Министерство образования Самарской области

ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ САМАРСКОЙ ОБЛАСТИ «ПОВОЛЖСКИЙ ГОСУДАРСТВЕННЫЙ КОЛЛЕДЖ»

Составитель: Лазарев Т.А.

МЕТОДИЧЕСКОЕ ПОСОБИЕ № 3 ДЛЯ ОПЦ.В.16 ОСНОВЫ РАЗРАБОТКИ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ

«профессиональный цикл»

программы подготовки специалистов среднего звена

09.02.07 Информационные системы и программирование

Самара, 2025

Методическое пособие для второй практической работы

Содержание

Начало работы

Создайте новый проект и выберите Empty Activity (рис. 1)

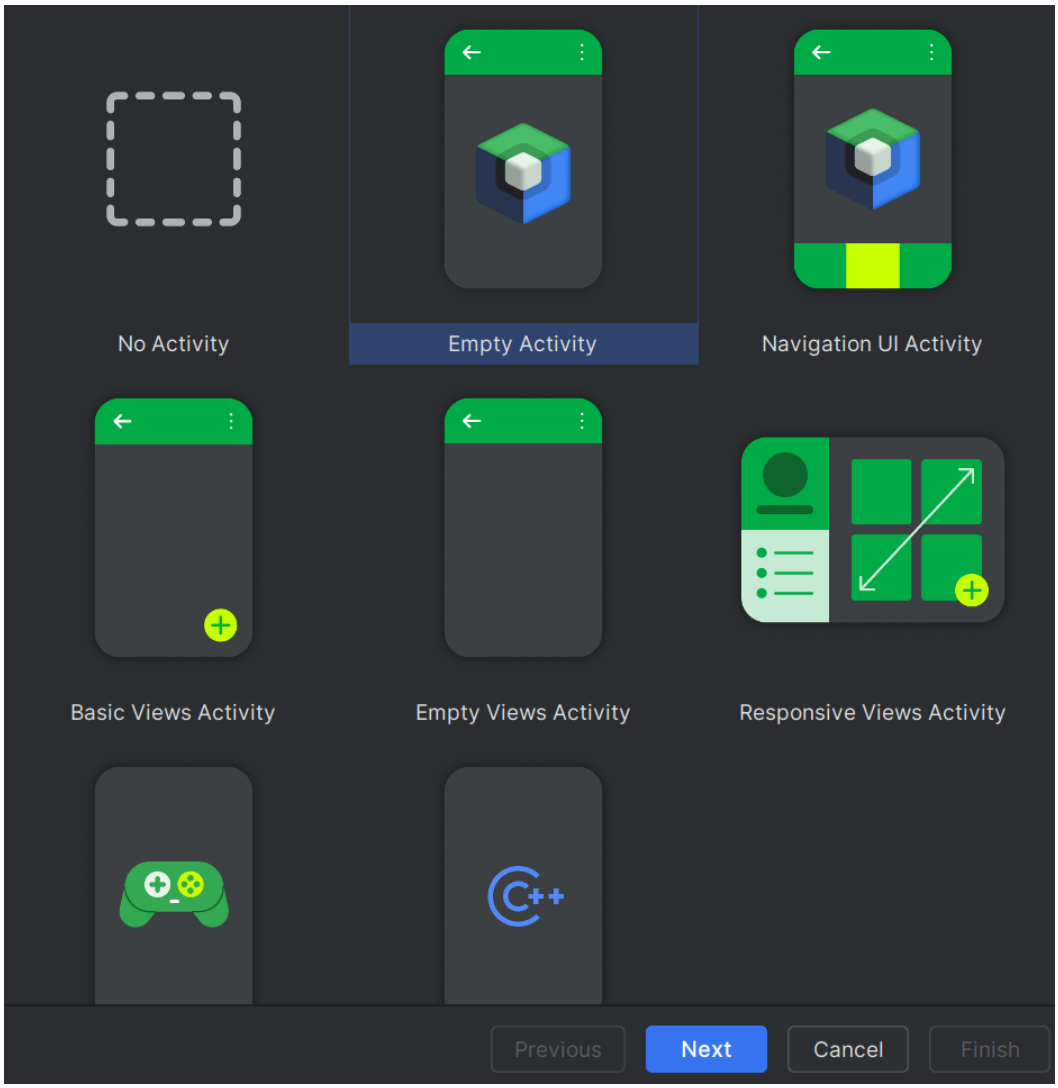

Рисунок 1 – Выбор приложения

После создания проекта, будет создана активити с подобным содержанием

Рисунок 2 – Содержание Activity в Compose

Внутри setContent находится код вашего приложения, который будет выполняться при его запуске.

Для предпросмотра в Jetpack compose, нужно создать отдельную Composable функцию и использовать аннотацию @Preview (рис. 3)

Рисунок 3 – Просматриваемый элемент

Тестирование логики работы макета можно проводить, используя интерактивный режим. Для этого надо найти предпросмотр и нажать на троеточие. Далее, выбрать Start Interactive Mode (рис. 4):

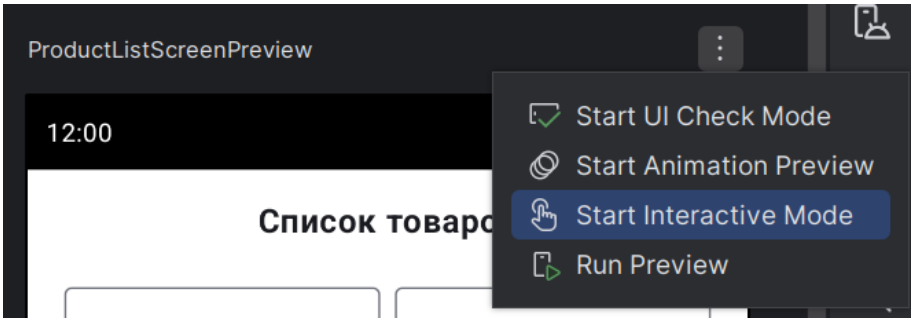

Рисунок 4 – Интерактивный режим

Парселизация и состояние

Обеспечение сохранения состояния «переменных» после переворота экрана происходит через rememberSaveable (рис. 5):

```
@Composable
fun ProductListScreen() {
    val products = rememberSaveable { mutableStateListOf(
        ProductItem( name = "Хлеб",  quantity = "12 шт.",  article = "XXA223456",  discount = "50%"),
        ProductItem( name = "Колбаса",  quantity = "5 шт.",  article = "XXA22322",  discount = "0%")
    ) }

    var productName by rememberSaveable { mutableStateOf( value = "") }
    var productQuantity by rememberSaveable { mutableStateOf( value = "") }
    var productArticle by rememberSaveable { mutableStateOf( value = "") }
    var productDiscount by rememberSaveable { mutableStateOf( value = "") }
```

Рисунок 5 – Использование rememberSaveable

Для использования класса ProductItem вместе с rememberSaveable, требуется добавить к нему соответствующий аттрибут @Parcelize и подключить интерфейс Parcelable (рис. 6).

```
@Parcelize
data class ProductItem(
    val name: String,
    val quantity: String,
    val article: String,
    val discount: String
): Parcelable
```

Рисунок 6 – Parcelable class

Подключение Parcelize происходит через подключение соответствующего плагина (рис. 7), а именно id(«kotlin-parcelize»)

```
plugins {
    alias(libs.plugins.android.application)
    alias(libs.plugins.kotlin.android)
    alias(libs.plugins.kotlin.compose)
    id("kotlin-parcelize")
}
```

Рисунок 7 – Подключение плагина

После подключения можно будет подключить аттрибут @Parcelize.

Виджеты Jetpack Compose

Если инструкция становится красной и под ней появляется белое подчеркивание при попытке навести курсор, то это означает, что её нужно импортировать, а именно нужно нажать на Import function или комбинацию клавиш Alt+Shift+Enter (рис. 8).

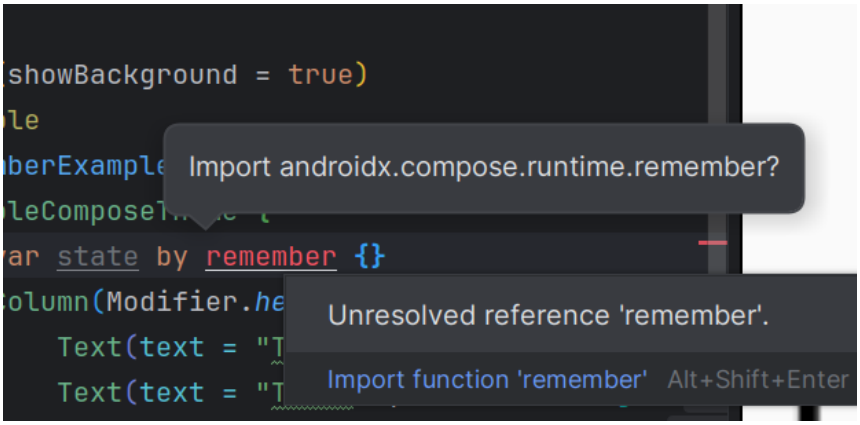

Рисунок 8 – Не разрешенная зависимость

Существуют следующие правила расположения виджетов. Элемент Row распределяет все свои элементы в строку (рис. 9):

```
@Preview(showBackground = true)
@Composable
fun RowExample() {
    ExampleComposeTheme {
        Row {
            Text(text = "Текст")
            Text(text = "Текст2")
            Text(text = "Текст3")
        }
    }
}
```

Рисунок 9 – Composable Row

Также можно распеделять элементы по колонке (рис. 10):

```
@Preview(showBackground = true)
@Composable
fun ColumnExample() {
    ExampleComposeTheme {
        Column(Modifier.height( height = 76.dp)) {
            Text(text = "Текст", Modifier.weight( weight = 1f))
            Text(text = "Текст2", Modifier.weight( weight = 2f))
            Text(text = "Текст", Modifier.weight( weight = 1f))
        }
    }
}
```

Рисунок 10 – Composable Column

К каждому Composable можно применять модификаторы, которые регулируют отступы, размеры, веса, границу и другие стилевые аттрибуты. В примере выше используются модификаторы высоты height и модификаторы веса weight, который задает пропорцию занимаемого пространства внутри Column.

Отрисовка компонентов выглядит следующим образом (рис. 11):

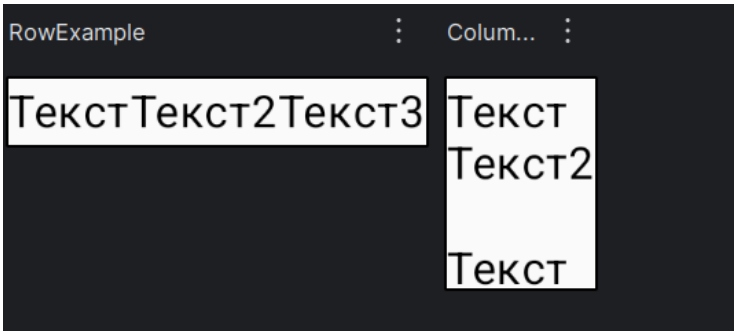

Рисунок 11 – Расположение элементов

Подробнее про разметку можно узнать в следующей статье: https://developer.android.com/develop/ui/compose/layouts/basics.

С модификаторами по подробнее можно ознакомиться в следующей статье: https://developer.android.com/develop/ui/compose/modifiers.

Следующий пример демонстрирует базовую работу с состоянием (рис. 12)

```
@Preview(showBackground = true)
@Composable
fun RememberExample() {
    ExampleComposeTheme {
        var state by remember { mutableStateOf( value = 0) }
        Column {
            Text( text = "Кнопка была нажата $state раз")
            Button(onClick = {state++}, Modifier.fillMaxWidth(), ) { Text( text = "Это кнопка")}
        }
    }
}
```

Рисунок 12 – Компонент счетчика

Нажатие на кнопку увеличивает значение состояния «переменной» state на единицу. Каждое изменение переменной вызывает рекомпозицию.

Подробнее с работой с состоянием можно ознакомиться в статье: https://developer.android.com/develop/ui/compose/state.

Следующий пример демонстрирует обертку над Box. Box это такой компонент (рис. 13), позволяющий дочерним компонентам распологаться друг над другом.

```
3 Usages
@Composable
fun EmptyBox(modifier: Modifier = Modifier,
             content: @Composable (BoxScope.() -> Unit) = {} ) {
        Box(content = content, modifier = modifier
            .border(border=BorderStroke( width = 1.dp, Color.Black),
                shape = RoundedCornerShape( size = 10.dp)).padding( all = 6.dp)
        )
}
```

Рисунок 13 – Обертка над Box

Composable EmptyBox – это компонент который оборачивает Box и добавляет к нему дополнительные стили. Компонент предусматривает наполнение (content), представляющие из себя лямбда выражение с выраженным контекстом BoxScope. Без указания контекста, Compose не позволит компонентам внутри наполнения выравнивать себя относительно

родительского компонента. Наполнение по конвенции всегдя является последним аргументом, так как, таким образом компилятор котлина может вынести лямблу за скобки аргументов. Примером такой лямбды являются пурпурные скобки (см. рис. 12).

Пример выравнивания, через Modifier.align() выглядит следующим образом (рис. 14):

```
@Preview(showBackground = true)
@Composable
fun BlockComposeExample() {
        ExampleComposeTheme {
            Column{
                EmptyBox(modifier = Modifier.size( width = 150.dp,  height = 100.dp) ) {
                    Text( text = "Текст 1", modifier = Modifier.align(Alignment.TopEnd))
                    Text( text = "Текст 2", modifier = Modifier.align(Alignment.TopStart))
                }
                EmptyBox(modifier = Modifier.size( width = 100.dp,  height = 100.dp)  ) {
                    Text( text = "Середина", modifier = Modifier.align(Alignment.Center))
                }
            }
        }
}
```

Рисунок 14 – Повторное использование компонентов и выравнивание

Списки, которые можно листать, когда они не умещаются, можно создавать используя компонент LazyColumn (рис. 15):

```
@Preview(showBackground = true)
@Composable
fun ListExample() {
    ExampleComposeTheme {
        val list = remember { mutableListOf("1", "2", "3") }
        LazyColumn(modifier = Modifier.height( height = 120.dp)) {
            items( items = list) { item ->
                EmptyBox(modifier = Modifier.size( width = 150.dp,  height = 100.dp) ) {
                    Text( text = item, modifier = Modifier.align(Alignment.Center))
                }
            }
        }
    }
}
```

Рисунок 15 – Использование списка элементов

Каждый item в примере является текстом из списка list.

Отрисовка примеров выше выглядит следующим образом (рис. 16):

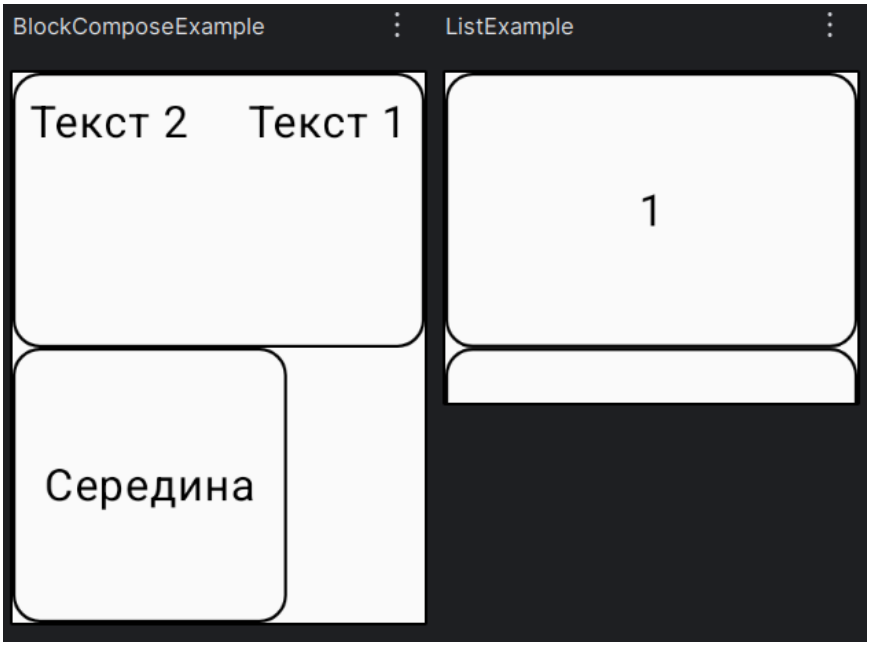

Рисунок 16 – Использование EmptyBox