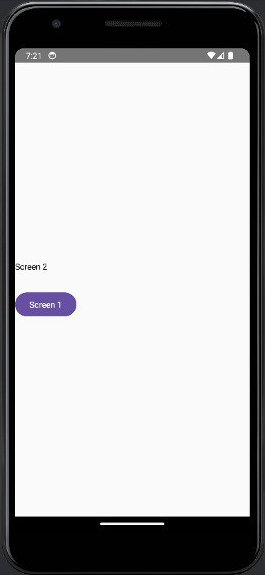


Рисунок 56 – Пример реализации navController

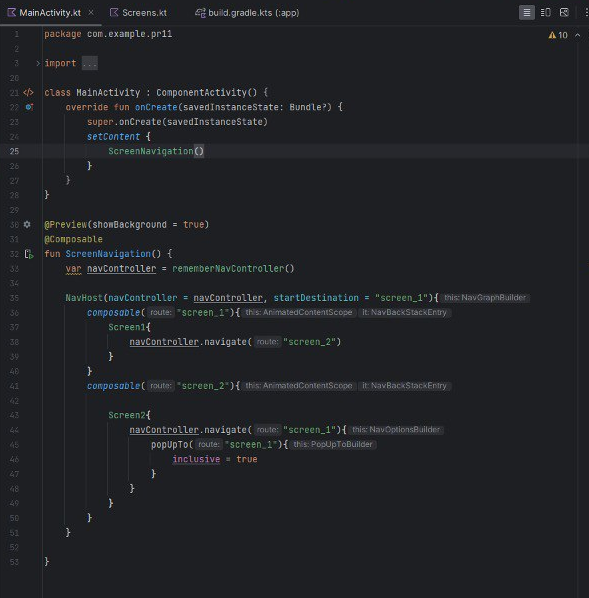


Рисунок 57 – Кодовая часть реализации навигации