Визуальные компоненты

Text

Неотъемлимой частью визуального интерфейса является текст. Для отображения текста Jetpack Compose предоставляет ряд встроенных компонентов. Прежде всего это компонент Text, который имеет следующее определение:

```
@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
): @Composable Unit
```

Параметры компонента:

- text: объект String, который представляет выводимый текст
- modifier: объект Modifier, который представляет применяемые к компоненту модификаторы
- color: объект Color, который представляет цвет текста. По умолчанию имеет значение Color.Unspecified
- fontSize: объект TextUnit, который представляет размер шрифта. По умолчанию равен TextUnit.Unspecified
- fontStyle: объект FontStyle?, который представляет стиль шрифта. По умолчанию равен null
- fontWeight: объект FontWeight?, который представляет толщину шрифта. По умолчанию равен null
- fontFamily: объект FontFamily?, который представляет тип шрифта. По умолчанию равен null
- letterSpacing: объект TextUnit, который представляет отступы между символами. По умолчанию равен TextUnit.Unspecified
- textDecoration: объект TextDecoration?, который представляет тип декораций (например, подчеркивание), применяемых к тексту. По умолчанию равен null
- textAlign: объект TextAlign?, который представляет выравнивание текста. По умолчанию равен null

- lineHeight: объект TextUnit, который представляет высоту строки текста. По умолчанию равен TextUnit.Unspecified
- overflow: объект TextOverflow, который определяет поведение текста при его выходе за границы контейнера. По умолчанию равен TextOverflow.Clip
- softWrap: объект Boolean, который определяет, должен ли текст переносится при завершении строки. При значении false текст не переносится, как будто строка имеет бесконечную длину. По умолчанию равен true
- maxLines: объект Int, который представляет максимальное количество строк. Если текст превысил установленное количество строк, то он усекается в соответствии с параметрами overflow и softWrap. По умолчанию равен Int.MAX_VALUE
- onTextLayout: объект (TextLayoutResult) -> Unit, который представляет функцию, выполняемую при определении компоновки текста.
- style: объект TextStyle, который представляет стиль текста. Значение по умолчанию - LocalTextStyle.current

Размер шрифта

Размер шрифта определяется параметром fontSize. В качестве параметру может передаваться значение типов Int, Double и Float, после которых указывается тип единиц. Это могут быть масштабируемые пиксели (единицы sp, например, 22.sp), либо это может быть относительный размер шрифта в единицах em (например, 18.em). Значение TextUnit.Unspecified указывает, что высота шрифта наследуется от настроек родительного компонента

Простейшее применение компонента:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.Text
import androidx.compose.foundation.layout.Column
import androidx.compose.ui.unit.em
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Column {
                Text(text = "Hello Jetpack Compose!", fontSize=25.sp)
                Text(text = "Hello Jetpack Compose!", fontSize=5.em)
            }
        }
    }
}
```

Цвет шрифта

За определение цвета шрифта отвечает параметр color, который представляет объект Color, ранее рассмотренный в статье Установка цвета.

Цвет шрифта

За определение цвета шрифта отвечает параметр color, который представляет объект Color, ранее рассмотренный в статье Установка цвета.

```
import androidx.compose.ui.graphics.Color
//..........
Column {
    Text(text = "Hello Jetpack Compose!",  fontSize=22.sp, color=Color.Red)
    Text(text = "Hello Jetpack Compose!",  fontSize=22.sp,
        color= Color(red = 0x44, green = 0x55, blue = 0x88, alpha = 0xFF))
}
```

Стиль шрифта

Стиль шрифта определяется параметром fontStyle, который представляет класс FontStyle?. Для определения этот класс предоставляет два встроенных значения:

- FontStyle.Italic (наклоннный шрифт)
- FontStyle.Normal (стандартный шрифт)

```
import androidx.compose.ui.text.font.FontStyle
//..........
Text(text = "Hello Jetpack Compose!",  fontSize=22.sp, fontStyle =
FontStyle.Italic)
Text(text = "Hello Jetpack Compose!",  fontSize=22.sp, fontStyle =
FontStyle.Normal)
```

Толщина шрифта

Толщина шрифта задается параметром fontWeight, который представляет класс FontWeight.

Есть два способа установки толщины шрифта. Прежде всего можно использовать конструктор этого класса, в который передается числовое значение от 1 до 1000. Чем больше значение, тем толще будет шрифт:

```
FontWeight(600)
```

Второй способ заключается в применении встроенных значений:

- FontWeight.Black (Эквивалентно значению W900)
- FontWeight.Bold (Эквивалентно значению W700)
- FontWeight.ExtraBold (Эквивалентно значению W800)
- FontWeight.ExtraLight (Эквивалентно значению W200)
- FontWeight.Light (Эквивалентно значению W300)
- FontWeight.Medium (Эквивалентно значению W500)
- FontWeight.Normal (Эквивалентно W400 - значение по умолчанию)
- FontWeight.SemiBold (Эквивалентно значению W600)
- FontWeight.Thin (Эквивалентно значению W100

Так, следующие определения компонента Text будут аналогичны:

```
import androidx.compose.ui.text.font.FontWeight
//..........
Text(text = "Hello Jetpack Compose!",  fontSize=22.sp, fontWeight=
FontWeight.Bold)
Text(text = "Hello Jetpack Compose!",  fontSize=22.sp, fontWeight=
FontWeight.W700)
Text(text = "Hello Jetpack Compose!",  fontSize=22.sp, fontWeight=
FontWeight(700))
```

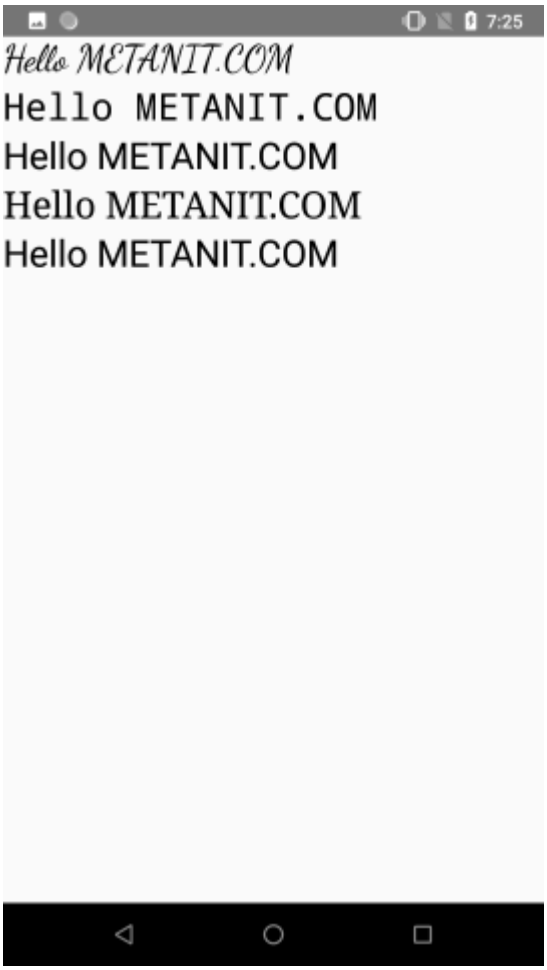
Тип шрифта Тип или семейство шрифта определяется параметром fontFamily, который представляет объект FontFamily?

Для определения шрифта FontFamily предоставляет ряд встроенных констант:

- FontFamily.Cursive (курсивный, рукописный шрифт)
- FontFamily.Monospace
- FontFamily.Serif
- FontFamily.SansSerif
- FontFamily.Default (шрифт по умолчанию на текущей платформе)
- FontFamily.SansSerif

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Column
import androidx.compose.material.Text
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.unit.sp
```

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Column {
                Text(text = "Hello Jetpack Compose!",  fontSize=22.sp,
fontFamily=FontFamily.Cursive)
                Text(text = "Hello Jetpack Compose!",  fontSize=22.sp,
fontFamily=FontFamily.Monospace)
                Text(text = "Hello Jetpack Compose!",  fontSize=22.sp,
fontFamily=FontFamily.SansSerif)
                Text(text = "Hello Jetpack Compose!",  fontSize=22.sp,
fontFamily=FontFamily.Serif)
                Text(text = "Hello Jetpack Compose!",  fontSize=22.sp,
fontFamily=FontFamily.Default)
            }
        }
    }
}
```



Расстояния между символами

Параметр letterSpacing задает расстояние между символами и представляет класс TextUnit. В данном случае мы можем установить расстояние, так как и размер шрифта, с помощью единиц sp или em:

```
Text(text = "Hello Jetpack Compose!",  fontSize=22.sp, letterSpacing= 1.3.sp)
Text(text = "Hello Jetpack Compose!",  fontSize=22.sp, letterSpacing= 0.3.em)
```

Декорации текста

Параметр textDecoration позволять задать декорации для текста. Данный параметр принимает объект класса TextDecoration, который предоставляет несколько встроенных значений:

- TextDecoration.LineThrough (зачеркивает текст)
- TextDecoration.Underline (подчеркивает текст)
- TextDecoration.None (отсутствие декораций)

```
import androidx.compose.ui.text.style.TextDecoration
//...........
Text(text = "Hello Jetpack Compose!",  fontSize=22.sp, textDecoration =
TextDecoration.LineThrough)
Text(text = "Hello Jetpack Compose!",  fontSize=22.sp, textDecoration =
TextDecoration.Underline)
Text(text = "Hello Jetpack Compose!",  fontSize=22.sp, textDecoration =
TextDecoration.None)
```

Выравнивание текста

Параметр textAlign управляет выравниванием текста и представляет объект класса TextAlign. В качестве значения этому параметру можно передать значение одного из свойств класса TextAlign:

- TextAlign.Center: выравнивание текста по центру контейнера
- TextAlign.Justify: текст равномерно растягивается по всей ширине контейнера
- TextAlign.End: выравнивание текста по конечному краю контейнера (в зависимости от ориентации текста это может быть левый или правый край)
- TextAlign.Start: выравнивание текста по началу контейнера (в зависимости от ориентации текста это может быть левый или правый край)
- TextAlign.Left: выравнивание текста по левому краю контейнера
- TextAlign.Right: выравнивание текста по правому краю контейнера

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.material.Text
import androidx.compose.ui.Modifier
```

```
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Column {
                Text(text = "Hello (Center)", modifier = Modifier.fillMaxWidth(1f)
,  fontSize=22.sp, textAlign = TextAlign.Center)
                Text(text = "Hello (Justify)", modifier =
Modifier.fillMaxWidth(1f),  fontSize=22.sp, textAlign = TextAlign.Justify)
                Text(text = "Hello (Left)", modifier = Modifier.fillMaxWidth(1f),
fontSize=22.sp, textAlign = TextAlign.Left)
                Text(text = "Hello (Right)", modifier = Modifier.fillMaxWidth(1f),
fontSize=22.sp, textAlign = TextAlign.Right)
                Text(text = "Hello (Start)", modifier = Modifier.fillMaxWidth(1f),
fontSize=22.sp, textAlign = TextAlign.Start)
                Text(text = "Hello (End)", modifier = Modifier.fillMaxWidth(1f),
fontSize=22.sp, textAlign = TextAlign.End)
            }
        }
    }
}
```

Усечение текста

Параметр overflow управляет тем, как будет обрабатываться текст при его выходе за границы контейнера. Этот параметр принимает значение класса TextOverflow. В качестве значения параметру можно передать значение одного из свойств данного класса:

- TextOverflow.Clip: выходящий за границы контейнера текст усекается
- TextOverflow.Ellipsis: текст усекается, а в конце текста добавляется многоточие
- TextOverflow.Visible: весь текст может отображаться

Перенос текста

Параметр softWrap управляет переносом текста. Если он равен true, то текст переносится. Если false, то нет.

Стиль текста

Параметр style управляет стилем текста. Он предоставляет класс TextStyle , который по сути объединяет ряд вышеупомянутых и несколько дополнительных параметров в одну сущность. Его конструктор принимает следующие параметры:

```
TextStyle(
    color: Color,
    fontSize: TextUnit,
    fontWeight: FontWeight?,
    fontStyle: FontStyle?,
    fontSynthesis: FontSynthesis?,
    fontFamily: FontFamily?,
    fontFeatureSettings: String?,
    letterSpacing: TextUnit,
    baselineShift: BaselineShift?,
    textGeometricTransform: TextGeometricTransform?,
    localeList: LocaleList?,
    background: Color,
    textDecoration: TextDecoration?,
    shadow: Shadow?,
    textAlign: TextAlign?,
    textDirection: TextDirection?,
    lineHeight: TextUnit,
    textIndent: TextIndent?
)
```

- color: объект Color, который представляет цвет текста. По умолчанию имеет значение Color.Unspecified

- background: объект Color, который фоновый цвет компонента. По умолчанию имеет значение Color.Unspecified
- fontSize: объект TextUnit, который представляет размер шрифта. По умолчанию равен TextUnit.Unspecified
- fontStyle: объект FontStyle?, который представляет стиль шрифта. По умолчанию равен null
- fontWeight: объект FontWeight?, который представляет толщину шрифта. По умолчанию равен null
- fontFamily: объект FontFamily?, который представляет тип шрифта. По умолчанию равен null
- fontFeatureSettings: объект String?, который определяет, как будут применяться настройки толщины шрифта и его наклон (то есть значения параметров fontWeight и fontStyle), если используемый шрифт не поддерживает выделение жирным и (или) наклон. По умолчанию равен null
- letterSpacing: объект TextUnit, который представляет отступы между символами. По умолчанию равен TextUnit.Unspecified
- baselineShift: объект BaselineShift?, который определяет, насколько текст будет сдвигаться относительно базовой линии (baseline). По умолчанию равен null
- textGeometricTransform: представляет применяемые к тексту геометрические трансформации в виде объекта TextGeometricTransform?. По умолчанию равен null
- localeList: объект LocaleList?, который представляет список со специфичными для егиона символами. По умолчанию равен null
- textDecoration: объект TextDecoration?, который представляет тип декораций (например, подчеркивание), применяемых к тексту. По умолчанию равен null
- textAlign: объект TextAlign?, который представляет выравнивание текста. По умолчанию равен null
- textDirection: объект TextDirection?, который представляет направление текста. По умолчанию равен null
- lineHeight: объект TextUnit, который представляет высоту строки текста. По умолчанию равен TextUnit.Unspecified
- shadow: объект Shadow?, который определяет применяемый к тексту эффект тени. По умолчанию равен null
- textIndent: объект TextIndent?, который представляет отступ от начала текста. По умолчанию равен null

Поскольку большая часть этих параметров применяется непосредственно в функции компонента Text, рассмотрим некоторые параметры, которые отстуствуют в функции компонента Text.

Геометрические трансформации

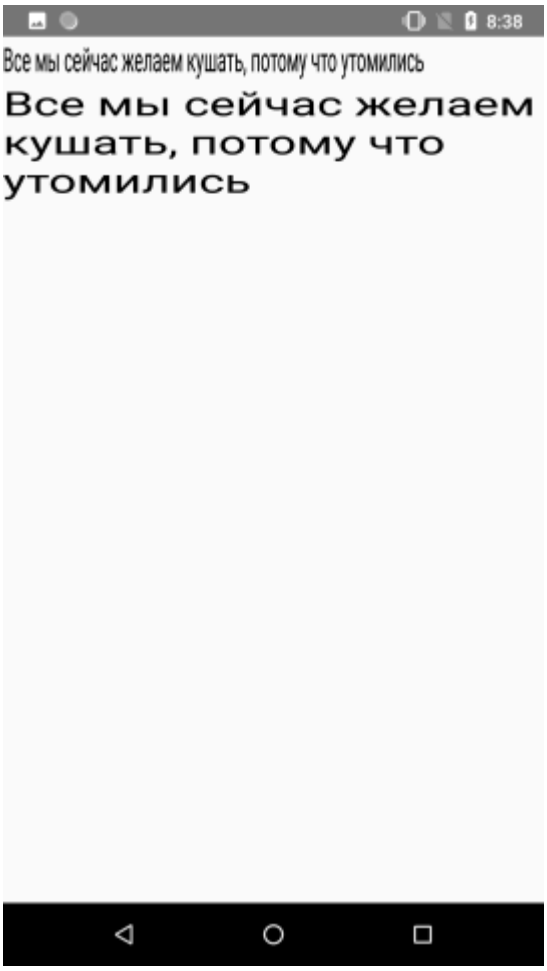
Параметр TextGeometricTransform задает геометрические трансформации текста с помощью объекта TextGeometricTransform:

```
TextGeometricTransform(scaleX: Float = 1.0f, skewX: Float = 0f)
```

Первый параметр - scaleX указывает на увеличение текста. Если значение меньше 1.0f, то текст сжимается, если больше - то увеличивается.

Второй параметр - skewX указывает на сдвиг текста. Например, точка с координатами (x, y), будет трансформирована в точку (x + y * skewX, y). Значение по умолчанию - 0.0f. Например:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Column
import androidx.compose.material.Text
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.style.TextGeometricTransform
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Column {
                Text(
                    text = "Все мы сейчас желаем кушать, потому что утомились",
                    fontSize = 22.sp,
                    style = TextStyle(textGeometricTransform =
TextGeometricTransform(0.5f))
                )
                Text(
                    text = "Все мы сейчас желаем кушать, потому что утомились",
                    fontSize = 22.sp,
                    style = TextStyle(textGeometricTransform =
TextGeometricTransform(1.5f))
                )
            }
        }
    }
}
```



Создание тени для текста

Параметр shadow задает затенение текста с помощью объекта Shadow:

```
Shadow(color: Color, offset: Offset, blurRadius: Float)
```

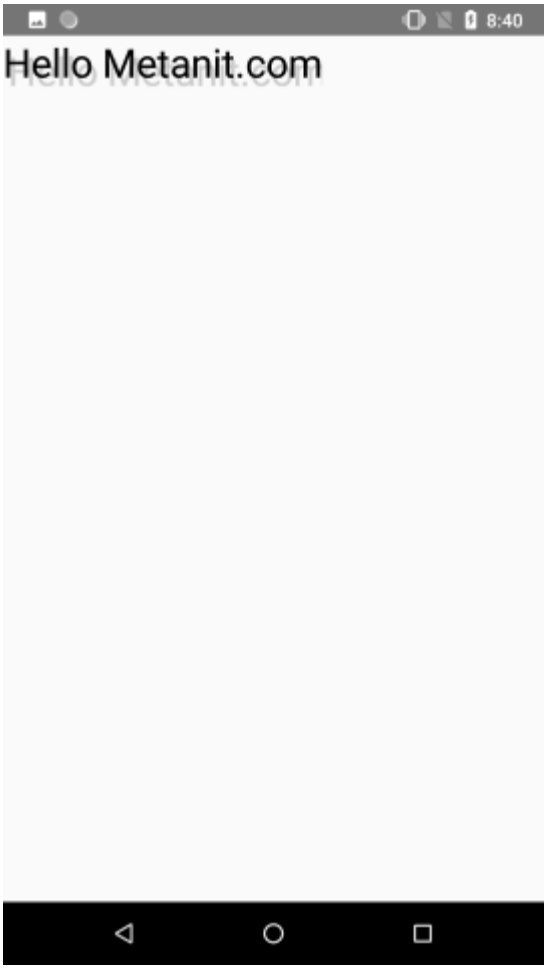
Первый параметр - color устанавливает цвет тени.

Второй параметр - offset смещение тени в виде объекта Offset.

Третий параметр - blurRadius задает радиус размытия.

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Column
import androidx.compose.material.Text
import androidx.compose.ui.geometry.Offset
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.Shadow
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.style.TextGeometricTransform
```

```
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Text(
                text = "Hello Metanit.com",
                fontSize = 30.sp,
                style = TextStyle(shadow = Shadow(Color.LightGray , Offset(10.0f,
16.5f), 1.0f))
            )
        }
    }
}
```



Направление текста

Параметр textDirection устанавливает направление текста и может принимать следующие значения:

- TextDirection.Content: направление текста зависит от первого направляющего символа в соответствии с алгоритмом Unicode Bidirectional Algorithm
- TextDirection.ContentOrLtr: направление текста зависит от первого направляющего символа в соответствии с алгоритмом Unicode Bidirectional Algorithm, либо представляет направление слева

направо

- TextDirection.ContentOrRtl: направление текста зависит от первого направляющего символа в соответствии с алгоритмом Unicode Bidirectional Algorithm, либо представляет направление справа налево
- TextDirection.Ltr: текс направлен слева направо
- TextDirection.Rtl: текст направлен справо налево

Например:

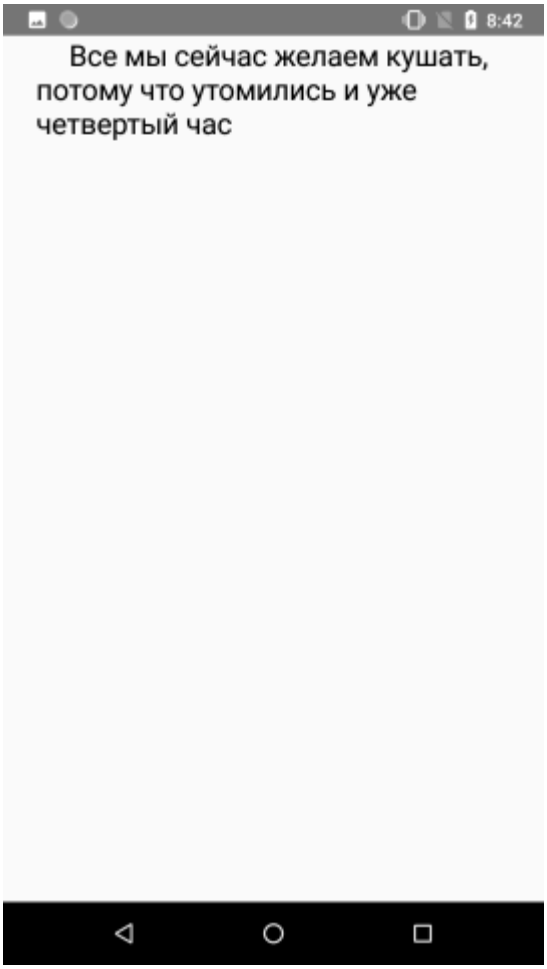
```
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.style.TextDirection
//............
Text(
    text = "Все мы сейчас желаем кушать, потому что утомились",
    fontSize=22.sp,
    style = TextStyle(textDirection = TextDirection.Rtl)
)
```

TextIndent

Параметр textIndent позволяет установить отступ от первого символа в тексте и от остального текста. Этот параметр представляет класс TextIndent, конструктор которого принимает два значения. Первое значение указывает на отступ от первого символа. Второе значение применяется, если текст многострочный и устанавливает отступ от остальных символов на второй и последующих строках. Для установки отступа применяются единицы sp. Например:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Column
import androidx.compose.material.Text
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.style.TextIndent
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Text(
                text = "Все мы сейчас желаем кушать, потому что утомились и уже
четвертый час",
                fontSize = 22.sp,
                style = TextStyle(textIndent = TextIndent(50.sp, 25.sp))
            )
```

```
        }
    }
}
```



Кнопка Button

Для создания кнопок в Jetpack Compose применяется компонент Button, который имеет следующее определение:

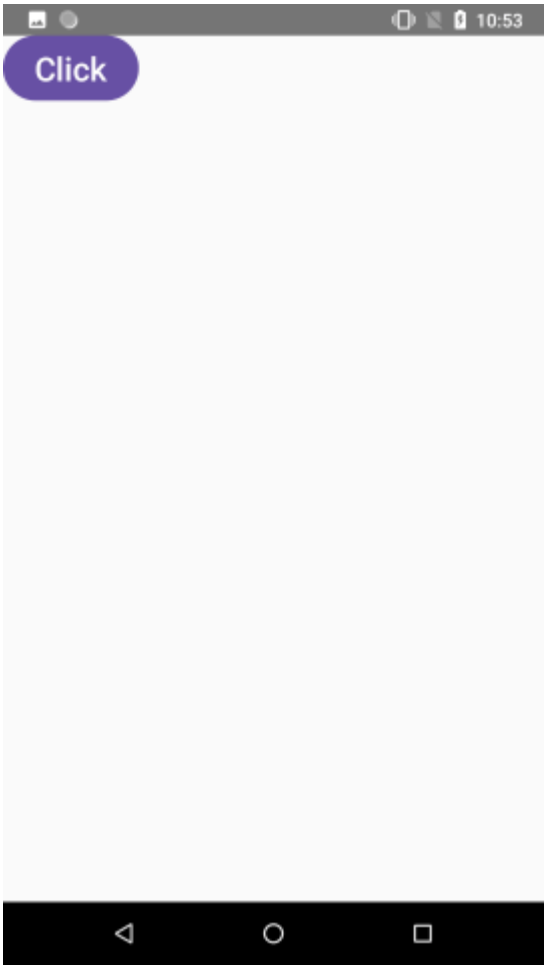
```
@Composable
fun Button(
    onClick: () -> Unit,
    modifier: Modifier = Modifier,
    enabled: Boolean = true,
    interactionSource: MutableInteractionSource = remember {
MutableInteractionSource() },
    elevation: ButtonElevation? = ButtonDefaults.elevation(),
    shape: Shape = MaterialTheme.shapes.small,
    border: BorderStroke? = null,
    colors: ButtonColors = ButtonDefaults.buttonColors(),
    contentPadding: PaddingValues = ButtonDefaults.ContentPadding,
    content: RowScope.() -> Unit
): @Composable Unit
```

Параметры функции компонента:

- onClick: представляет функцию-обработчик нажатия кнопки
- modifier: представляет объект Modifier, который определяет модификаторы кнопки
- enabled: значение типа Boolean устанавливает, доступна ли кнопка для нажатия. По умолчанию равно true (то есть кнопка доступна для нажатия)
- interactionSource: представляет объект типа MutableInteractionSource, который устанавливает поток взаимодействий для кнопки. Значение по умолчанию - remember { MutableInteractionSource() }
- elevation: объект типа ButtonElevation?, который определяет анимацию для кнопки. По умолчанию равно ButtonDefaults.elevation()
- shape: объект типа Shape, который устанавливает форму кнопки. По умолчанию равно MaterialTheme.shapes.small
- border: объект типа BorderStroke?, который устанавливает границу кнопки. По умолчанию равно null
- colors: объект типа ButtonColors, который устанавливает цвета кнопки. По умолчанию равно ButtonDefaults.buttonColors()
- contentPadding: объект типа PaddingValues, который устанавливает отступы между границами кнопки и ее содержимым. По умолчанию равно ButtonDefaults.ContentPadding
- content: содержимое кнопки в виде строки Row

Определим простейшую кнопку:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.Button
import androidx.compose.material.Text
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Button(onClick = {}){
                Text("Click", fontSize = 25.sp)
            }
        }
    }
}
```

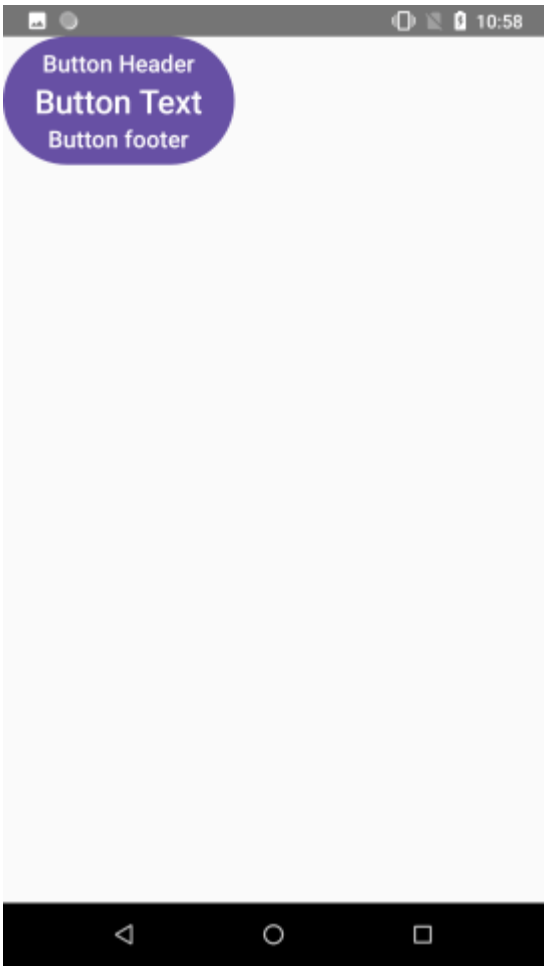


Здесь надо отметить следующие моменты. Прежде всего, необходимо определить как минимум один параметр - onClick, однако в данном случае это пустая функция, которая ничего не делает.

Второй момент - кнопка представляет сложный компонент, который может содержать другие компоненты. Таким образом мы можем создавать комплексные кнопки с различным содержимым. Например:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Column
import androidx.compose.material.Button
import androidx.compose.material.Text
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Button(onClick = {},)
            {
                Column{
                    Text("Button Header", fontSize = 30.sp)
```

```
                    Text("Button Text", fontSize = 22.sp)
                    Text("Button footer", fontSize = 18.sp)
                }
            }
        }
    }
}
```



Обработка нажатия

Для обработки нажатия параметру onClick передается функция, которая будет выполняться при нажатии.

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.Button
import androidx.compose.material.Text
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
```

```
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val label = remember{mutableStateOf("Click")}
            Button(onClick = {label.value = "Hello"}){
                Text(label.value, fontSize = 25.sp)
            }
        }
    }
}
```

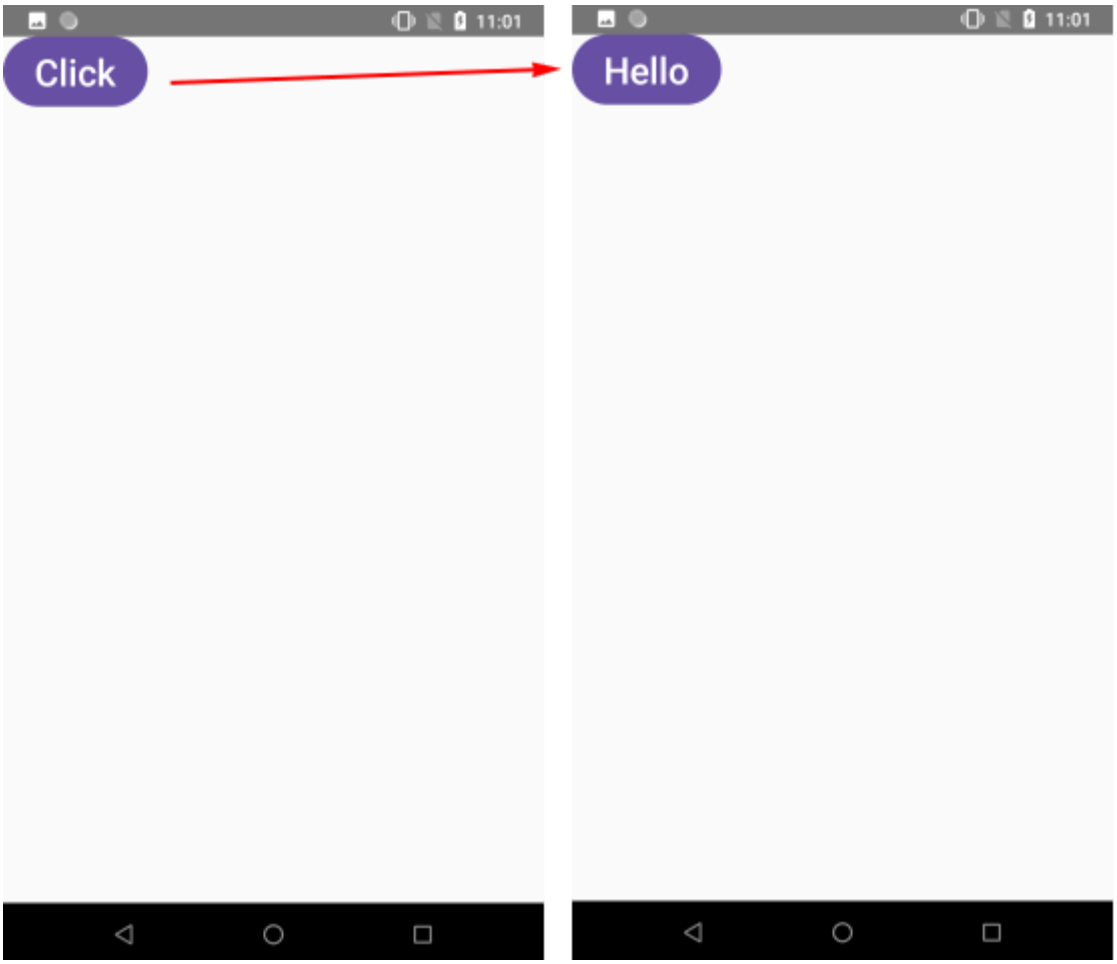
В данном случае мы определяем переменную label, которая будет хранить текст для компонента Text. Однако это не просто строка - объект типа String, а объект MutableState, который создается функцией mutableStateOf(). В эту функцию передается собственно хранимое значение, которое затем можно получить с помощью свойства value объекта MutableState и которое в данном случае мы отображаем в компоненте Text:

```
Text(label.value, fontSize = 25.sp)
```

А в обработчике нажатия мы изменяем это значение:

```
onClick = {label.value = "Hello"}
```

В итоге при нажатии на кнопку изменится ее текст.



elevation

Параметр elevation определяет анимацию для кнопки в разных состояниях и представляет объект интерфейса ButtonElevation. По умолчанию этот параметр в качестве значения имеет компонент ButtonDefaults.elevation:

```
@Composable
fun elevation(
    defaultElevation: Dp = 2.dp,
    pressedElevation: Dp = 8.dp,
    disabledElevation: Dp = 0.dp
): @Composable ButtonElevation
```

Этот компонент определяет ряд параметров, между значениями которых будет идти анимация при переключении состояния кнопки:

- defaultElevation: определяет анимацию, когда кнопка доступна для нажатия и когда для нее не определено других объектов Interaction, которые определяют состояние для кнопки.
- pressedElevation: определяет анимацию для кнопки в нажатом состоянии.
- defaultElevation: определяет анимацию для кнопки в отключенном состоянии.

Цвета кнопки

Цвета кнопки задаются с помощью параметра colors, который предоставляет объект ButtonColors и по умолчанию равен компоненту ButtonColors.buttonColors:

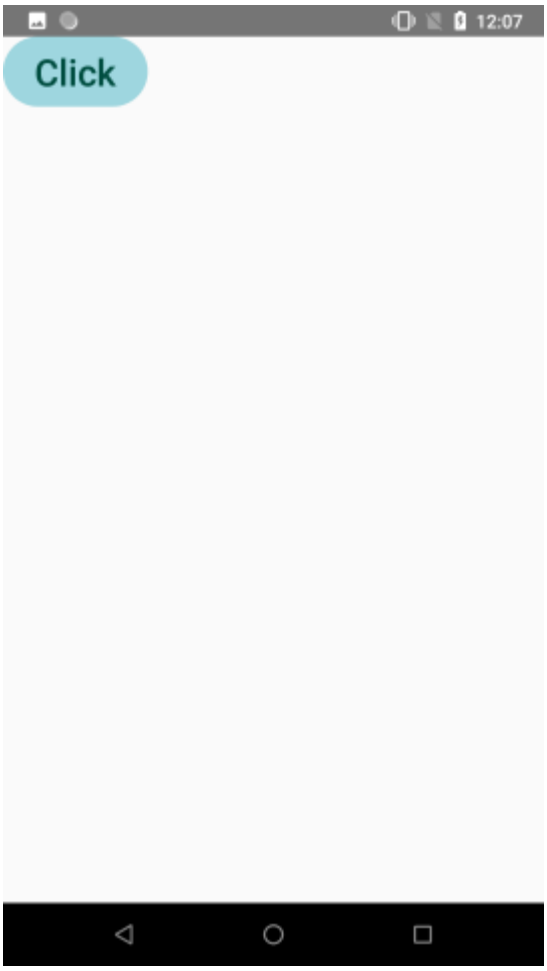
```
@Composable
fun buttonColors(
    backgroundColor: Color = MaterialTheme.colors.primary,
    contentColor: Color = contentColorFor(backgroundColor),
    disabledBackgroundColor: Color = MaterialTheme.colors.onSurface.copy(alpha =
0.12f)
            .compositeOver(MaterialTheme.colors.surface),
    disabledContentColor: Color = MaterialTheme.colors.onSurface
            .copy(alpha = ContentAlpha.disabled)
): @Composable ButtonColors
```

- backgroundColor: определяет фоновый цвет, когда кнопка доступна для нажатия.
- contentColor: определяет цвет содержимого, когда кнопка доступна для нажатия.
- disabledBackgroundColor: определяет фоновый цвет, когда кнопка не доступна для нажатия.
- disabledContentColor: определяет цвет содержимого, когда кнопка не доступна для нажатия.

Для установки цветов кнопки мы можем создать свой класс или компонент интерфейса ButtonColors, либо воспользоваться встроенным компонентом ButtonDefaults.buttonColors. Применим второй способ:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.Button
import androidx.compose.material.ButtonDefaults
import androidx.compose.material.Text
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Button(onClick = {},
                colors = ButtonDefaults.buttonColors(backgroundColor = Color.Red,
contentColor = Color.Black)
            )
            {
                Text("Click", fontSize = 25.sp)
            }
        }
    }
}
```

Здесь в качестве фонового цвета применяется красный, а в качестве цвета содержимого - черный:



Граница кнопки

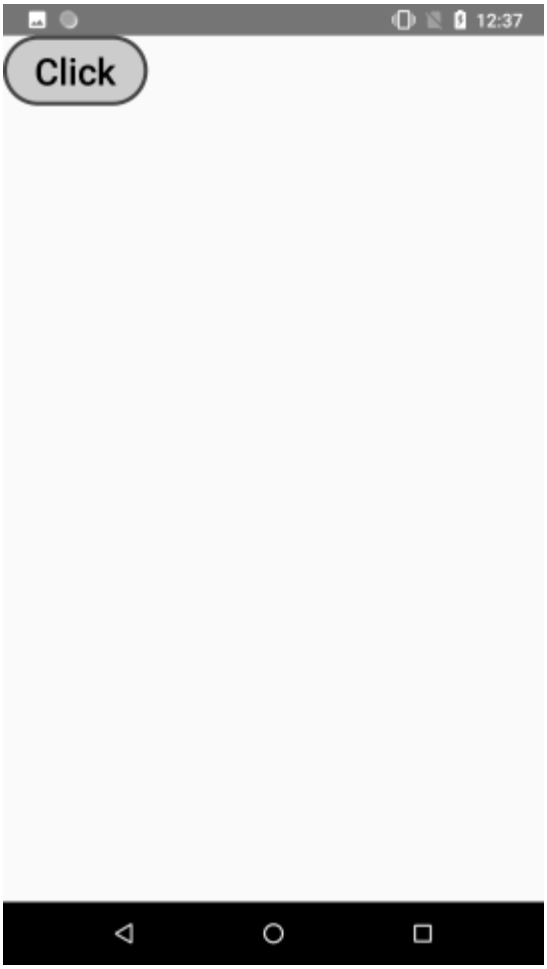
За установки границы кнопки (а именно ее толщины и цвета) отвечает параметр border, который представляет класс BorderStroke со следующими конструкторами:

```
BorderStroke(width: Dp, color: Color)
BorderStroke(width: Dp, brush: Brush)
```

Первый параметр конструкторов устанавливает толщину границы, а второй - ее цвет с помощью объекта Brush или Color. Например, определим границу у кнопку:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.BorderStroke
import androidx.compose.material.Button
import androidx.compose.material.ButtonDefaults
import androidx.compose.material.Text
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
```

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Button(onClick = {},
                colors = ButtonDefaults.buttonColors(backgroundColor =
Color.LightGray, contentColor = Color.Black),
                border = BorderStroke(3.dp, Color.DarkGray)
            )
            {
                Text("Click", fontSize = 30.sp)
            }
        }
    }
}
```



OutlinedButton

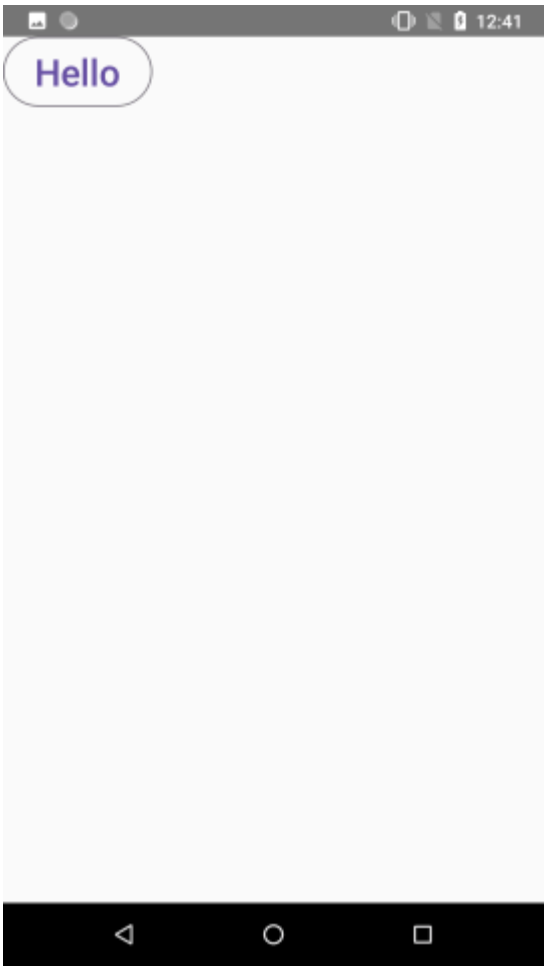
Компонент OutlinedButton также представляет кнопку и имеет тот же набор параметров:

```
@Composable
fun OutlinedButton(
    onClick: () -> Unit,
    modifier: Modifier = Modifier,
```

```
    enabled: Boolean = true,
    interactionSource: MutableInteractionSource = remember {
MutableInteractionSource() },
    elevation: ButtonElevation? = null,
    shape: Shape = MaterialTheme.shapes.small,
    border: BorderStroke? = ButtonDefaults.outlinedBorder,
    colors: ButtonColors = ButtonDefaults.outlinedButtonColors(),
    contentPadding: PaddingValues = ButtonDefaults.ContentPadding,
    content: RowScope.() -> Unit
): @Composable Unit
```

Он работает похожим образом. Главное отличие от стандартных кнопок - немного иная стилизация, которая по умолчанию добавляет ярко выраженную границу компонента и применяет иную цветоую гамму. Например:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.OutlinedButton
import androidx.compose.material.Text
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val label = remember{mutableStateOf("Click")}
            OutlinedButton(onClick = {label.value = "Hello"}){
                Text(label.value, fontSize = 25.sp)
            }
        }
    }
}
```



Для раскраски кнопки применяется компонент ButtonDefaults.outlinedButtonColors:

```
@Composable
fun outlinedButtonColors(
    backgroundColor: Color = MaterialTheme.colors.surface,
    contentColor: Color = MaterialTheme.colors.primary,
    disabledContentColor: Color = MaterialTheme.colors.onSurface
            .copy(alpha = ContentAlpha.disabled)
): @Composable ButtonColors
```

TextButton

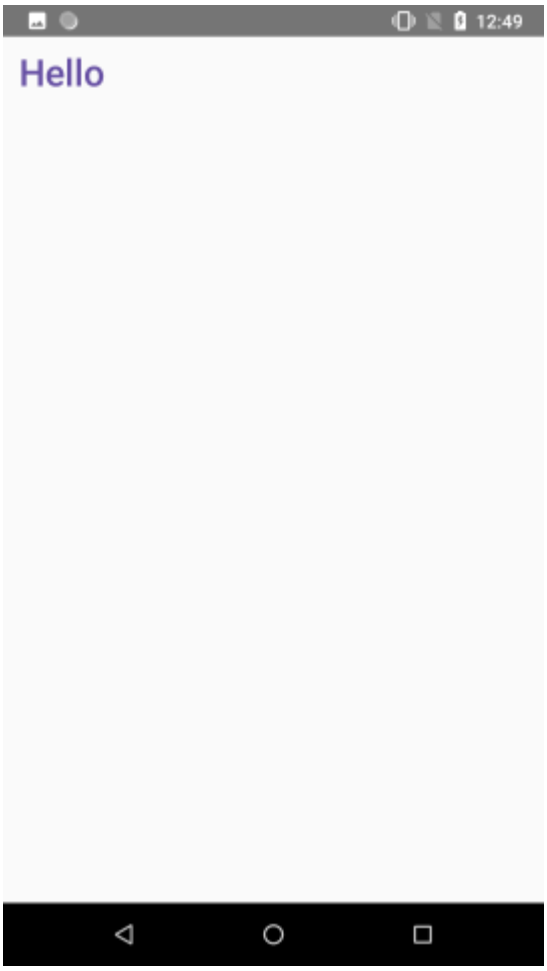
Компонент TextButton представляет еще один встроенный тип кнопок, который не имеет границы и имеет прозрачный фон. Он имеет тот же набор параметров, что и Button и OutlinedButton:

```
@Composable
fun TextButton(
    onClick: () -> Unit,
    modifier: Modifier = Modifier,
    enabled: Boolean = true,
    interactionSource: MutableInteractionSource = remember {
MutableInteractionSource() },
```

```
    elevation: ButtonElevation? = null,
    shape: Shape = MaterialTheme.shapes.small,
    border: BorderStroke? = null,
    colors: ButtonColors = ButtonDefaults.textButtonColors(),
    contentPadding: PaddingValues = ButtonDefaults.TextButtonContentPadding,
    content: RowScope.() -> Unit
): @Composable Unit
```

Используем TextButton:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.TextButton
import androidx.compose.material.Text
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val label = remember{mutableStateOf("Click")}
            TextButton(onClick = {label.value = "Hello"}){
                Text(label.value, fontSize = 25.sp)
            }
        }
    }
}
```



Для раскраски кнопки применяется компонент ButtonDefaults.textButtonColors:

```
@Composable
fun textButtonColors(
    backgroundColor: Color = Color.Transparent,
    contentColor: Color = MaterialTheme.colors.primary,
    disabledContentColor: Color = MaterialTheme.colors.onSurface
            .copy(alpha = ContentAlpha.disabled)
): @Composable ButtonColors
```

Ввод текста, TextField и OutlinedTextField

Для ввода текста в приложении предназначен компонент TextField, который определяется с помощью следующей функции:

```
@Composable
fun TextField(
    value: String,
    onValueChange: (String) -> Unit,
    modifier: Modifier = Modifier,
    enabled: Boolean = true,
    readOnly: Boolean = false,
```

```
    textStyle: TextStyle = LocalTextStyle.current,
    label: () -> Unit = null,
    placeholder: () -> Unit = null,
    leadingIcon: () -> Unit = null,
    trailingIcon: () -> Unit = null,
    isError: Boolean = false,
    visualTransformation: VisualTransformation = VisualTransformation.None,
    keyboardOptions: KeyboardOptions = KeyboardOptions.Default,
    keyboardActions: KeyboardActions = KeyboardActions(),
    singleLine: Boolean = false,
    maxLines: Int = Int.MAX_VALUE,
    interactionSource: MutableInteractionSource = remember {
MutableInteractionSource() },
    shape: Shape = MaterialTheme.shapes.small.copy(bottomEnd = ZeroCornerSize,
bottomStart = ZeroCornerSize),
    colors: TextFieldColors = TextFieldDefaults.textFieldColors()
): @Composable Unit
```

Параметры компонента:

- value: представляет введенное в текстовое поле значение в виде строки, то есть объекта String
- onValueChange: функция обработки изменения введенного значения. Представляет функцию типа (String) -> Unit
- modifier: объект типа Modifier, который задает модификаторы компонента
- enabled: устанавливает, будет ли поле доступно для ввода. Представляет значение типа Boolean. По умолчанию равно true, то есть поле доступно для ввода
- readOnly: устанавливает, будет ли поле доступно только для чтения. Представляет значение типа Boolean. По умолчанию равно false, то есть поле доступно не только для чтения, но для изменения значения
- textStyle: объект типа TextStyle, который устанавливает стиль текста. Значение по умолчанию - LocalTextStyle.current
- label: устанавливает дополнительную метку, которая отображается внутри поля. Для установки метки применяется функция типа () -> Unit. Значение по умолчанию - null
- placeholder: плейсхолдер - временный текст, который отображается внутри поля. Для установки этого текста применяется функция типа () -> Unit. Значение по умолчанию - null
- leadingIcon: устанавливает иконку, которая отображается перед текстом. Для установки применяется функция типа () -> Unit. Значение по умолчанию - null
- trailingIcon: устанавливает иконку, которая отображается после текста. Для установки применяется функция типа () -> Unit. Значение по умолчанию - null
- isError: указывает, является ли текущее введенное в поле значение некорректным. Представляет значение типа Boolean. По умолчанию равно false, то есть введенное значение корректно. Если

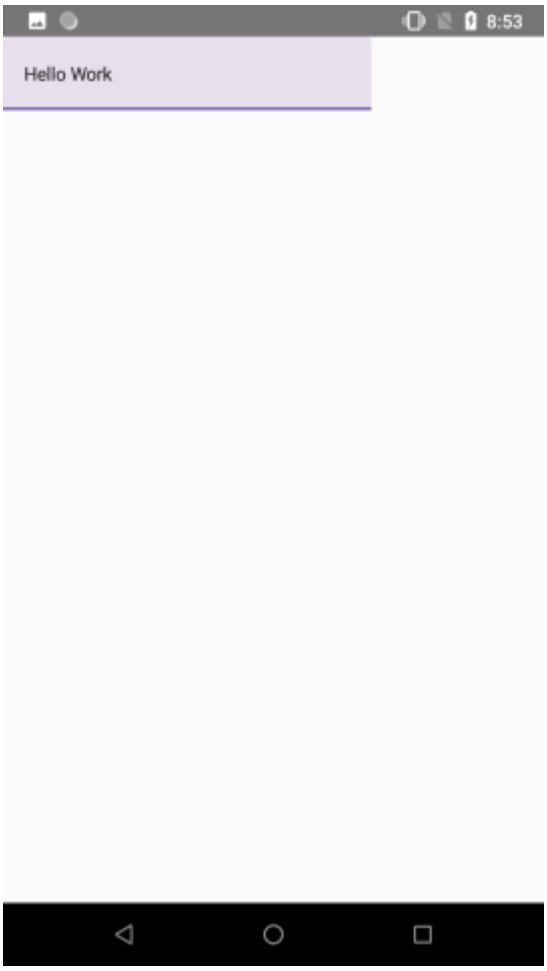
равно true, то для поля устанавливаютс соответствующие индикаторы - метка, иконка, выделение цветом, которые поддчеркивают, что введенное значение некорректно.

- visualTransformation: объект типа VisualTransformation, который задает визуальные трансформации для вводимого текста. Значение по умолчанию - VisualTransformation.None
- keyboardOptions: объект KeyboardOptions, который задает параметры клавиатуры (например, ее тип). Значение по умолчанию - KeyboardOptions.Default
- keyboardActions: KeyboardActions, который задает набор функций, которые вызываются в ответ на некоторые действия пользователя. Значение по умолчанию - KeyboardActions()
- singleLine: устанавливает, будет ли текст однострочным. По умолчанию равно false, то есть поле будет многосточным
- maxLines: задает максимальное количество строк в поле. По умолчанию равно Int.MAX_VALUE
- interactionSource: объект MutableInteractionSource, который задает поток взаимодействий для поля ввода. Значение по умолчанию - remember { MutableInteractionSource() }
- shape: представляет объект Shape, который задает форму для поля ввода. Значение по умолчанию - MaterialTheme.shapes.small.copy(bottomEnd = ZeroCornerSize, bottomStart = ZeroCornerSize)
- colors: объект TextFieldColors, который задает цвета для поля ввода. Значение по умолчанию - TextFieldDefaults.textFieldColors()

Определение простейшего поля ввода:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.TextField

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            TextField(value = "Hello Work", onValueChange = {})
        }
    }
}
```

При создании поля необходимо задать как минимум два параметра - текущее значение (параметр value) и функцию обработки ввода текста (параметр onValueChange).



Обработка ввода текста

Параметр onValueChange принимает функцию обработки ввода текста. Она в качестве параметра получает введенный текст в виде объекта String:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Column
import androidx.compose.material.Text
import androidx.compose.material.TextField
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val message = remember{mutableStateOf("")}

            Column {
                Text(message.value, fontSize = 28.sp)
```

```
                TextField(
                    value = message.value,
                    textStyle = TextStyle(fontSize=25.sp),
                    onValueChange = {newText -> message.value = newText}
                )
            }
        }
    }
}
```

В данном случае мы определяем переменную message, которая будет хранить введенный текст. Однако это не просто строка. Она будет представлять объект типа MutableState, который создается функцией mutableStateOf(). В дальнейшем мы подробнее разберем объект MutableState и функцию mutableStateOf, а пока достуточно знать, что в эту функцию передается собственно хранимое значение, которое затем можно получить с помощью свойства value объекта MutableState. А функция remember позволяет сохранить это значение.

В коде с помощью свойства value мы привязываем значение переменной к свойству text компонента Text и свойству value компонента TextField:

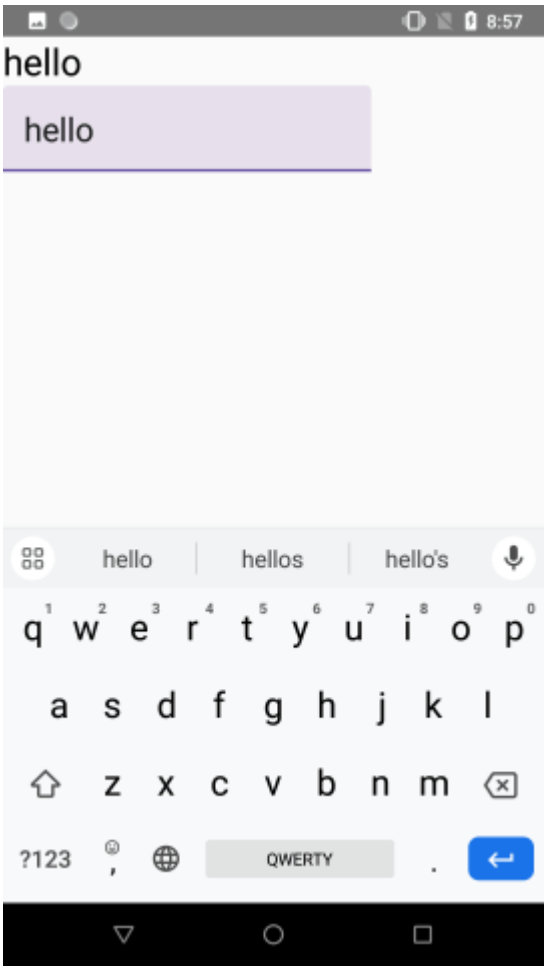
```
Text(message.value, fontSize = 28.sp)
TextField( value = message.value,
```

А в функции обработки ввода текста передаем в переменную введенный текст:

```
onValueChange = {newText -> message.value = newText}
```

В данном случае функция задается с помощью лямбда-выражения, где параметр newText представляет введенный текст.

В итоге при ввода текста в поле изменится значение в переменной message и соответственно изменится текст компонента Text.



Тип клавиатуры

В зависимости от задач приложения может потребоваться вводить разную информацию - когда буквы, когда числа и т.д. Для упрощения ввода Jetpack Compose предоставляет тип KeyboardType, который позволяет настроить тип клавиатуры с помощью своих свойств:

- KeyboardType.Ascii: предоставляет ввод символов ASCII
- KeyboardType.Email: для ввода электронного адреса
- KeyboardType.Number: для ввода цифр
- KeyboardType.NumberPassword: для ввода пароля из цифр
- KeyboardType.Password: для ввода пароля
- KeyboardType.Phone: для ввода номера телефона
- KeyboardType.Text: предоставляет стандартную клавиатуру
- KeyboardType.Uri: предоставляет клавиатуру для ввода URI

С помощью параметра keyboardOptions, который представляет класс KeyboardOptions, можно задать тип клавиатуры. Например, применим клавиатуру для ввода номера телефона:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.material.TextField
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val phone = remember{mutableStateOf("")}

            TextField(
                phone.value,
                {phone.value = it},
                textStyle = TextStyle(fontSize =  28.sp),
                keyboardOptions = KeyboardOptions(keyboardType =
KeyboardType.Phone)
            )
        }
    }
}
```

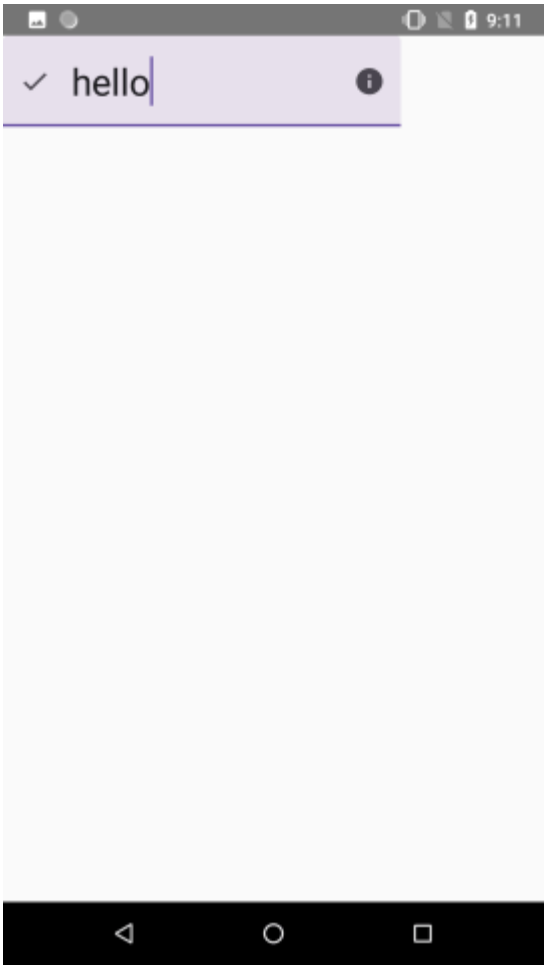
Установка иконок

Параметр leadingIcon задает иконку перед текстом, а параметр trailingIcon - иконку после текста. В качестве значения оба параметра принимают функцию типа () -> Unit. Определим иконки для поля ввода:

```
package com.example.helloapp

import android.R.attr
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.Icon
import androidx.compose.material.TextField
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.unit.sp

import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Check
import androidx.compose.material.icons.filled.Info

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
```

```
        setContent {
            val message = remember{mutableStateOf("")}

            TextField(
                message.value,
                {message.value = it},
                textStyle = TextStyle(fontSize =  28.sp),
                leadingIcon = { Icon(Icons.Filled.Check, contentDescription =
"Проверено") },
                trailingIcon = { Icon(Icons.Filled.Info, contentDescription =
"Дополнительная информация") }
            )
        }
    }
}
```

Для определения иконок применяется встроенный тип androidx.compose.material.Icon, в функцию которого передается значок иконки. В данном случае применяются встроенные иконки Icons.Filled.Check и Icons.Filled.Info. Второй параметр - contentDescription позволяет указать к иконке описание.



Плейсхолдер

Параметр placeholder устанавливает плейсхолдер или заменитель текста, отображаемый в поле ввода, в виде другого компонента:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.Text
import androidx.compose.material.TextField
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val message = remember{mutableStateOf("")}
            TextField(
                message.value,
                {message.value = it},
                textStyle = TextStyle(fontSize =  28.sp),
                placeholder = { Text("Hello Work!") }
            )
        }
    }
}
```

Установка цветовой палитры поля ввода

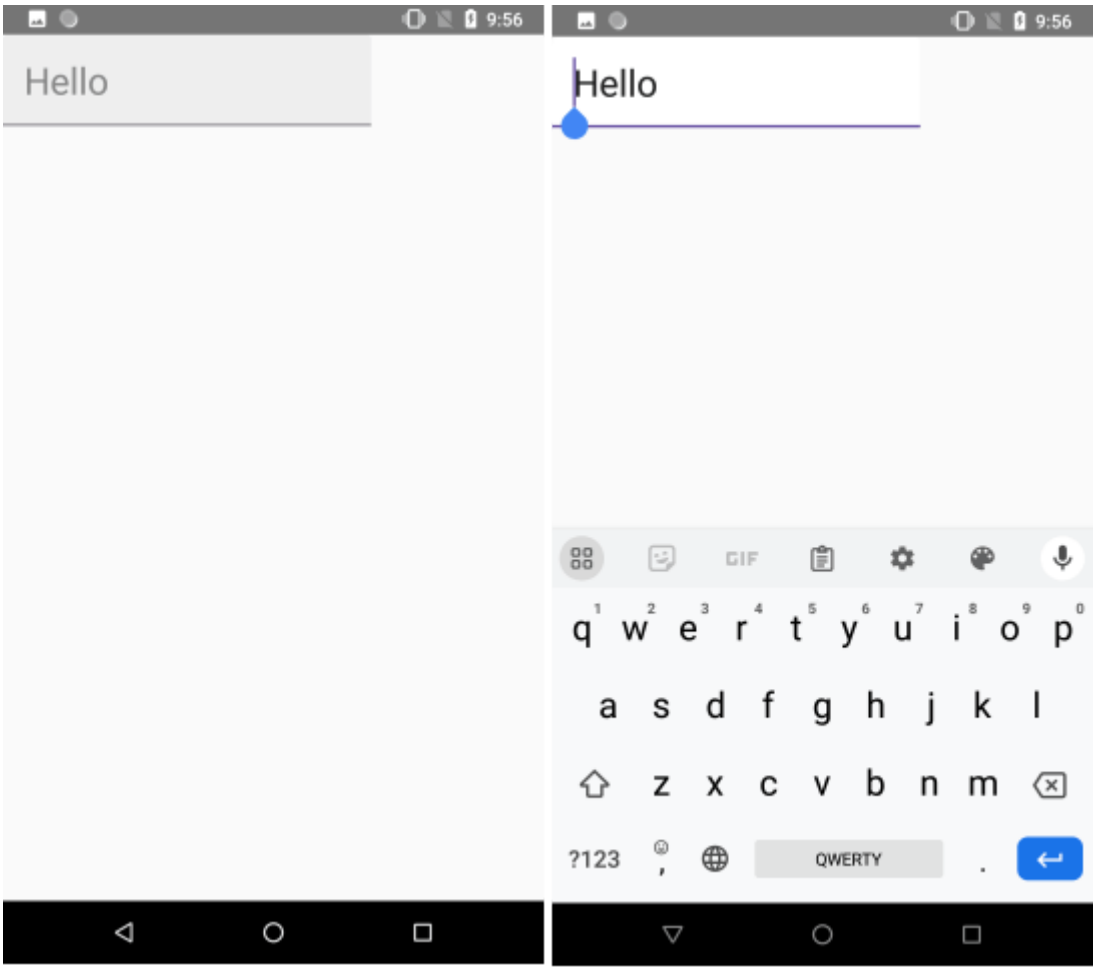
Параметр colors, который представляет объект интерфейса TextFieldColors, задает цвета для поля ввода. По умолчанию он принимает компонент TextFieldDefaults.textFieldColors(), который устанавливает цвета для самых различных состояний:

```
@Composable
fun textFieldColors(
    textColor: Color = LocalContentColor.current.copy(LocalContentAlpha.current),
    disabledTextColor: Color = textColor.copy(ContentAlpha.disabled),
    backgroundColor: Color = MaterialTheme.colors.onSurface.copy(alpha =
BackgroundOpacity),
    cursorColor: Color = MaterialTheme.colors.primary,
    errorCursorColor: Color = MaterialTheme.colors.error,
    focusedIndicatorColor: Color = MaterialTheme.colors.primary.copy(alpha =
ContentAlpha.high),
    unfocusedIndicatorColor: Color = MaterialTheme.colors.onSurface.copy(alpha =
UnfocusedIndicatorLineOpacity),
    disabledIndicatorColor: Color = unfocusedIndicatorColor.copy(alpha =
ContentAlpha.disabled),
    errorIndicatorColor: Color = MaterialTheme.colors.error,
    leadingIconColor: Color = MaterialTheme.colors.onSurface.copy(alpha =
IconOpacity),
```

```
    disabledLeadingIconColor: Color = leadingIconColor.copy(alpha =
ContentAlpha.disabled),
    errorLeadingIconColor: Color = leadingIconColor,
    trailingIconColor: Color = MaterialTheme.colors.onSurface.copy(alpha =
IconOpacity),
    disabledTrailingIconColor: Color = trailingIconColor.copy(alpha =
ContentAlpha.disabled),
    errorTrailingIconColor: Color = MaterialTheme.colors.error,
    focusedLabelColor: Color = MaterialTheme.colors.primary.copy(alpha =
ContentAlpha.high),
    unfocusedLabelColor: Color =
MaterialTheme.colors.onSurface.copy(ContentAlpha.medium),
    disabledLabelColor: Color = unfocusedLabelColor.copy(ContentAlpha.disabled),
    errorLabelColor: Color = MaterialTheme.colors.error,
    placeholderColor: Color =
MaterialTheme.colors.onSurface.copy(ContentAlpha.medium),
    disabledPlaceholderColor: Color = placeholderColor.copy(ContentAlpha.disabled)
): @Composable TextFieldColors
```

Мы можем использовать этот компонент для настройки цветовой гаммы поля ввода:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.TextField
import androidx.compose.material.TextFieldDefaults
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val message = remember{mutableStateOf("")}

            TextField(
                message.value,
                {message.value = it},
                textStyle = TextStyle(fontSize =  28.sp),
                colors = TextFieldDefaults.textFieldColors(textColor=Color.Red,
backgroundColor = Color.LightGray)
            )
        }
    }
}
```



OutlinedTextField

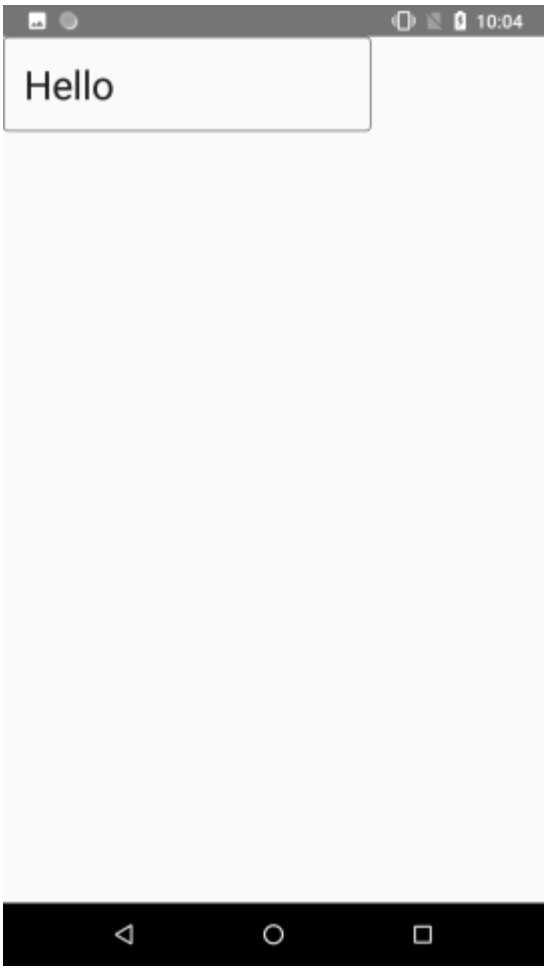
Компонент OutlinedTextField во многом похож на TextField за тем исключением, что он добавляет границу вокуг поля ввода. Его функция принимает следующие параметры:

```
@Composable
fun OutlinedTextField(
    value: String,
    onValueChange: (String) -> Unit,
    modifier: Modifier = Modifier,
    enabled: Boolean = true,
    readOnly: Boolean = false,
    textStyle: TextStyle = LocalTextStyle.current,
    label: () -> Unit = null,
    placeholder: () -> Unit = null,
    leadingIcon: () -> Unit = null,
    trailingIcon: () -> Unit = null,
    isError: Boolean = false,
    visualTransformation: VisualTransformation = VisualTransformation.None,
    keyboardOptions: KeyboardOptions = KeyboardOptions.Default,
    keyboardActions: KeyboardActions = KeyboardActions.Default,
    singleLine: Boolean = false,
    maxLines: Int = Int.MAX_VALUE,
    interactionSource: MutableInteractionSource = remember {
MutableInteractionSource() },
    shape: Shape = MaterialTheme.shapes.small,
```

```
    colors: TextFieldColors = TextFieldDefaults.outlinedTextFieldColors()
): @Composable Unit
```

Простейшее определение компонента OutlinedTextField:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.OutlinedTextField
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val message = remember{mutableStateOf("")}

            OutlinedTextField(
                message.value,
                {message.value = it},
                textStyle = TextStyle(fontSize =  30.sp),
            )
        }
    }
}
```



В функции компонента OutlinedTextField мы видим, что в отличие от TextField, меняется стандартное значение для последнего параметра - colors, который устанавливает цвета. Теперь он по умолчанию равен компоненту TextFieldDefaults.outlinedTextFieldColors():

```
@Composable
fun outlinedTextFieldColors(
    textColor: Color = LocalContentColor.current.copy(LocalContentAlpha.current),
    disabledTextColor: Color = textColor.copy(ContentAlpha.disabled),
    backgroundColor: Color = Color.Transparent,
    cursorColor: Color = MaterialTheme.colors.primary,
    errorCursorColor: Color = MaterialTheme.colors.error,
    focusedBorderColor: Color = MaterialTheme.colors.primary.copy(alpha =
ContentAlpha.high),
    unfocusedBorderColor: Color = MaterialTheme.colors.onSurface.copy(alpha =
ContentAlpha.disabled),
    disabledBorderColor: Color = unfocusedBorderColor.copy(alpha =
ContentAlpha.disabled),
    errorBorderColor: Color = MaterialTheme.colors.error,
    leadingIconColor: Color = MaterialTheme.colors.onSurface.copy(alpha =
IconOpacity),
    disabledLeadingIconColor: Color = leadingIconColor.copy(alpha =
ContentAlpha.disabled),
    errorLeadingIconColor: Color = leadingIconColor,
    trailingIconColor: Color = MaterialTheme.colors.onSurface.copy(alpha =
IconOpacity),
    disabledTrailingIconColor: Color = trailingIconColor.copy(alpha =
```

```
ContentAlpha.disabled),
    errorTrailingIconColor: Color = MaterialTheme.colors.error,
    focusedLabelColor: Color = MaterialTheme.colors.primary.copy(alpha =
ContentAlpha.high),
    unfocusedLabelColor: Color =
MaterialTheme.colors.onSurface.copy(ContentAlpha.medium),
    disabledLabelColor: Color = unfocusedLabelColor.copy(ContentAlpha.disabled),
    errorLabelColor: Color = MaterialTheme.colors.error,
    placeholderColor: Color =
MaterialTheme.colors.onSurface.copy(ContentAlpha.medium),
    disabledPlaceholderColor: Color = placeholderColor.copy(ContentAlpha.disabled)
): @Composable TextFieldColors
```

Здесь мы можем найти такие параметры как focusedBorderColor, unfocusedBorderColor, disabledBorderColor и errorBorderColor, который позволяют установить цвет границы для различных состояний поля ввода. Например:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.OutlinedTextField
import androidx.compose.material.TextFieldDefaults
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val message = remember{mutableStateOf("")}

            OutlinedTextField(
                message.value,
                {message.value = it},
                textStyle = TextStyle(fontSize =  30.sp),
                colors = TextFieldDefaults.outlinedTextFieldColors(
                    focusedBorderColor= Color.Green, // цвет при получении фокуса
                    unfocusedBorderColor = Color.LightGray  // цвет при отсутствии
фокуса
                )
            )
        }
    }
}
```

Modifier.toggleable

Модификатор Modifier.toggleable устанавливает для компонента два состояния и позволяет переключаться между этими состояниями. Подобные компоненты в этом плане похожи на флажки (checkbox), в которых можно поставить отметку, а можно ее снять, тем самым изменив состояние флажка. Функция модификатора принимает следующие параметры:

```
fun Modifier.toggleable(
    value: Boolean,
    enabled: Boolean = true,
    role: Role? = null,
    onValueChange: (Boolean) -> Unit
): Modifier
```

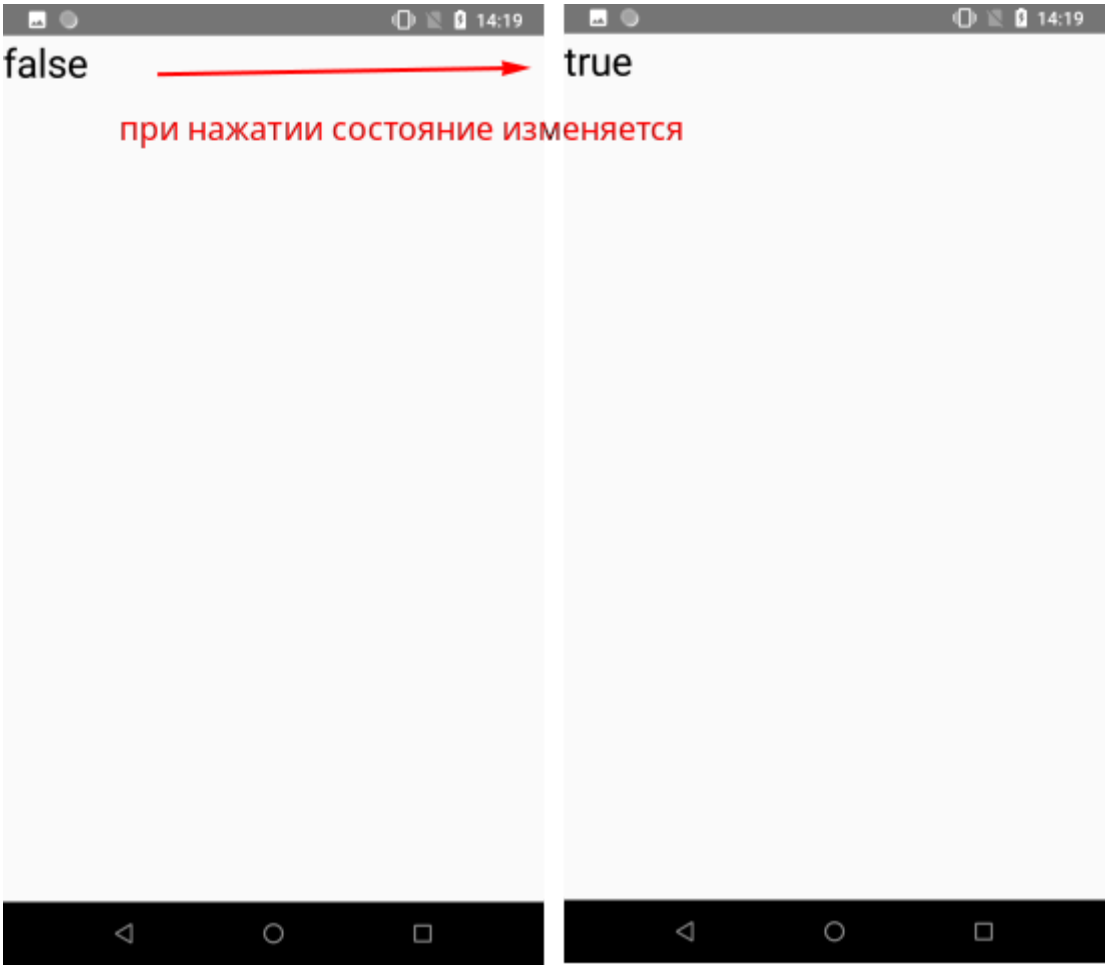
- value хранит состояние компонента в виде объекта Boolean, поэтому состояние может принимать только два значения: true и false
- enabled указывает, будет ли компонент доступен для выбора. Если он имеет значение true (значение по умолчанию), то компонент будет доступен
- role представляет объект Role, который представляет тип элемента интерфейса
- onValueChange представляет функцию типа (Boolean) -> Unit, которая вызывается при нажатии на компонент

Создадим переключаемый компонент на основе компонента Text:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.selection.toggleable
import androidx.compose.material.Text
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        setContent {
            var checked = remember { mutableStateOf(false) }
            Text(
                modifier = Modifier.toggleable(value = checked.value,
onValueChange = { checked.value = it }),
                text = checked.value.toString(),
```

```
                fontSize = 30.sp
            )
        }
    }
}
```

Для отслеживания состояния создается переменная checked, которая по умолчанию хранит значение false. Это значение передается в функцию модификатора toggleable параметру value. А с помощью другого параметра - onValueChange задаем функцию, которая получает новое состояние через параметр it и передает его переменной checked. Таким образом, при нажатии состояние компонента будет переключаться.



Checkbox

Компонент Checkbox представляет флажок, который может быть в отмеченном и неотмеченном состоянии. Его функция принимает следующие параметры:

```
@Composable
fun Checkbox(
    checked: Boolean,
    onCheckedChange: (Boolean) -> Unit,
    modifier: Modifier = Modifier,
    enabled: Boolean = true,
```

```
    interactionSource: MutableInteractionSource = remember {
MutableInteractionSource() },
    colors: CheckboxColors = CheckboxDefaults.colors()
): @Composable Unit
```

- checked: указывает, будет ли отмечен флажок. Представляет значение Boolean. Если равен true, то флажок отмечен.
- onCheckedChange: представляет функции типа (Boolean) -> Unit, которая выполняется при изменении состояния флажка (установки или снятия отметки). В качестве параметра в функцию передается новое состояние флажка.
- modifier: объект Modifier, который устанавливает для флажка модификаторы
- enabled: указывает, будет ли доступен флажок. Представляет значение Boolean и по умолчанию равен true (то есть флажок будет доступен).
- interactionSource: объект MutableInteractionSource, который задает поток взаимодействий для флажка. По умолчанию равен remember { MutableInteractionSource() }.
- colors: объект CheckboxColors, который задает цвета для флажка. По умолчанию равен CheckboxDefaults.colors().

Простейший флажок:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.Checkbox
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val checkedState = remember { mutableStateOf(true) }
            Checkbox(
                checked = checkedState.value,
                onCheckedChange = { checkedState.value = it }
            )
        }
    }
}
```

Здесь надо отметить два момента. Прежде всего для компонента Checkbox необходимо задать как минимум два параметра: checked и onCheckedChange. Для этого в примере выше определена переменная checkedState, которая представляет состояние флажка (отмечен или нет). Эта переменная

представляет объект типа MutableState, который создается функцией mutableStateOf(). В эту функцию передается собственно хранимое значение - в данном случае значение true, которое затем можно получить с помощью свойства value объекта MutableState. А функция remember позволяет сохранить это значение.

В коде с помощью свойства value мы привязываем значение переменной к свойству checked:

```
checked = checkedState.value
```

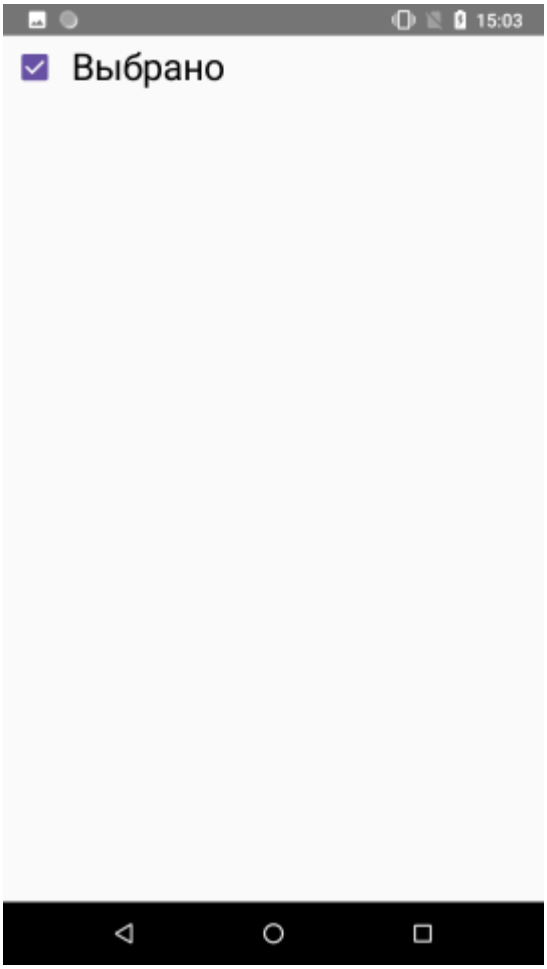
А в функции обработки изменения состояния флажка с помощью параметра it передаем в переменную новое состояние флажка (если отмечен - true, если отметка отсутвует - false):

```
onCheckedChange = { checkedState.value = it }
```

Второй момент, который надо отметить, флажок не предоставляет встроенных возможностей по установке текстовой метки. Однако, как правило, при флажке идет некоторый текст, который некоторым образом объясняет назначение флажка. Но в Jetpack Compose подобную текстовую метку нам надо устанавливать дополнительно:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.padding
import androidx.compose.material.Checkbox
import androidx.compose.material.Text
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val checkedState = remember { mutableStateOf(true) }
            Row{
                Checkbox(
                    checked = checkedState.value,
                    onCheckedChange = { checkedState.value = it },
                    modifier = Modifier.padding(5.dp)
                )
                Text("Выбрано", fontSize = 22.sp)
            }
```

```
            }
        }
    }
```



Из параметров следует отметить параметр colors, который представляет тип CheckboxColors и устанавливает цветовую гамму флажка. По умолчанию он хранит компонент CheckboxDefaults.colors

```
@Composable
fun colors(
    checkedColor: Color = MaterialTheme.colors.secondary,
    uncheckedColor: Color = MaterialTheme.colors.onSurface.copy(alpha = 0.6f),
    checkmarkColor: Color = MaterialTheme.colors.surface,
    disabledColor: Color = MaterialTheme.colors.onSurface.copy(alpha =
ContentAlpha.disabled),
    disabledIndeterminateColor: Color = checkedColor.copy(alpha =
ContentAlpha.disabled)
): @Composable CheckboxColors
```

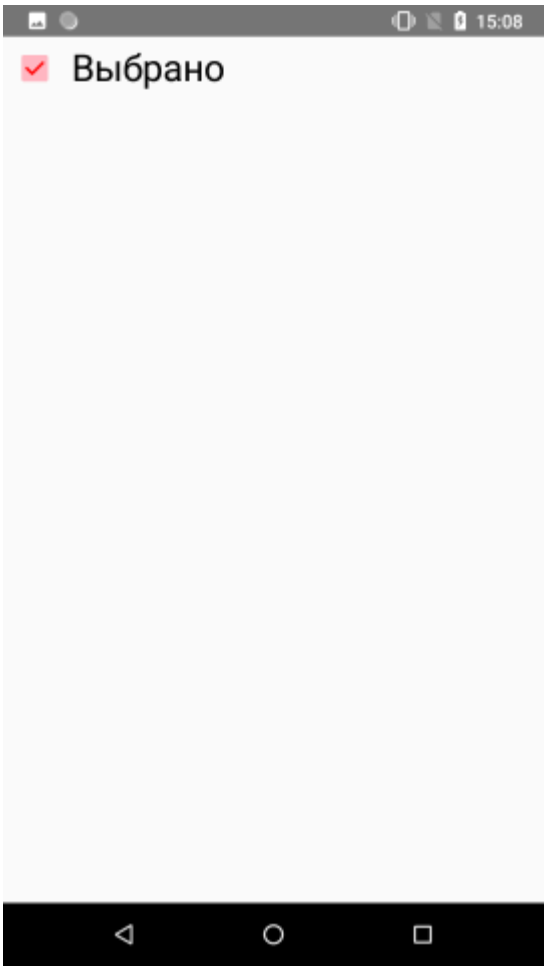
Его параметры:

- checkedColor: цвет флажка, когда он находится в отмеченном состоянии
- uncheckedColor: цвет флажка, когда он неотмечен

- checkmarkColor: цвет отметки флажка
- disabledColor: цвет флажка, когда он не доступен для нажатия
- disabledIndeterminateColor: цвет флажка типа TriStateCheckbox, когда он не доступен и одновременно находится в состоянии ToggleableState.Indeterminate

Используем данный параметр для настройки цветов:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.padding
import androidx.compose.material.Checkbox
import androidx.compose.material.CheckboxDefaults
import androidx.compose.material.Text
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val checkedState = remember { mutableStateOf(true) }
            Row{
                Checkbox(
                    checked = checkedState.value,
                    onCheckedChange = { checkedState.value = it },
                    modifier = Modifier.padding(5.dp),
                    colors  = CheckboxDefaults.colors(checkedColor = Color(0xff,
0xb6, 0xc1), checkmarkColor = Color.Red)
                )
                Text("Выбрано", fontSize = 22.sp)
            }
        }
    }
}
```



TriStateCheckbox

Компонент TriStateCheckbox расширяет Checkbox, добавляя возможность установить флажок в третье - неопределенное состояние

```
@Composable
fun TriStateCheckbox(
    state: ToggleableState,
    onClick: () -> Unit,
    modifier: Modifier = Modifier,
    enabled: Boolean = true,
    interactionSource: MutableInteractionSource = remember {
MutableInteractionSource() },
    colors: CheckboxColors = CheckboxDefaults.colors()
): @Composable Unit
```

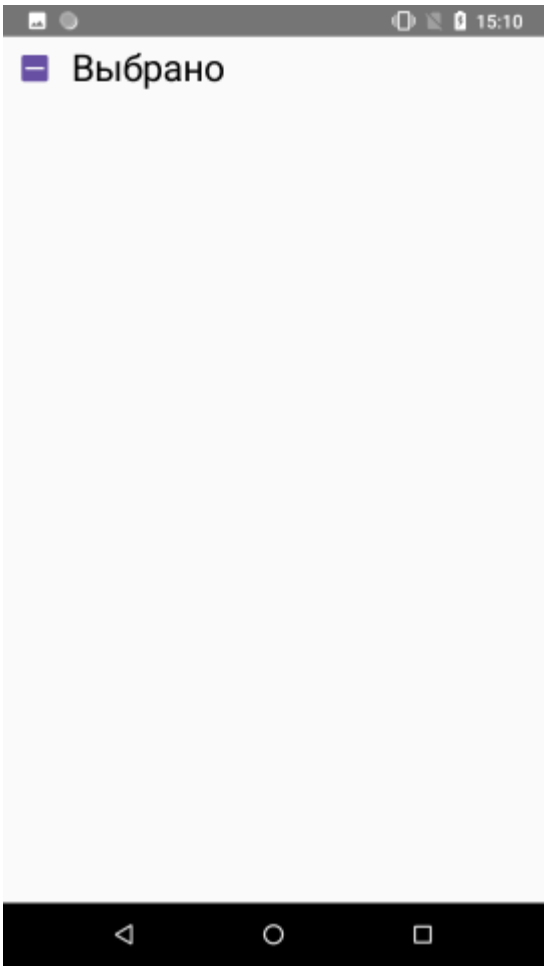
Он принимает почти те же параметры, что и Checkbox, за исключением двух параметров. Так, параметр state представляет тип ToggleableState и устанавливает состояние флажка с помощью следующих значений:

- ToggleableState.Indeterminate: неопределенное состояние
- ToggleableState.Off: отметка снята
- ToggleableState.On: флажок отмечен

Второй отличающийся параметр - onClick задает функцию изменения состояния флажка.

Определим простейший компонент TriStateCheckbox:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.padding
import androidx.compose.material.TriStateCheckbox
import androidx.compose.material.Text
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Modifier
import androidx.compose.ui.state.ToggleableState
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val checkedState = remember {
mutableStateOf(ToggleableState.Indeterminate) }
            Row{
                TriStateCheckbox(
                    state = checkedState.value,
                    onClick = {
                        if (checkedState.value == ToggleableState.Indeterminate ||
checkedState.value == ToggleableState.Off)
                            checkedState.value = ToggleableState.On
                        else checkedState.value = ToggleableState.Off
                    },
                    modifier = Modifier.padding(5.dp),
                )
                Text("Выбрано", fontSize = 22.sp)
            }
        }
    }
}
```



Выбираемый компонент и модификатор selectable

В данной статье мы рассмотим, как сделать компонент выбираемым. Чтобы сделать любой компонент выбираемым, применяется модификатор Modifier.selectable, который имеет следующее определение:

Модификатор Modifier.selectable() делает компонент (в данном случае компонент Row) выделяемым. То есть мы можем выбрать не просто радиокнопку, а всю строку. Данный модификатор имеет следующие параметры:

```
fun Modifier.selectable(
    selected: Boolean,
    enabled: Boolean = true,
    role: Role? = null,
    onClick: () -> Unit
): Modifier
```

- selected указывает, будет ли компонент выбран. Если он имеет значение true, то компонент выбран
- enabled указывает, будет ли компонент доступен для выбора. Если он имеет значение true (значение по умолчанию), то компонент будет доступен
- role представляет объект Role, который представляет тип элемента интерфейса
- onClick представляет функцию типа () -> Unit, которая вызывается при нажатии на компонент

Рассмотрим простейший пример:

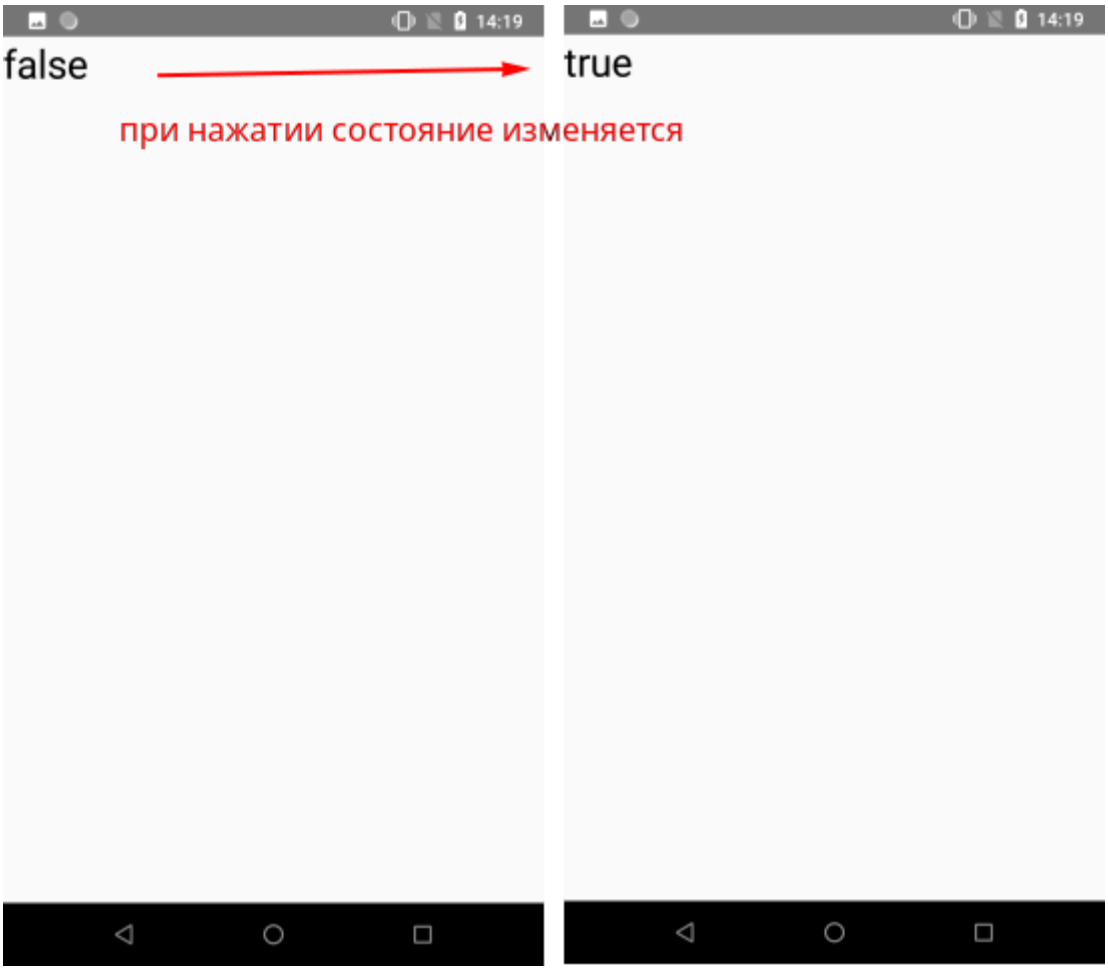
```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.selection.selectable
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Modifier
import androidx.compose.material.Text
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val selected = remember { mutableStateOf(true) }

            Text(
                text= selected.value.toString(),
                fontSize = 30.sp,
                modifier = Modifier.selectable(
                        selected = selected.value,
                        onClick = { selected.value = !selected.value }
                    )
            )
        }
    }
}
```

Здесь для компонента Text устанавливается модификатор Modifier.selectable:

```
modifier = Modifier.selectable(
    selected = selected.value,
    onClick = { selected.value = !selected.value }
)
```

Параметр selected модификатора, который указывает, будет ли выбран компонент, получает значение из переменной selected, которая представляет объект MutableState, то есть значение selected.value будет равно true или false.

А параметр onClick, который задает функцию, выполняемую при нажатии на компонент, будет переключать значение selected.value. То есть если это значение было равно true, оно становится равным false и наоборот.

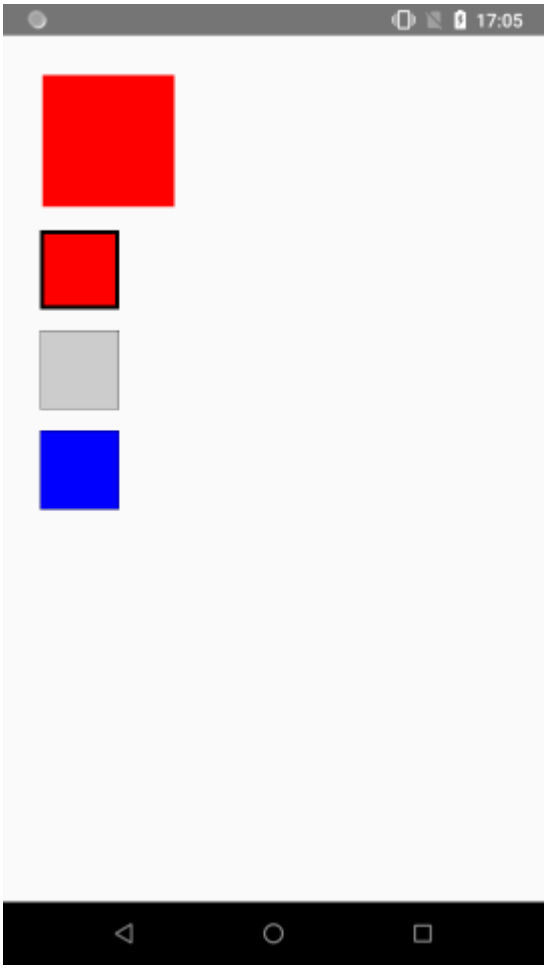


Выбор из группы компонентов

Обычно данный модификатор применяется для настройки выбора из нескольких компонентов. В этом случае все эти компоненты рассматриваются как единая группа, из которой одномоменто можно выбрать только один компонент. Рассмотрим небольшой пример:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.background
import androidx.compose.foundation.border
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.selection.selectable
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        setContent {
```

```
            val colors = listOf(Color.Red, Color.Green, Color.Blue)
            val selectedOption = remember { mutableStateOf(colors[0]) }
            Column(modifier =Modifier.padding(20.dp)) {
                Box( Modifier.padding(10.dp).size(100.dp).background(color =
selectedOption.value))

                colors.forEach { color ->
                    val selected = selectedOption.value == color
                    Box(
                        Modifier.padding(8.dp)
                            .size(60.dp)
                            .background(color = color)
                            .selectable(
                                selected = selected,
                                onClick = { selectedOption.value = color }
                            )
                            .border(
                                width= if(selected){2.dp} else{0.dp},
                                color = Color.Black
                            )
                    )
                }
            }
        }
    }
}
```



В данном случае у нас три варианта выбора, которые описываются списком colors:

```
val colors = listOf(Color.Red, Color.Green, Color.Blue)
```

Для хранения текущего выбранного элемента создаем переменную selectedOption, которая по умолчанию указывает на первый элемент из списка:

```
val selectedOption = remember { mutableStateOf(colors[0]) }
```

Каждый элемент списка представляет описание цвета. С помощью функции forEach() пробегаемся по каждому И для каждого из этих элементов мы создаем отдельный элемент Box:

```
colors.forEach { color ->
    val selected = selectedOption.value == color
    Box(
```

Условно говоря, закрепляем за каждым компонентом Box свой элемент из списка colors. В функцию forEach() передается лямбда-выражение, в которое в качестве параметра передается текущий элемент списка. Затем устанавливаем переменную selected, которая для данного элемента будет указывать, выбран ли данный элемент (иначе говоря равен ли он значению selectedOption.value)

В модификаторе selectable устанавливаем, что компонент выбран, если выбран текущий элемент списка colors:

```
.selectable(
    selected = selected,
    onClick = { selectedOption.value = color }
)
```

А в параметру onClick передается функция, которая при нажатии на данный компонент присваивает переменной selectedOption значение текщего элемента списка.

Комплексные компоненты

Теперь изменим пример выше, добавим к компонентам Box текстовые метки:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.background
import androidx.compose.foundation.border
```

```
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.selection.selectable
import androidx.compose.material.Text
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        setContent {
            val colors = listOf(
                ColorData("красный", Color.Red),
                ColorData("зеленый", Color.Green),
                ColorData("синий", Color.Blue))
            val selectedOption = remember { mutableStateOf(colors[0]) }
            Column(modifier =Modifier.padding(20.dp)) {
                Box( Modifier.padding(10.dp).size(100.dp).background(color =
selectedOption.value.color))

                colors.forEach { colorData ->
                    val selected = selectedOption.value == colorData
                    Row(modifier = Modifier.selectable(
                                    selected = selected,
                                    onClick = { selectedOption.value = colorData
}),
                        verticalAlignment = Alignment.CenterVertically
                    ){
                        Box(
                            Modifier.padding(8.dp)
                                .size(60.dp)
                                .background(color = colorData.color)
                                .border(
                                    width= if(selected){2.dp} else{0.dp},
                                    color = Color.Black
                                )
                        )
                        Text(text = colorData.title, fontSize = 22.sp)
                    }
                }
            }
        }
    }
}
data class ColorData(val title:String, val color: Color)
```

В данном случае создаем класс ColorData, который описывает применяемые данные и который хранит два значения - название цвета и сам цвет. В коде MainActivity также создаем список данных, только теперь каждый элемент списка представляет объект ColorData. Далее проходим по каждому из этих объектов и для каждого из них создаем строку Row, которая содержит Box, выржабщим цвет, и Text, который отображает название цвета.

RadioButton

Компонент RadioButton представляет переключатель или радиокнопку и служит для создания группы радиокнопок, из которых одномоментно можно выбрать только один переключатель. Этот компонент имеет следующие параметры:

```
@Composable
fun RadioButton(
    selected: Boolean,
    onClick: () -> Unit,
    modifier: Modifier = Modifier,
    enabled: Boolean = true,
    interactionSource: MutableInteractionSource = remember {
MutableInteractionSource() },
    colors: RadioButtonColors = RadioButtonDefaults.colors()
): @Composable Unit
```

- selected: указывает, будет ли отмечена радиокнопка. Представляет значение Boolean. Если равен true, то радиокнопка отмечена.
- onClick: представляет функцию типа () -> Unit, которая выполняется при нажатия на радиокнопку.
- modifier: объект Modifier, который устанавливает для радиокнопки модификаторы
- enabled: указывает, будет ли доступна радиокнопка. Представляет значение Boolean и по умолчанию равен true (то есть радиокнопка будет доступна).
- interactionSource: объект MutableInteractionSource, который задает поток взаимодействий для радиокнопки. По умолчанию равен remember { MutableInteractionSource() }.
- colors: объект RadioButtonColors, который задает цвета для радиокнопки. По умолчанию равен RadioButtonDefaults.colors().

Создадим группу из двух радиокнопок:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.selection.selectableGroup
import androidx.compose.material.RadioButton
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Modifier

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val state = remember { mutableStateOf(true) }
            Column(Modifier.selectableGroup())
            {
                RadioButton(
                    selected = state.value,
                    onClick = { state.value = true }
                )
                RadioButton(
                    selected = !state.value,
                    onClick = { state.value = false }
                )
            }
        }
    }
}
```

Для создании группы радиокнопок, которые рассматриваются именно как группа или единое целое, у контейнера - компонента Column или Row устанавливается модификатор Modifier.selectableGroup(). В данном случае радиокнопки помещаются в Column и соответственно будут располагаться в столбик:

```
Column(Modifier.selectableGroup())
```

Хотя также можно было бы расположить радиокнопки в строку, поместив в контейнер Row.

Для хранения состояния радиокнопок определяется переменная state, которая представляет тип `MutableState<Boolean>`:

```
val state = remember { mutableStateOf(true) }
```

С помощью свойства value получаем хранимое в переменной значение (true или false) и передаем его параметру selected радиокнопок:

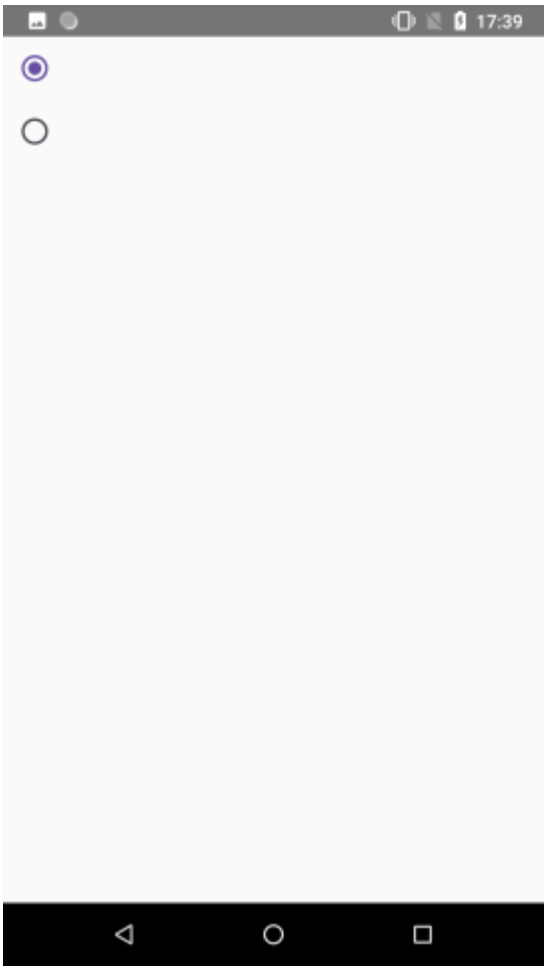
```
selected = state.value
```

Но поскольку только одна радиокнопка одномоментно может быть выбрана, то другой радиокнопке передается противоположеное значение:

```
selected = !state.value
```

А в обработчике нажатия из параметра onClick изменяем данное значение:

```
onClick = { state.value = true }
```

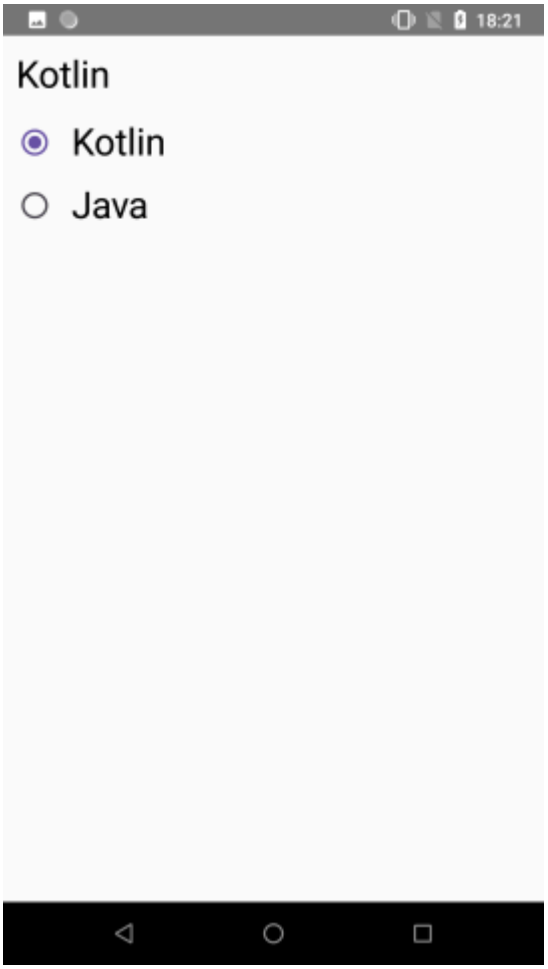


Добавление радиокнопкам текстовых меток

В примере выше мы видим, что для радиокнопок, как и для флажков, неопределяется никакой текстовой метки, которая несла бы самую минимальную информацию о радиокнопки. В этом случае необходимо самостоятельно комбинировать радиокнопку с текстовыми компонентами:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.selection.selectableGroup
import androidx.compose.material.RadioButton
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Modifier
import androidx.compose.material.Text
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
```

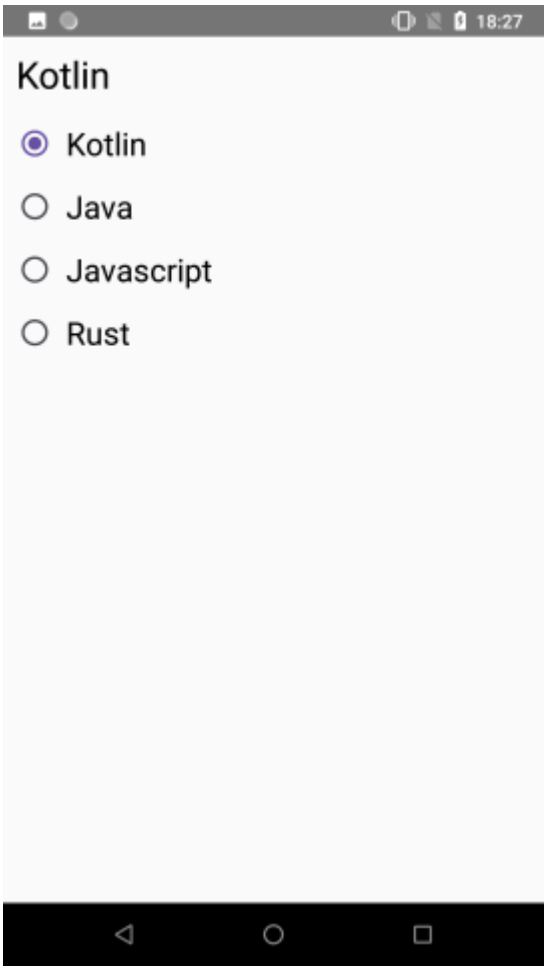
```
        setContent {
            val state = remember { mutableStateOf(true) }
            val change= { state.value = !state.value }
            Column(Modifier.selectableGroup())
            {
                Row{
                    RadioButton(
                        selected = state.value,
                        onClick = { state.value = true },
                        modifier = Modifier.padding(8.dp)
                    )
                    Text("Kotlin", fontSize = 22.sp)
                }
                Row{
                    RadioButton(
                        selected = !state.value,
                        onClick = { state.value = false },
                        modifier = Modifier.padding(8.dp)
                    )
                    Text("Java", fontSize = 22.sp)
                }
            }
        }
    }
}
```



Обработка выбора варианта в группе радиокнопок

Выше приведенный пример довольно прост в том плане, что у нас только две радиокнопки - когда у одной кнопки параметр selected равен true, у другой равен false. В этом плане довольно просто задать логику переключения между радиокнопками. Однако что если у нас 3 и более переключателей? Рассмотрим следующий пример:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.selection.selectable
import androidx.compose.foundation.selection.selectableGroup
import androidx.compose.material.RadioButton
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Modifier
import androidx.compose.material.Text
import androidx.compose.ui.Alignment
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val languages = listOf("Kotlin", "Java", "Javascript", "Rust")
            val (selectedOption, onOptionSelected) = remember {
mutableStateOf(languages[0]) }

            Column(Modifier.selectableGroup()) {
                languages.forEach { text ->
                    Row( Modifier.fillMaxWidth().height(56.dp), verticalAlignment
= Alignment.CenterVertically)
                    {
                        RadioButton(
                            selected = (text == selectedOption),
                            onClick = { onOptionSelected(text) }
                        )
                        Text( text = text, fontSize = 22.sp )
                    }
                }
            }
        }
    }
}
```



В примере выше прежде всего все данные, которые будут представлять радиокнопки, помещаются в список languages:

```
val languages = listOf("Kotlin", "Java", "Javascript", "Rust")
```

Здесь четыре элемента, соответственно мы будем создавать четыре радиокнопки для каждого из этих элементов.

Далее мы получаем объект `MutableState<String>`, который необходим для и отслеживания выбранного значения:

```
val (selectedOption, onOptionSelected) = remember { mutableStateOf(languages[0]) }
```

В функцию mutableStateOf() передается первый элемент из списка, то есть по умолчанию будет выбран первый элемент списка languages.

Однако мы не просто берем объект MutableState, а раскладываем его на два компонента - selectedOption и onOptionSelected. Значение selectedOption будет представлять отслеживаемый объектом MutableState элемент списка languages. А onOptionSelected - функция типа (String) -> Unit, которая будет вызываться при изменении значения в MutableState и которая в качестве параметра будет получать новое значение.

Как и в примерах выше, чтобы задать группу выбираемых компонентов, для контейнера (в данном случае компонента Column) устанавливается модификатор selectableGroup:

```
Column(Modifier.selectableGroup()){ ..... }
```

Далее перебираем список languages с помощью функции forEach(), в которую передается функция, вызываемая для каждого перебираемого элемента:

```
languages.forEach { text ->
    Row( Modifier.fillMaxWidth().height(56.dp), verticalAlignment =
Alignment.CenterVertically)
    {
        RadioButton(
            selected = (text == selectedOption),
            onClick = { onOptionSelected(text) }
        )
        Text( text = text, fontSize = 22.sp )
    }
}
```

Фактически в данном случае за каждой строкой закрепляется определенный элемент из списка languages. И радиокнопка является выбранный, если значение selectedOption совпадает со значением элемента из списка languages, закрепленным за данным компонентом Row:

```
selected = (text == selectedOption)
```

При нажатии на компонент срабатывает функция из параметра onClick, в которой вызывается функция onOptionSelected:

```
onClick = { onOptionSelected(text) }
```

В функции onOptionSelected передается закрепленный за компонентом Row элемент из списка languages, благодаря чему изменится выбранный элемент.

Выбор всей строки

Пример выше прекрасно работает, однако имеет один недостаток: чтобы выбрать радиокноку, необходимо пальцем попасть в этот небольшой кружок, который представляет радиокнопку. Было бы гораздо лучше, если бы мы могли нажать на любой место в строке, например, на текстовую метку, и тем самым выбрать радиокноку. Для этого изменим код следующим образом:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.selection.selectable
import androidx.compose.foundation.selection.selectableGroup
import androidx.compose.material.RadioButton
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Modifier
import androidx.compose.material.Text
import androidx.compose.ui.Alignment
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val languages = listOf("Kotlin", "Java", "Javascript", "Rust")
            val (selectedOption, onOptionSelected) = remember {
mutableStateOf(languages[0]) }

            Column(Modifier.selectableGroup()) {
                languages.forEach { text ->
                    Row(
                        Modifier
                            .fillMaxWidth()
                            .height(56.dp)
                            .selectable(
                                selected = (text == selectedOption),
                                onClick = { onOptionSelected(text) }
                            ),
                        verticalAlignment = Alignment.CenterVertically
                    ) {
                        RadioButton(
                            selected = (text == selectedOption),
                            onClick = null
                        )
                        Text( text = text, fontSize = 22.sp )
                    }
                }
            }
        }
    }
}
```

Ключевым моментом здесь является установка модификатора Modifier.selectable:

```
Row(
    Modifier.selectable(
        selected = (text == selectedOption),
        onClick = { onOptionSelected(text) }
),
```

Модификатор Modifier.selectable() делает компонент (в данном случае компонент Row) выделяемым. То есть мы можем выбрать не просто радиокнопку, а всю строку. В примере выше компонент Row является выбранным, если значение selectedOption совпадает со значением элемента из списка languages, закрепленным за данным компонентом Row:

```
selected = (text == selectedOption)
```

При нажатии на компонент срабатывает функция из параметра onClick, в которой вызывается функция onOptionSelected:

```
onClick = { onOptionSelected(text) }
```

В функции onOptionSelected передается закрепленный за компонентом Row элемент из списка languages, благодаря чему изменится выбранный элемент.

Кроме того, также надо настроить радиокнопки, которые выводятся в строке Row:

```
RadioButton(
    selected = (text == selectedOption),
    onClick = null
)
```

Для выбора радиокнопки действует тот же алгоритм, что и для контейнера Row: радиокнопка выбрана, если текущий элемент из languages совпадает со значением selectedOption.

И поскольку выбор элемента обрабатывается в родительском контейнере Row, то нет смысла обрабатывать выбор элемента в радиокнопке, поэтому ее параметру onClick передается значение null.

Таким образом, внешне мы получим тот же визуальный интерфейс, только теперь для выделения радиокнопки достаточно нажать на любое место в строке.

Иконки и компоненты IconButton и IconToggleButton

Компонент IconButton представляет кликабельную иконку, по нажатию на которую можно выполнить некоторые действия. То есть, фактически он объединяет иконку и кнопку. Обычно этот компонент

применяется в панелях инстументов.

Данный компонент имеет следующие параметры:

```
@Composable
fun IconButton(
    onClick: () -> Unit,
    modifier: Modifier = Modifier,
    enabled: Boolean = true,
    interactionSource: MutableInteractionSource = remember {
MutableInteractionSource() },
    content: () -> Unit
): @Composable Unit
```

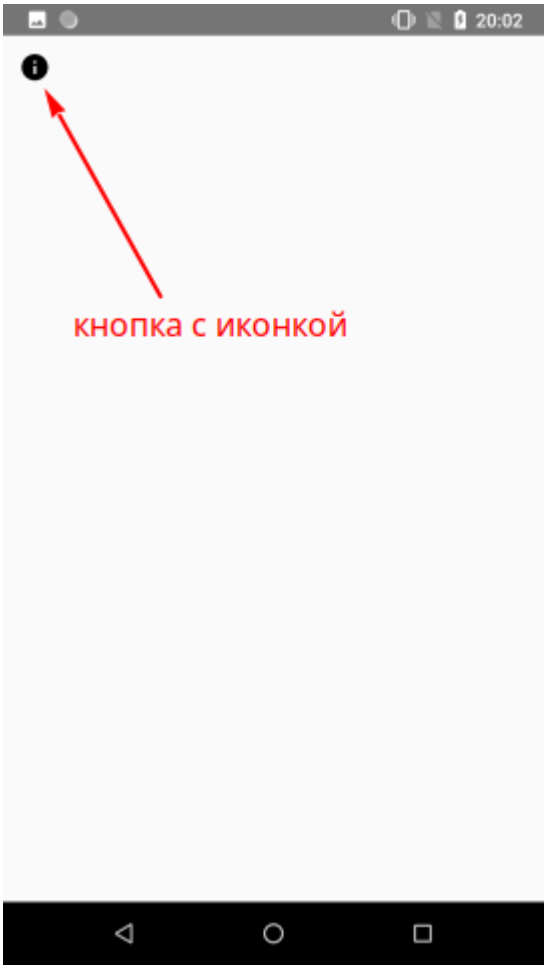
Параметры функции компонента:

- onClick: представляет функцию-обработчик нажатия кнопки
- modifier: предствляет объект Modifier, который определяет модификаторы кнопки
- enabled: значение типа Boolean устанавливает, доступна ли кнопка для нажатия. По умолчанию равно true (то есть кнопка доступна для нажатия)
- interactionSource: представляет объект типа MutableInteractionSource, который устанавливает поток взаимодействий для кнопки. Значение по умолчанию - remember { MutableInteractionSource() }
- content: содержимое кнопки - обычно представляет иконку в виде объекта типа Icon. В качестве иконки можно использовать встроенные иконки из пакета androidx.compose.material.icons.Icons. А также можно создавать и использовать свои иконки.

Определим простейшую кнопку-иконку:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.Icon
import androidx.compose.material.IconButton
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Info

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            IconButton(onClick = {  }) {
                Icon(Icons.Filled.Info, contentDescription = "Информация о
приложении")
```

```
            }
        }
    }
}
```



Компонент Icon, который задает иконку, имеет ряд версий:

```
@Composable
fun Icon(
    imageVector: ImageVector,
    contentDescription: String?,
    modifier: Modifier = Modifier,
    tint: Color = LocalContentColor.current.copy(alpha =
LocalContentAlpha.current)
): @Composable Unit

@Composable
fun Icon(
    bitmap: ImageBitmap,
    contentDescription: String?,
    modifier: Modifier = Modifier,
    tint: Color = LocalContentColor.current.copy(alpha =
LocalContentAlpha.current)
): @Composable Unit
```

```
@Composable
fun Icon(
    painter: Painter,
    contentDescription: String?,
    modifier: Modifier = Modifier,
    tint: Color = LocalContentColor.current.copy(alpha =
LocalContentAlpha.current)
): @Composable Unit
```

Во всех трех версиях первый параметр представляет рисунок, который отображается на иконке. В примере выше этому параметру передавалась встроенная иконка Icons.Filled.Info. Jetpack Compose располагает большим набором встроенных иконок. Весь этот набор можно посмотреть по ссылке Icons

Второй параметр - contentDescription задает описание иконки, которое используется сервисами accessibility. Данное описание нигде в визуальном интерфейсе не оображается и служит только для служебных целей.

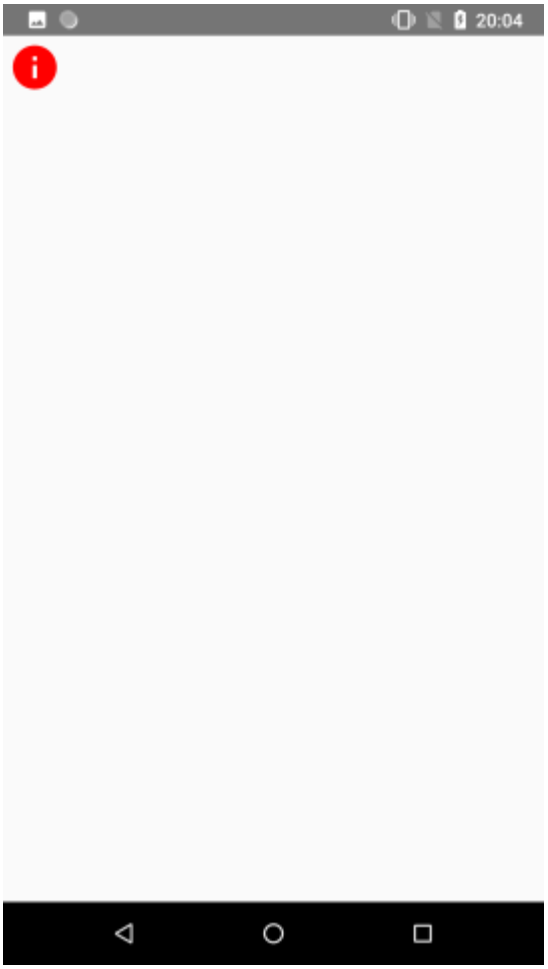
Третий параметр - modifier - объект Modifier, который, как и для других компонентов, задает модификаторы для данного компонента.

Четвертый параметр - tint устанавливает цвет иконки.

Например, немного изменить иконку:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.size
import androidx.compose.material.Icon
import androidx.compose.material.IconButton
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Info
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            IconButton(onClick = {  }) {
                Icon(
                    Icons.Filled.Info,
                    contentDescription = "Информация о приложении", modifier =
Modifier.size(80.dp),
                    tint = Color.Red
                )
            }
        }
```

```
    }
}
```



IconToggleButton

Компонент IconToggleButton фактически предоставляет IconButton, который может находиться в двух состояниях. Он принимает следующие параметры:

```
@Composable
fun IconToggleButton(
    checked: Boolean,
    onCheckedChange: (Boolean) -> Unit,
    modifier: Modifier = Modifier,
    enabled: Boolean = true,
    interactionSource: MutableInteractionSource = remember {
MutableInteractionSource() },
    content: () -> Unit
): @Composable Unit
```

Параметры функции компонента:

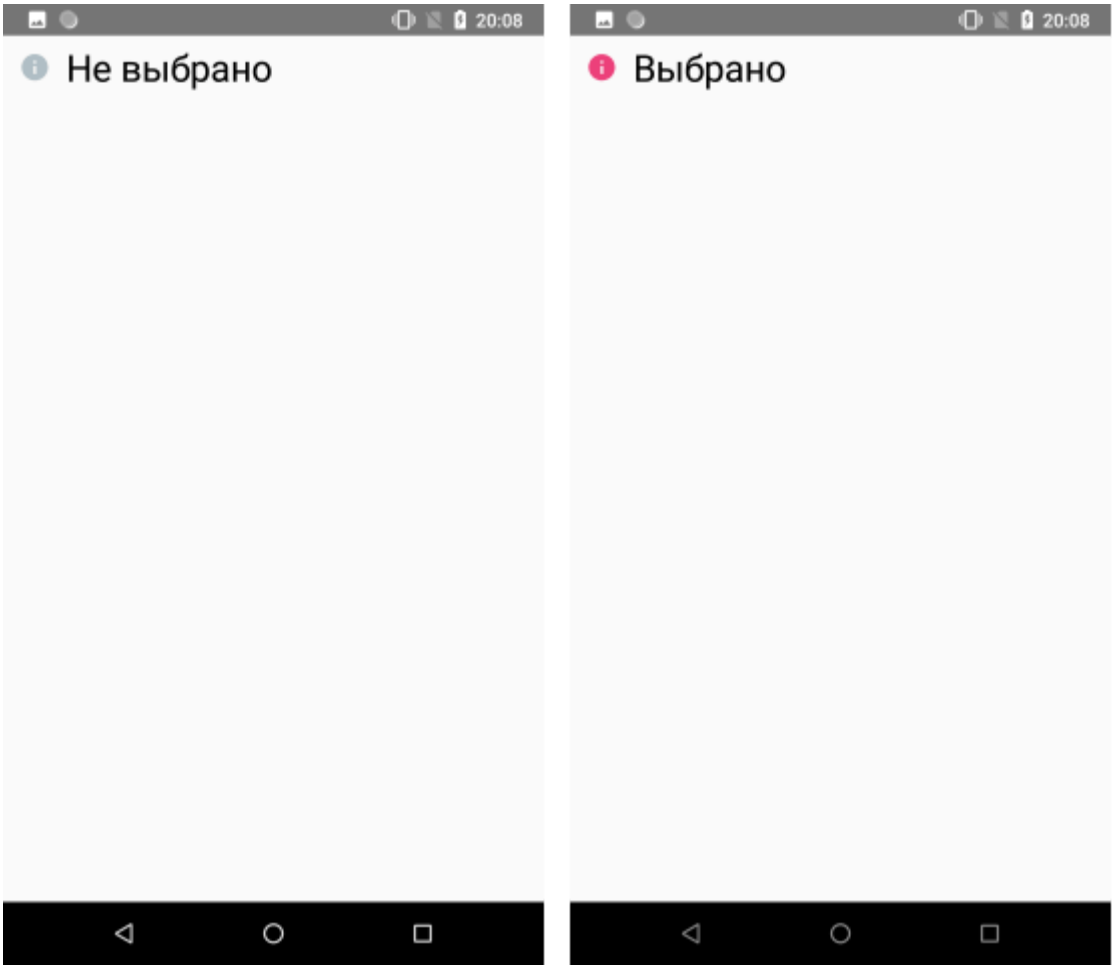
- checked: значение типа Boolean, которое указывает, будет ли компонент выбран (значение true) или нет (false)

- onCheckedChange: представляет функцию-обработчик нажатия кнопки
- modifier: представляет объект Modifier, который определяет модификаторы кнопки
- enabled: значение типа Boolean устанавливает, доступна ли кнопка для нажатия. По умолчанию равно true (то есть кнопка доступна для нажатия)
- interactionSource: представляет объект типа MutableInteractionSource, который устанавливает поток взаимодействий для кнопки. Значение по умолчанию - remember { MutableInteractionSource() }
- content: содержимое кнопки - обычно представляет иконку в виде объекта типа Icon. В качестве иконки можно использовать встроенные иконки из пакета androidx.compose.material.icons.Icons. А также можно создавать и использовать свои иконки.

Пример с IconToggleButton:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Row
import androidx.compose.material.Icon
import androidx.compose.material.IconToggleButton
import androidx.compose.material.Text
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Info
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Alignment
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val checked = remember { mutableStateOf(false) }
            Row(verticalAlignment = Alignment.CenterVertically){
                IconToggleButton(checked = checked.value, onCheckedChange = {
checked.value = it }) {
                    Icon(
                        Icons.Filled.Info,
                        contentDescription = "Информация о приложении",
                        tint = if (checked.value) Color(0xFFEC407A) else
Color(0xFFB0BEC5)
                    )
                }
                Text(
                    text = if(checked.value) "Выбрано" else "Не выбрано",
                    fontSize = 28.sp
```

```
                )
            }
        }
    }
}
```



В данном случае для отслеживания, выбрана ли кнопка, определена переменная checked, в зависимости от значения которой устанавливаются тест компонента Text и цвет иконки в IconToggleButton

FloatingActionButton и ExtendedFloatingActionButton

Компонент FloatingActionButton представляет кнопку, которая обычно располагается внизу экрана поверх основного содержимого и приглашает к некотором действию. Этот компонент имеет следующие параметры:

```
@Composable
fun FloatingActionButton(
    onClick: () -> Unit,
    modifier: Modifier = Modifier,
    interactionSource: MutableInteractionSource = remember {
MutableInteractionSource() },
    shape: Shape = MaterialTheme.shapes.small.copy(CornerSize(percent = 50)),
```

```
    backgroundColor: Color = MaterialTheme.colors.secondary,
    contentColor: Color = contentColorFor(backgroundColor),
    elevation: FloatingActionButtonElevation =
FloatingActionButtonDefaults.elevation(),
    content: () -> Unit
): @Composable Unit
```

Параметры функции компонента:

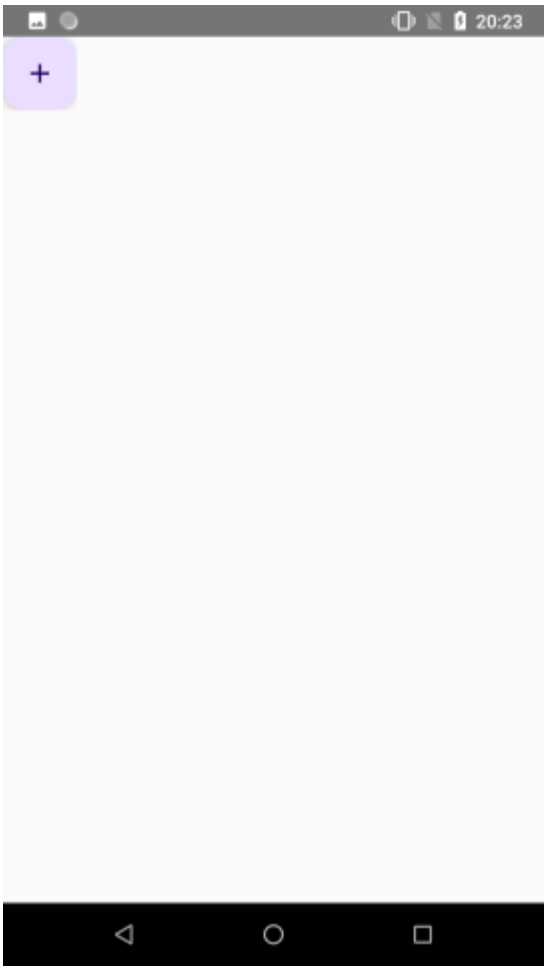
- onClick: представляет функцию-обработчик нажатия кнопки
- modifier: предствляет объект Modifier, который определяет модификаторы кнопки
- interactionSource: представляет объект типа MutableInteractionSource, который устанавливает поток взаимодействий для кнопки. Значение по умолчанию - remember { MutableInteractionSource() }
- elevation: объект типа FloatingActionButtonElevation, который определяет анимацию при нажатии кнопки. По умолчанию равно FloatingActionButtonDefaults.elevation()
- shape: объект типа Shape, который устанавливает форму кнопки. По умолчанию равно MaterialTheme.shapes.small.copy(CornerSize(percent = 50))
- backgroundColor: фоновый цвет компонента. По умолчанию имеет значение MaterialTheme.colors.secondary
- contentColor: цвет содержимого компонента. По умолчанию устанавливается функцией contentColorFor(backgroundColor), которая устанавливает цвет в зависимости от фонового цвета
- content: содержимое кнопки - обычно представляет иконку в виде объекта типа Icon. В качестве иконки можно использовать встроенные иконки из пакета androidx.compose.material.icons.Icons. А также можно создавать и использовать свои иконки.

Определим простейший элемент FloatingActionButton:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.FloatingActionButton
import androidx.compose.material.Icon
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Add

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            FloatingActionButton(onClick = { }) {
                Icon(Icons.Filled.Add, contentDescription = "Добавить")
```

```
            }
        }
    }
}
```

В данном случае кнопка никак не обрабатывает нажатия и просто отображает иконку Icons.Filled.Add из стандартного набора иконок.



Как и в обычных кнопках, мы можем обработать нажатия:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Column
import androidx.compose.material.FloatingActionButton
import androidx.compose.material.Icon
import androidx.compose.material.Text
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Add
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.unit.sp
```

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Column{
                val label = remember{ mutableStateOf("")}
                Text(text = label.value, fontSize = 28.sp)
                FloatingActionButton(onClick = {label.value = "Добавлено!" }) {
                    Icon(Icons.Filled.Add, contentDescription = "Добавить")
                }
            }
        }
    }
}
```

ExtendedFloatingActionButton

Jetpack Compose также предоставляет похожий компонент, который по сути расширяет FloatingActionButton - компонент ExtendedFloatingActionButton:

```
@Composable
fun ExtendedFloatingActionButton(
    text: () -> Unit,
    onClick: () -> Unit,
    modifier: Modifier = Modifier,
    icon: () -> Unit = null,
    interactionSource: MutableInteractionSource = remember {
MutableInteractionSource() },
    shape: Shape = MaterialTheme.shapes.small.copy(CornerSize(percent = 50)),
    backgroundColor: Color = MaterialTheme.colors.secondary,
    contentColor: Color = contentColorFor(backgroundColor),
    elevation: FloatingActionButtonElevation =
FloatingActionButtonDefaults.elevation()
): @Composable Unit
```

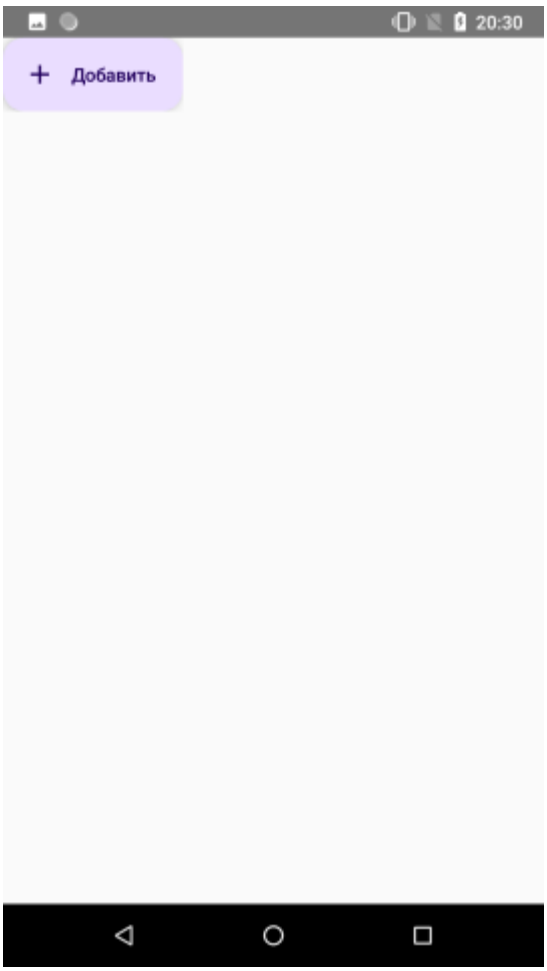
В отличие от FloatingActionButton вместо одного параметра content для установки содержимого применяются два параметра:

- text: устанавливает текст кнопки
- icon: устанавливает иконку кнопку

Простейший пример:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.ExtendedFloatingActionButton
```

```
import androidx.compose.material.Icon
import androidx.compose.material.Text
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Add

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            ExtendedFloatingActionButton(
                icon = { Icon(Icons.Filled.Add, contentDescription = "Добавить")
},
                text = { Text("Добавить") },
                onClick = {}
            )
        }
    }
}
```

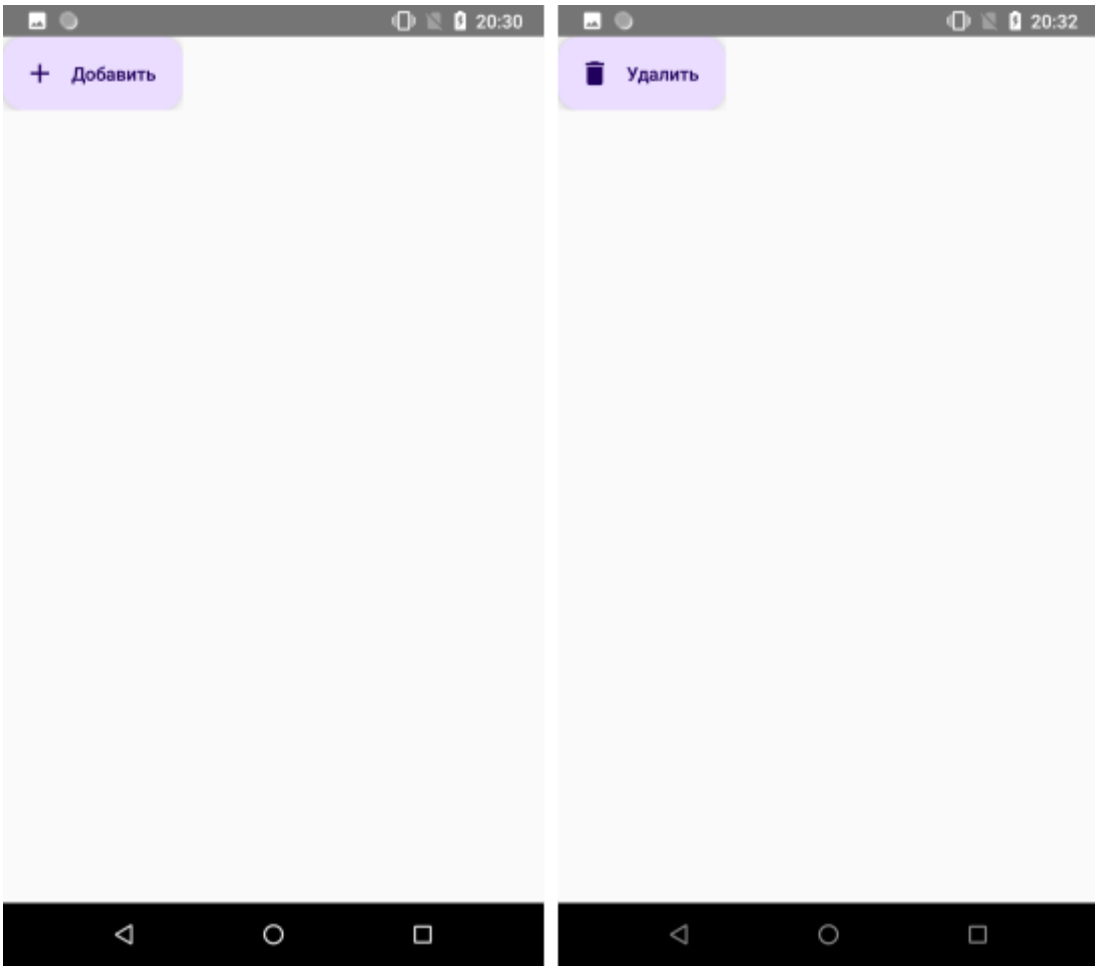


Добавим обработку нажатия:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
```

```
import androidx.activity.compose.setContent
import androidx.compose.material.ExtendedFloatingActionButton
import androidx.compose.material.Icon
import androidx.compose.material.Text
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Add
import androidx.compose.material.icons.filled.Delete
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val added = remember{ mutableStateOf(false)}
            ExtendedFloatingActionButton(
                icon = {
                    Icon(
                        if(added.value) Icons.Filled.Delete else Icons.Filled.Add,
                        contentDescription = "Добавить"
                    ) },
                text = { Text(if(added.value) "Удалить" else "Добавить") },
                onClick = {added.value = !added.value}
            )
        }
    }
}
```

В данном случае по нажатию изменяем текст и иконку кнопки:



Панели приложения TopAppBar и BottomAppBar

По умолчанию Jetpack Compose предоставляет два компонента для создания верхней и нижней панели в приложении.

TopAppBar

Компонент TopAppBar позволяет определить верхнюю панель. Он имеет следующие параметры:

```
@Composable
fun TopAppBar(
    modifier: Modifier = Modifier,
    backgroundColor: Color = MaterialTheme.colors.primarySurface,
    contentColor: Color = contentColorFor(backgroundColor),
    elevation: Dp = AppBarDefaults.TopAppBarElevation,
    contentPadding: PaddingValues = AppBarDefaults.ContentPadding,
    content: RowScope.() -> Unit
): @Composable Unit
```

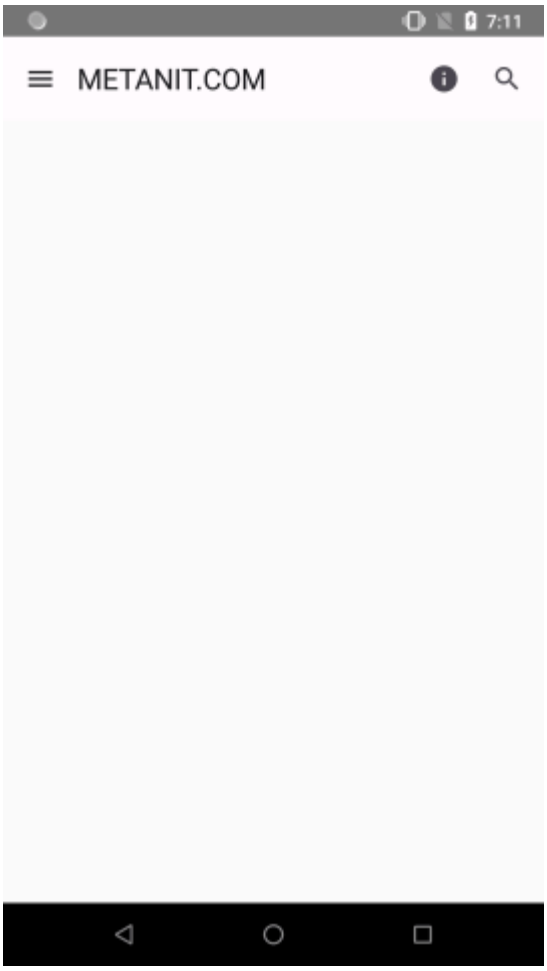
Параметры функции компонента:

- modifier: представляет объект Modifier, который определяет модификаторы панели

- backgroundColor: значение типа Color, которое задает фоновый цвет панели. По умолчанию равно MaterialTheme.colors.primarySurface
- contentColor: значение типа Color, которое устанавливает предпочтительный цвет для содержимого панели. По умолчанию предоставляет вызов contentColorFor(backgroundColor), которое для установки использует значение параметра backgroundColor
- elevation: объект типа ButtonElevation?, который определяет анимацию для содержимого панели. По умолчанию равно AppBarDefaults.TopAppBarElevation
- contentPadding: объект типа PaddingValues, который устанавливает отступы между границами панели и ее содержимым. По умолчанию равно AppBarDefaults.ContentPadding
- content: задает контент панели. По умолчанию содержимое располагается в виде строки с помощью контейнера Row. Часто в роли содержимого выступают кнопки с иконками, например, в виде объектов IconButton.

Определим простейшую панель TopAppBar:

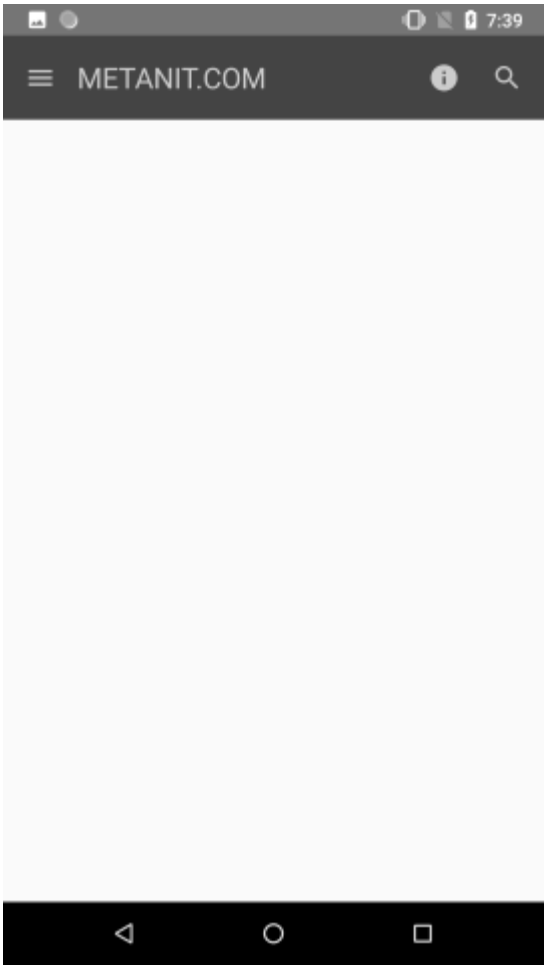
```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.*
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            TopAppBar {
                IconButton(onClick = {  }) {
                    Icon(Icons.Filled.Menu, contentDescription = "Меню")
                }
                Text("METANIT.COM", fontSize = 22.sp)
                IconButton(onClick = { }) {
                    Icon(Icons.Filled.Info, contentDescription = "Информация о
приложении")
                }
                IconButton(onClick = { }) {
                    Icon(Icons.Filled.Search, contentDescription = "Поиск")
                }
            }
        }
    }
}
```



Нередко элементы панели инстументов распределяются по всей ширине панели таким образом, что первая прижата к одному краю, а последняя - другому краю панели, даже если между ними довольно много пространства. Чтобы создать подобный эффект, можно использовать компонент Spacer, растянув его на все оставшееся пространство панели:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Spacer
import androidx.compose.material.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            TopAppBar {
                IconButton(onClick = {  }) {
                    Icon(Icons.Filled.Menu, contentDescription = "Меню")
                }
                Text("METANIT.COM", fontSize = 22.sp)
```

```
                Spacer(Modifier.weight(1f, true))

                IconButton(onClick = { }) {
                    Icon(Icons.Filled.Info, contentDescription = "Информация о
приложении")
                }
                IconButton(onClick = { }) {
                    Icon(Icons.Filled.Search, contentDescription = "Поиск")
                }
            }
        }
    }
}
```



BottomAppBar

Компонент BottomAppBar позволяет определить нижнюю панель. Он имеет следующие параметры:

```
@Composable
fun BottomAppBar(
    modifier: Modifier = Modifier,
    backgroundColor: Color = MaterialTheme.colors.primarySurface,
    contentColor: Color = contentColorFor(backgroundColor),
    cutoutShape: Shape? = null,
    elevation: Dp = AppBarDefaults.BottomAppBarElevation,
```

```
    contentPadding: PaddingValues = AppBarDefaults.ContentPadding,
    content: RowScope.() -> Unit
): @Composable Unit
```

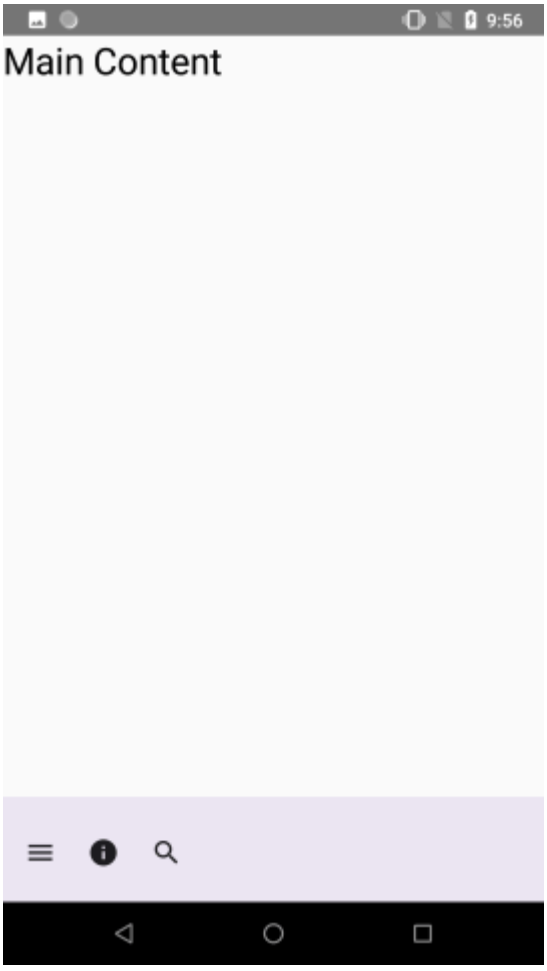
Параметры функции компонента:

- modifier: представляет объект Modifier, который определяет модификаторы панели
- backgroundColor: значение типа Color, которое задает фоновый цвет панели. По умолчанию равно MaterialTheme.colors.primarySurface
- contentColor: значение типа Color, которое устанавливает предпочтительный цвет для содержимого панели. По умолчанию предоставляет вызов contentColorFor(backgroundColor), которое для установки использует значение параметра backgroundColor
- cutoutShape: представляет объект типа Shape, который задает форму.
- elevation: объект типа ButtonElevation?, который определяет анимацию для содержимого панели. По умолчанию равно AppBarDefaults.BottomAppBarElevation
- contentPadding: объект типа PaddingValues, который устанавливает отступы между границами панели и ее содержимым. По умолчанию равно AppBarDefaults.ContentPadding
- content: задает контент панели. По умолчанию содержимое располагается в виде строки с помощью контейнера Row.

BottomAppBar применяется похожим образом, что и TopAppBar:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxHeight
import androidx.compose.foundation.layout.size
import androidx.compose.material.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {Row(verticalAlignment = Alignment.Bottom, modifier =
Modifier.fillMaxHeight()){
            BottomAppBar {
                IconButton(onClick = {  }) {
                    Icon(Icons.Filled.Menu, contentDescription = "Меню")
```

```
                }
                Spacer(Modifier.weight(1f, true))

                IconButton(onClick = { }) {
                    Icon(Icons.Filled.Info, contentDescription = "Информация о
приложении")
                }
                IconButton(onClick = { }) {
                    Icon(Icons.Filled.Search, contentDescription = "Избранное")
                }
            }
        }}
    }
}
```



В данном случае для визуального эффекта и имитации прикрепления панели BottomAppBar к нижней кромке приложения данная панель обернута в строку Row, растянутую по всей высоте с выравниванием контента по низу. Однако Jetpack Compose также предоставляет компоненты, которые позволяют автоматически разместить верхнюю и нижнюю панель соответственно вверху и внизу окна приложения и которые мы далее рассмотрим.

Scaffold

androidx.compose.material.Scaffold представляет сложный компонент, который устанавливает общую структуру страницы. И мы можем встроить в эту структуру различные компоненты. В некотором роде Scaffold выступает в роли еще одного компонента-контейнера. Определение функции компонента:

```
@Composable
fun Scaffold(
    modifier: Modifier = Modifier,
    scaffoldState: ScaffoldState = rememberScaffoldState(),
    topBar: () -> Unit = {},
    bottomBar: () -> Unit = {},
    snackbarHost: (SnackbarHostState) -> Unit = { SnackbarHost(it) },
    floatingActionButton: () -> Unit = {},
    floatingActionButtonPosition: FabPosition = FabPosition.End,
    isFloatingActionButtonDocked: Boolean = false,
    drawerContent: ColumnScope.() -> Unit = null,
    drawerGesturesEnabled: Boolean = true,
    drawerShape: Shape = MaterialTheme.shapes.large,
    drawerElevation: Dp = DrawerDefaults.Elevation,
    drawerBackgroundColor: Color = MaterialTheme.colors.surface,
    drawerContentColor: Color = contentColorFor(drawerBackgroundColor),
    drawerScrimColor: Color = DrawerDefaults.scrimColor,
    backgroundColor: Color = MaterialTheme.colors.background,
    contentColor: Color = contentColorFor(backgroundColor),
    content: (PaddingValues) -> Unit
): @Composable Unit
```

Параметры функции компонента:

- modifier: представляет объект Modifier, который определяет модификаторы компонента
- scaffoldState: : представляет объект ScaffoldState, который определяет состояние компонента Scaffold. По умолчанию равен результату вызова rememberScaffoldState()
- topBar: устанавливает верхнюю панель окна
- bottomBar: устанавливает нижнюю панель окна
- snackbarHost: представляет компонент, который содержит отображаемые на экран объекты Snackbar. Обычно это компонент SnackbarHost.
- floatingActionButton: устанавливает кнопку для выполнения некоторого действия. Обычно в качестве подобной кнопки выступает компонент FloatingActionButton
- floatingActionButtonPosition: задает позицию кнопки из параметра floatingActionButton. Представляет объект типа FabPosition и по умолчанию равно значению FabPosition.End
- isFloatingActionButtonDocked: указывает, будет ли кнопка из параметра floatingActionButton перекрывать нижнюю панель. По умолчанию равно false
- drawerContent: задает содержимое выдвижной панели Drawer

- drawerGesturesEnabled: указывает, будет ли для выдвижной панели Drawer доступна поддержка жестов. По умолчанию равно значению true (поддержка доступна)
- drawerShape: объект Shape, который задает форму для выдвижной панели Drawer. По умолчанию равно MaterialTheme.shapes.large
- drawerElevation: задает анимацию для выдвижной панели Drawer. По умолчанию равно DrawerDefaults.Elevation
- drawerBackgroundColor: задает фоновый цвет для компонента Drawer. По умолчанию равно MaterialTheme.colors.surface
- drawerContentColor: задает цвет содержимого для компонента Drawer. По умолчанию равно вызову функции contentColorFor(drawerBackgroundColor), который применяет для установки цвета параметр drawerBackgroundColor
- drawerScrimColor: задает цвет затенения содержимого в компоненте Drawer, когда этот компонент раскрыт. По умолчанию равно DrawerDefaults.scrimColor
- backgroundColor: фоновый цвет компонента Scaffold. По умолчанию равно значению MaterialTheme.colors.background
- contentColor: цвет содержимого компонента Scaffold. По умолчанию равно вызову функции contentColorFor(backgroundColor), который применяет для установки цвета параметр backgroundColor
- content: содержимое Scaffold. Представляет вызов функции типа (PaddingValues) -> Unit

Определим простейший Scaffold:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.*
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Scaffold{
                Text("Hello METANIT.COM", fontSize = 28.sp)
            }
        }
    }
}
```

В данном случае Scaffold содержит один компонент Text.

Теперь добавим в Scaffold верхнюю и нижнюю панели:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Spacer
import androidx.compose.material.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Favorite
import androidx.compose.material.icons.filled.Info
import androidx.compose.material.icons.filled.Menu
import androidx.compose.material.icons.filled.Search
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Scaffold(
                topBar = {
                    TopAppBar {
                        IconButton(onClick = { }) {Icon(Icons.Filled.Menu,
contentDescription = "Меню") }
```

```
                        Text("METANIT.COM", fontSize = 22.sp)
                        Spacer(Modifier.weight(1f, true))
                        IconButton(onClick = { }) { Icon(Icons.Filled.Search,
contentDescription = "Поиск" ) }
                    }
                },
                bottomBar = {
                    BottomAppBar{
                        IconButton(onClick = {  }) { Icon(Icons.Filled.Favorite,
contentDescription = "Избранное")}
                        Spacer(Modifier.weight(1f, true))
                        IconButton(onClick = {  }) { Icon(Icons.Filled.Info,
contentDescription = "Информация о приложении")}
                    }
                }
            ){
                Text("Hello Scaffold", fontSize = 28.sp)
            }
        }
    }
}
```

Установка кнопки floatingActionButton

Компонент Scaffold может включать кнопку, которая устанавливается через параметр floatingActionButton и по нажатию на которую выполняется некоторое действие. Обычно это кнопки

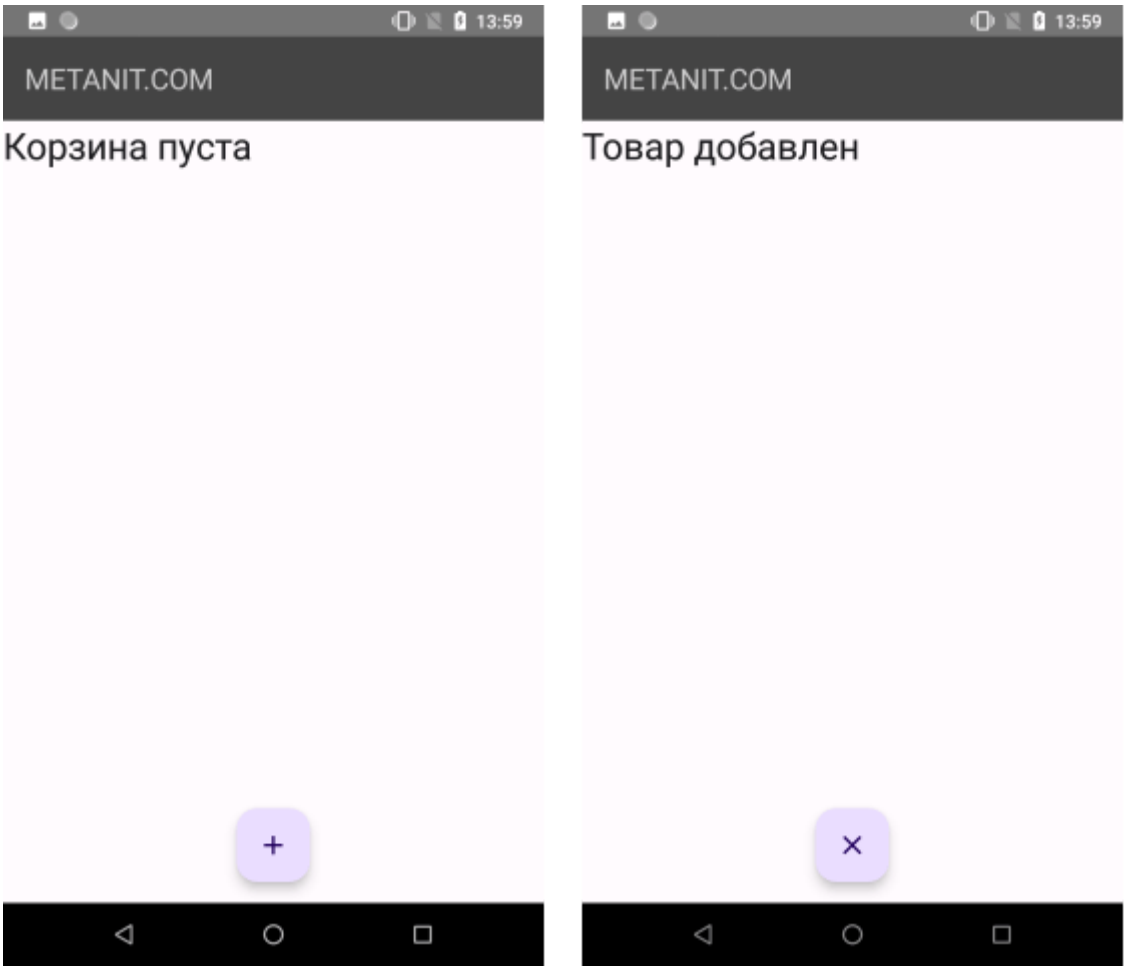
типа FloatingActionButton или ExtendedFloatingActionButton. Добавим подобную кнопку:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Spacer
import androidx.compose.material.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.*
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val isAdded = remember{ mutableStateOf(false) }
            Scaffold(
                topBar = { TopAppBar {Text("METANIT.COM", fontSize = 22.sp)} },
                bottomBar = {
                    BottomAppBar{
                        IconButton(onClick = {  }) { Icon(Icons.Filled.Menu,
contentDescription = "Меню")}
                        Spacer(Modifier.weight(1f, true))
                        IconButton(onClick = {  }) { Icon(Icons.Filled.Search,
contentDescription = "Поиск")}
                    }
                },
                floatingActionButton = {
                    FloatingActionButton(
                        content = {
                                    if(isAdded.value) Icon(Icons.Filled.Clear,
contentDescription = "Удалить")
                                    else Icon(Icons.Filled.Add, contentDescription
= "Добавить") },
                        onClick = { isAdded.value = !isAdded.value}
                    )
                },
                floatingActionButtonPosition = FabPosition.Center,
                isFloatingActionButtonDocked = true,
            ){
                Text(if(isAdded.value) "Товар добавлен" else "Корзина пуста",
fontSize = 28.sp)
            }
        }
    }
}
```

Здесь параметр floatingActionButton получает кнопку FloatingActionButton, которая переключает значение переменной isAdded с true на false и обратно. В зависимости от этого значения устанавливается иконка данной кнопки, а также текст содержимого в Scaffold.

Кроме того, с помощью параметра floatingActionButtonPosition для кнопки установлено позиционирование по центру. Этот параметр принимает одно из двух значений: FabPosition.Center (позиционирование по центру) и FabPosition.End (позиционирование по нижнем углу).

И еще один параметр - isFloatingActionButtonDocked задает прикрепление к нижней панели - значение true (при значении false кнопка была бы откреплена и располагалась бы выше нижней панели).



Всплывающие сообщения и Snackbar

Всплывающие сообщения, которые извещают пользователя о некоторых процессах в приложения и которые через некоторое время исчезают, являются обычным делом в приложении. И Jetpack Compose предоставляет для создания подобных сообщений встроенный функционал.

Snackbar

Ключевым компонентом для создания всплывающих сообщений является Snackbar, который предоставляет короткое сообщение, отображаемое внизу экрана. Данный компонент имеет две версии. Первая версия:

```
@Composable
fun Snackbar(
    modifier: Modifier = Modifier,
    action: () -> Unit = null,
    actionOnNewLine: Boolean = false,
    shape: Shape = MaterialTheme.shapes.small,
    backgroundColor: Color = SnackbarDefaults.backgroundColor,
    contentColor: Color = MaterialTheme.colors.surface,
    elevation: Dp = 6.dp,
    content: () -> Unit
): @Composable Unit
```

Параметры функции компонента:

- modifier: представляет объект Modifier, который определяет модификаторы компонента
- action: вложенный компонент (обычно текст), по нажатию на который компонент уведомляе систему, что необходимо ввыполнить некоторое действие.
- actionOnNewLine: указывает, будет ли действие располагаться на новой строке. По умолчанию равно false
- shape: объект Shape, который задает форму компонента. По умолчанию равно MaterialTheme.shapes.small
- elevation: задает анимацию по нажатию. По умолчанию равно 6.dp
- backgroundColor: фоновый цвет. По умолчанию - SnackbarDefaults.backgroundColor
- contentColor: цвет содержимого. По умолчанию - MaterialTheme.colors.surface
- content: содержимое компонента

Вторая версия Snackbar:

```
@Composable
fun Snackbar(
    snackbarData: SnackbarData,
    modifier: Modifier = Modifier,
    actionOnNewLine: Boolean = false,
    shape: Shape = MaterialTheme.shapes.small,
    backgroundColor: Color = SnackbarDefaults.backgroundColor,
    contentColor: Color = MaterialTheme.colors.surface,
    actionColor: Color = SnackbarDefaults.primaryActionColor,
    elevation: Dp = 6.dp
): @Composable Unit
```

Тут в пинципе применяются те же параметры, за исключением того, что содержимое Snackbar устанавливается с помощью параметра snackbarData, который представляет объект SnackbarData.

SnackbarData - это интерфейс, который предоставляет ряд свойств и методов для управления сообщением:

- Свойство message: текст сообщения
- Свойство actionLabel: текстовая метка, нажатие на которую будет извещать систему, что надо выполнить некотоое действие
- Свойство duration: время отображения сообщения, представляет объект SnackbarDuration
- Метод performAction(): уведомляет систему, что произошло нажатие на метку, представленную параметром actionLabel
- Метод dismiss(): уведомляет систему, что отображение сообщения завершилось без нажатия на метку из параметра actionLabel

В принципе мы можем определить Snackbar как обычный компонент и сами управлять им. Например:

```
Snackbar{
    Text("Загрузка завершена", fontSize = 22.sp)
}
```

Однако в реальности мы можем даже явным образом не создавать объект Snackbar для отображения сообщение, а воспользоваться функцией showSnackbar() объекта SnackbarHostState.

SnackbarHostState

Объект SnackbarHostState отображает или ставит в очередь для отображения компоненты Snackbar. SnackbarHostState гарантирует, что одномоментно только один объект Snackbar будет отображаться на экране.

Для отображения Snackbar в SnackbarHostState определена функция showSnackbar():

```
suspend SnackbarResult showSnackbar(
    message: String,
    actionLabel: String?,
    duration: SnackbarDuration
)
```

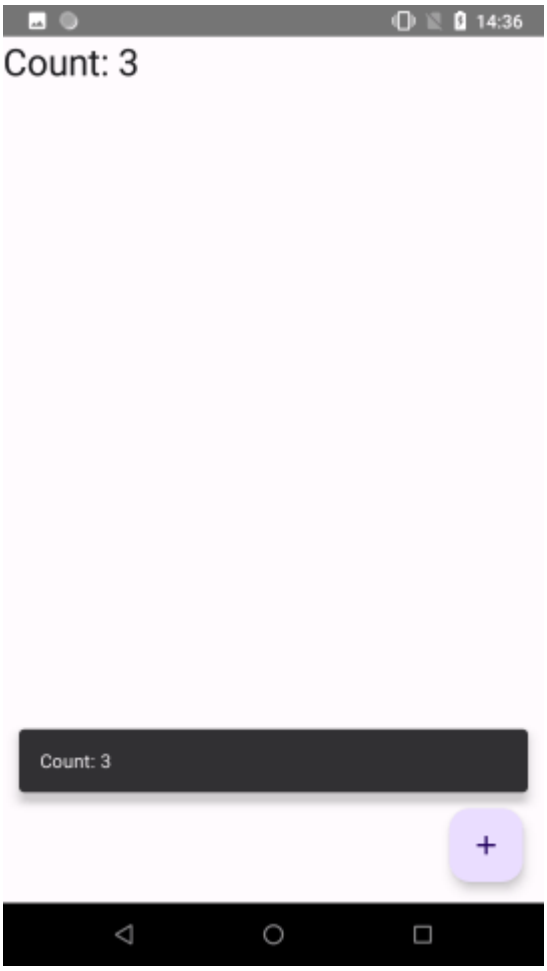
- message: отображаемое в Snackbar сообщение
- actionLabel: метка, которая отображается в виде кнопки в Snackbar
- duration: длительность отображения сообщения в виде объекта SnackbarDuration. Значение по умолчанию SnackbarDuration.Short. Другие возможные значения: SnackbarDuration.Long и SnackbarDuration.Indefinite

Фактически все эти параметры аналогичны свойствам объекта SnackbarData.

В качестве результата функция возвращает объект SnackbarResult. Он может иметь следующие значения: SnackbarResult.ActionPerformed (если была нажата метка действия в Snackbar) и SnackbarResult.Dismissed (если Snackbar был скрыт из-за действий пользователя или из-за того, что окончилось его время отображения)

Итак, используем функцию showSnackbar() для отображения всплывающего сообщения:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.*
import androidx.compose.runtime.*
import androidx.compose.ui.unit.sp
import kotlinx.coroutines.launch

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val scope = rememberCoroutineScope()
            val scaffoldState = rememberScaffoldState()
            val count = remember{ mutableStateOf(0) }
            Scaffold(
                scaffoldState = scaffoldState,
                floatingActionButton = {
                    FloatingActionButton(
                        content = {Icon(Icons.Filled.Add, contentDescription =
"Добавить")},
                        onClick = {
                            scope.launch {

scaffoldState.snackbarHostState.showSnackbar("Count: ${++count.value}")
                            }
                        }
                    )
                }
            ){
                Text("Count", fontSize = 28.sp)
            }
        }
    }
}
```



Здесь надо отметить несколько моментов. Во-первых, для обращения к объекту SnackbarHostState в Scaffold нам нужно получить состояние компонента Scaffold. Для создания объекта состояния применяется встроенная функция rememberScaffoldState()(). Далее, используя это состояние, обращаемся к SnackbarHostState и его функции showSnackbar:

```
scaffoldState.snackbarHostState.showSnackbar("Count: ${++count.value}")
```

Во-вторых, функция showSnackbar() - это suspend-функция, которую необходимо вызывать в рамках корутины. Для создания контекста корутины применяется другая встроенная функция - rememberCoroutineScope(). Далее с помощью этого контекста запускаем корутину и в ней вызывем функцию showSnackbar:

```
scope.launch {
    scaffoldState.snackbarHostState.showSnackbar("Count: ${++count.value}")
}
```

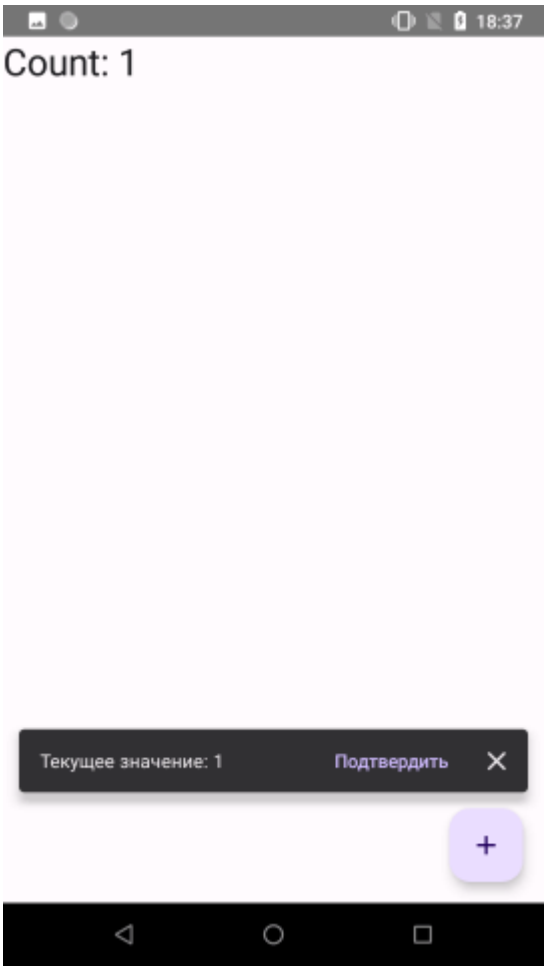
В самой функции showSnackbar увеличиваем значение в переменной count.

actionLabel

Теперь свяжем сообщение с некоторым действием. Для этого устанавим в функции showSnackbar() параметр actionLabel - фактически это просто текстовая метка, а не конкретное действие:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.*
import androidx.compose.runtime.*
import androidx.compose.ui.unit.sp
import kotlinx.coroutines.launch

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val scope = rememberCoroutineScope()
            val scaffoldState = rememberScaffoldState()
            val count = remember{ mutableStateOf(0) }
            Scaffold(
                scaffoldState = scaffoldState,
                floatingActionButton = {
                    FloatingActionButton(
                        content = {Icon(Icons.Filled.Add, contentDescription =
"Добавить")},
                        onClick = {
                            scope.launch {
                                val result =
scaffoldState.snackbarHostState.showSnackbar("Count: ${count.value}", "Click")
                                when (result) {
                                    SnackbarResult.ActionPerformed -> {
count.value++; }
                                    SnackbarResult.Dismissed -> { count.value--; }
                                }
                            }
                        }
                    )
                }
            ){
                Text("Count", fontSize = 28.sp)
            }
        }
    }
}
```

В данном случае текстовая метка в сообщении, которая приглашает к некоторому действию, имеет текст "Click" и отображается в правой части окна сообщения.



Хотя сама эта метка ничего не делает, но теперь по нажатию на эту метку функция showSnackbar будет возвращать значение SnackbarResult.ActionPerformed. Если нажатия не было, и сообщение само по себе пропало, то возвращается значение SnackbarResult.Dismissed

Получив результат функции, с помощью конструкции when мы можем проверить его значение и выполнить определенные действия. В данном случае для простоты при одном результате к переменной count добавляется единица, а при другом результате - единица вычитается.

Может сложиться впечатление, что для отображения всплывающего сообщения необходимо обрабатывать нажатия обязательно в floatingActionButton. Однако в реальности мы можем запускать отображение в любом месте Scaffold. Например, по нажатию на обычную кнопку:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.*
import androidx.compose.runtime.*
import androidx.compose.ui.unit.sp
import kotlinx.coroutines.launch

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
```

```
        setContent {
            val scope = rememberCoroutineScope()
            val scaffoldState = rememberScaffoldState()
            val count = remember{ mutableStateOf(0) }
            Scaffold(
                scaffoldState = scaffoldState
            ){
                Button(
                    onClick = {
                        scope.launch {
                            val result =
scaffoldState.snackbarHostState.showSnackbar("Count: ${count.value}", "Click")
                            if(result==SnackbarResult.ActionPerformed)
count.value++
                        }
                    }
                ){
                    Text("Click", fontSize = 28.sp)
                }
            }
        }
    }
}
```

SnackbarHost

Выше система сама определяла визуальные аспекты вслывающего сообщения, мы только задавали текстовую соотавляющую. Однако в реальности мы также можем настроить отображение сообщения. Для этого необходимо явным образом создать отображаемый объект Snackbar.

Кроме того, для установки объектов Snackbar, которые связаны со Scaffold компонент предоставляет параметр snackbarHost, который по умолчанию представляет объект SnackbarHost. SnackbarHost фактически представляет хранилище объектов Snackbar и позволяет управлять их отображением. Его определение:

```
@Composable
fun SnackbarHost(
    hostState: SnackbarHostState,
    modifier: Modifier = Modifier,
    snackbar: (SnackbarData) -> Unit = { Snackbar(it) }
): @Composable Unit
```

- hostState: представляет состояние SnackbarHost в виде объекта SnackbarHostState, который управляет отображением сообщения
- snackbar: данные для Snackbar в виде объекта SnackbarData, которыт будут отображаться

Например, настроим цветовую гамму сообщения:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.*
import androidx.compose.runtime.*
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.sp
import kotlinx.coroutines.launch

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val scope = rememberCoroutineScope()
            val scaffoldState = rememberScaffoldState()
            val count = remember{ mutableStateOf(0) }
            Scaffold(
                scaffoldState = scaffoldState,
                snackbarHost = {
                    SnackbarHost(it){ data ->
                        Snackbar(
                            snackbarData = data,
                            backgroundColor = Color(0xFF004D40),
                            contentColor = Color(0xFFB2DFDB),
                            actionOnNewLine =  true,
                            actionColor = Color(0xFF009688)
                        )
                    }
                }
            ){
                Button(
                    onClick = {
                        scope.launch {
                            val result =
scaffoldState.snackbarHostState.showSnackbar("Count: ${count.value}", "Click")
                            if(result==SnackbarResult.ActionPerformed)
count.value++
                        }
                    }
                ){
                    Text("Click", fontSize = 28.sp)
                }
            }
        }
    }
}
```

В данном случае в SnackbarHost в качестве параметра передается объект SnackbarHostState, который запускает отображение сообщения:

```
SnackbarHost(it){ data ->
    Snackbar(
        snackbarData = data,
```

То есть в данном случае it - это объект SnackbarHostState. Далее в SnackbarHost определяется объект Snackbar, в который передается объект SnackbarData через параметр data. Этот объект будет содержать отображаемое сообщение и название метки действия. Но здесь мы их никак не изменям. Мы их будет отобажать, как они определены в функции showSnackbar(), изменяется только визуальная составляющая.

Полная настройка Snackbar

Выше у Snackbar был изменен применяемый цвет. Однако мы также можем полностью изменить его содержимое:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
```

```
import androidx.compose.foundation.layout.padding
import androidx.compose.material.*
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import kotlinx.coroutines.launch

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val scope = rememberCoroutineScope()
            val scaffoldState = rememberScaffoldState()
            val count = remember{ mutableStateOf(0) }
            Scaffold(
                scaffoldState = scaffoldState,
                snackbarHost = {
                    SnackbarHost(it){ data ->
                        Snackbar(
                            modifier = Modifier.padding(10.dp),
                            backgroundColor = Color(0xFF004D40),
                            contentColor = Color(0xFFB2DFDB),
                            action = {
                                TextButton(onClick={ data.performAction() }){
                                    Text("Add click", fontSize=22.sp,
color=Color(0xFFB2DFDB))
                                }
                            }
                        ){
                            Text("Clicks count: ${count.value}", fontSize=26.sp)
                        }
                    }
                }
            ){
                Button(
                    onClick = {
                        scope.launch {
                            val result =
scaffoldState.snackbarHostState.showSnackbar("")
                            if(result==SnackbarResult.ActionPerformed)
count.value++
                        }
                    }
                ){
                    Text("Click", fontSize = 28.sp)
                }
            }
        }
    }
}
```

В данном случае полностью переопределяем внутреннее содержимое и метку действия у Snackbar. При этом параметры message и actionLabel из функции showSnackbar() могут игнорироваться. Однако даже в этом случае передаваемый параметр SnackbarData может нам понадобиться. Так, в данном случае при нажатии на метку действия (которая в примере выше представлена компонентом TextButton) вызывается метод performAction()

```
TextButton(onClick={ data.performAction() }){
    Text("Add click", fontSize=22.sp, color=Color(0xFFB2DFDB))
}
```

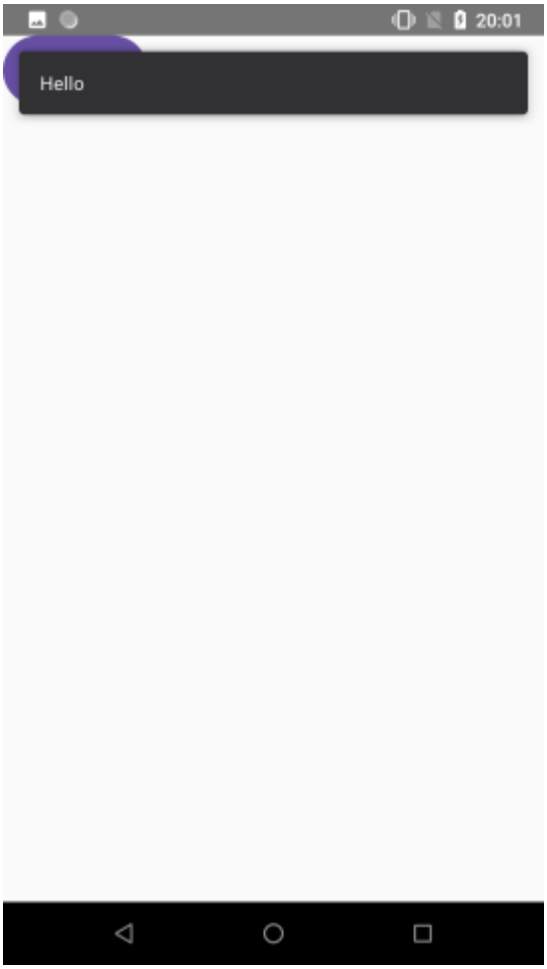
Благодаря этому система узнает, что результатом функции showSnackbar() является значение SnackbarResult.ActionPerformed.

Snackbar вне Scaffold

Может возникнуть вопрос, а можно ли использовать Snackbar вне Scaffold? В принципе можно, но в этом случае придется приложить дополнительные усилия, написать дополнительный код для позиционирования сообщения:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
```

```
import androidx.activity.compose.setContent
import androidx.compose.material.*
import androidx.compose.runtime.*
import androidx.compose.ui.unit.sp
import kotlinx.coroutines.launch

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val scope = rememberCoroutineScope()
            val snackbarHostState = remember { mutableStateOf(SnackbarHostState())
}
            Button(
                onClick = {
                    scope.launch {
                        snackbarHostState.value.showSnackbar("Hello")
                    }
                }
            ) {
                Text("Click", fontSize = 28.sp)
            }
            SnackbarHost(snackbarHostState.value)
        }
    }
}
```



Выдвижная панель drawer в Scaffold

Jetpack Compose позволяет определить для компонента Scaffold панель Drawer и управлять ею. Drawer - это выдвижная панель, которая может отображаться и быть скрыта.

Фактически панель Drawer в Scaffold - это произвольный набор компонентов который помещается в компонент Column с помощью параметра drawerContent:

```
drawerContent: ColumnScope.() -> Unit = null,
```

Для управления выдвижной панелью в Jetpack Compose определен класс DrawerState, который предоставляет ряд свойств и методов. Отметим некоторые:

- Метод close() скрывает выдвижную панель
- Метод open() отобажает выдвижную панель
- Свойство isOpen возвращает true, если выдвижная панель отображается
- Свойство isClosed возвращает true, если выдвижная панель скрыта

Определим простейшую панель Drawer:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.*
import androidx.compose.runtime.rememberCoroutineScope
import androidx.compose.ui.unit.sp
import kotlinx.coroutines.launch

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val scaffoldState = rememberScaffoldState()
            val scope = rememberCoroutineScope()

            Scaffold(
                scaffoldState = scaffoldState,
                drawerContent={
                    Text("Пункт меню 1", fontSize = 28.sp)
                    Text("Пункт меню 2", fontSize = 28.sp)
                    Text("Пункт меню 3", fontSize = 28.sp)
                }
            ){
                Button(onClick = {
```

```
                    scope.launch{
                        scaffoldState.drawerState.open()
                    }
                }) {
                    Text("Меню", fontSize = 28.sp)
                }
            }
        }
    }
}
```

Здесь выдвижная панель предствлена набором из трех компонентов Text:

```
drawerContent={
    Text("Пункт меню 1", fontSize = 28.sp)
    Text("Пункт меню 2", fontSize = 28.sp)
    Text("Пункт меню 3", fontSize = 28.sp)
}
```

По умолчанию эта панель скрыта. Чтобы ее открыть, нам надо обратиться к функции open() объекта DrawerState. Внутри Scaffold это состояние хранится внутри состояния самого Scaffold - ScaffoldState. Поэтому сначала получаем состояние самого Scaffold с помощью встроенной функции rememberScaffoldState():

```
val scaffoldState = rememberScaffoldState()
```

Далее из состояния Scaffold мы можем обратиться к состоянию drawer и его методам и свойствам:

```
scaffoldState.drawerState.open()
```

Но поскольку drawerState.open() - это suspend-функция, ее необходимо запускать в рамках корутины. И для этого вначале создаем контекст корутины с помощью встроенной функции rememberCoroutineScope():

```
val scope = rememberCoroutineScope()
```

И затем в рамках запущеной корутины запускаем отображение выдвижной панели:

```
scope.launch{ scaffoldState.drawerState.open()}
```

Аналогично с помощью функции scaffoldState.drawerState.close() можно программно скрыть панель.

Взаимодействие с Drawer

Выше была определена выдвижная панель с тремя условными пунктами меню, представленными компонентами Text. Посмотрим, как мы можем взаимодействовать с этой панелью, например, выбирать тот или иной пункт меню:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Column
import androidx.compose.material.*
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.rememberCoroutineScope
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.sp
import kotlinx.coroutines.launch

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val scaffoldState = rememberScaffoldState()
            val scope = rememberCoroutineScope()
            val selectedItem = remember{ mutableStateOf("")}
            val items = listOf("Главная", "Контакты", "О приложении")
            Scaffold(
                scaffoldState = scaffoldState,
                drawerContent={
                    for (item in items) {
                        Text(
                            item,
                            fontSize = 28.sp,
                            modifier = Modifier.clickable {
                                selectedItem.value = item
                                scope.launch{ scaffoldState.drawerState.close() }
                            }
                        )
                    }
                }
            ){
                Column{
                    Button(onClick = {
                        scope.launch{ scaffoldState.drawerState.open() }
                    }) {
                        Text("Меню", fontSize = 28.sp)
                    }
```

```
                    Text("Выбран пункт: ${selectedItem.value}", fontSize = 28.sp)
                }
            }
        }
    }
}
```

Здесь выбранный пункт хранится в переменной selectedItem. При создании меню к каждому компоненту Text добавляется модификатор clickable, в котором обрабатывается нажатие посредством передачи переменной selectedItem текущего элемента. После чего выдвижная панель скрывается.

Настройка визуального вида drawer

Scaffold предоставляет ряд дополнительных параметров для управления панелью drawer:

- drawerGesturesEnabled: указывает, будет ли для выдвижной панели доступна поддержка жестов. По умолчанию равно значению true (поддержка доступна)
- drawerShape: объект Shape, который задает форму для выдвижной панели. По умолчанию равно MaterialTheme.shapes.large
- drawerElevation: задает анимацию для выдвижной панели. По умолчанию равно DrawerDefaults.Elevation
- drawerBackgroundColor: задает фоновый цвет для выдвижной панели. По умолчанию равно MaterialTheme.colors.surface
- drawerContentColor: задает цвет содержимого для выдвижной панели. По умолчанию равно вызову функции contentColorFor(drawerBackgroundColor), который применяет для установки цвета параметр drawerBackgroundColor
- drawerScrimColor: задает цвет затенения Scaffold, когда панель открыта. По умолчанию равно DrawerDefaults.scrimColor

Например, изменение цветовой гаммы:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Column
import androidx.compose.material.*
import androidx.compose.runtime.rememberCoroutineScope
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.sp
import kotlinx.coroutines.launch

class MainActivity : ComponentActivity() {
```

```
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val scaffoldState = rememberScaffoldState()
            val scope = rememberCoroutineScope()
            val items = listOf("Главная", "Контакты", "О приложении")
            Scaffold(
                scaffoldState = scaffoldState,
                drawerBackgroundColor = Color(0xFFE1F5FE),
                drawerContentColor = Color(0xFF0277BD),
                drawerScrimColor =  Color(0x99E0F7FA),
                drawerContent={
                    items.forEach{item -> Text( item, fontSize = 28.sp)}
                }
            ){
                Column{
                    Button(onClick = {
                        scope.launch{ scaffoldState.drawerState.open() }
                    }) {
                        Text("Меню", fontSize = 28.sp)
                    }
                }
            }
        }
    }
}
```

Slider

Компонент Slider представляет шкалу с некоторым диапазоном числовых значений, из которых мы можем выбрать одно из значений. Типичный прример подобной шкалы - элемент для установки громкости.

Функция компонента имеет следующее определение:

```
@Composable
fun Slider(
    value: Float,
    onValueChange: (Float) -> Unit,
    modifier: Modifier = Modifier,
    enabled: Boolean = true,
    valueRange: ClosedFloatingPointRange<float> = 0f..1f,
    steps: Int = 0,
    onValueChangeFinished: () -> Unit = null,
    interactionSource: MutableInteractionSource = remember {
MutableInteractionSource() },
    colors: SliderColors = SliderDefaults.colors()
): @Composable Unit
```

Параметры компонента:

- value: текущее значение слайдера в виде объекта Float
- onValueChange: функция обработки изменения введенного значения. Представляет функцию типа (Float) -> Unit, в которую в качестве параметра передается новое значение
- modifier: объект типа Modifier, который задает модификаторы компонента
- enabled: устанавливает, будет ли слайдер доступен для ввода. Представляет значение типа Boolean. По умолчанию равно true, то есть поле доступно для ввода
- valueRange: устанавливает диапазон доступных для выбора значений в виде объекта ClosedFloatingPointRange. По умолчанию равно диапазону 0f..1f.
- steps: количество делений на диапазоне valueRange. Если это значение больше 0, то значения равномено распределяются по шкале, и мы можем выбрать одно из этих делений. Если это значение равно 0, то мы можем выбрать любое значение на шкале. Значение по умолчанию - 0
- onValueChangeFinished: устанавливает функцию типа () -> Unit, которая вызывается после завершения установки значения слайдера. Эта функция позволяет указать, что пользователь завершил изменение значения слайдера. Значение по умолчанию - null
- interactionSource: объект MutableInteractionSource, который задает поток взаимодействий для поля ввода. Значение по умолчанию - remember { MutableInteractionSource() }
- colors: объект SliderColors, который задает цвета для поля ввода. Значение по умолчанию - SliderDefaults.colors()

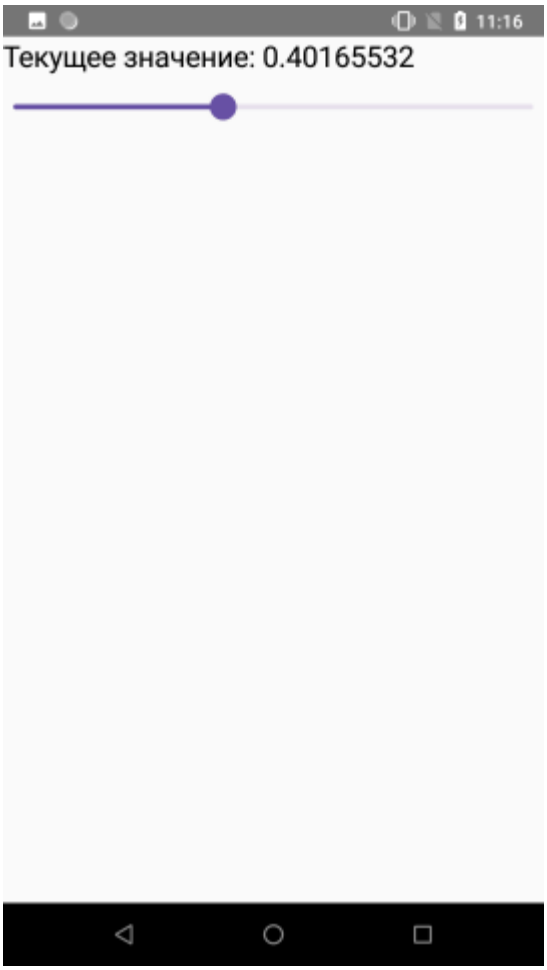
Определим простейший слайдер:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Column
import androidx.compose.material.Slider
import androidx.compose.material.Text
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            var sliderPosition by remember{mutableStateOf(0f)}
            Column{
                Text(text = "Текущее значение: ${sliderPosition}", fontSize =
```

```
    22.sp)
                    Slider(
                        value = sliderPosition,
                        onValueChange = { sliderPosition = it }
                    )
                }
            }
        }
    }
```

В данном случае для хранения значения слайдера определена переменная sliderPosition. По умолчанию она равна 0.

Для слайдера необходимо задать как минимум два параметра: value и onValueChange. Параметр value привязан к значению переменной sliderPosition, а в функции onValueChange получаем новое значение и переустанавливаем значение переменной sliderPosition.

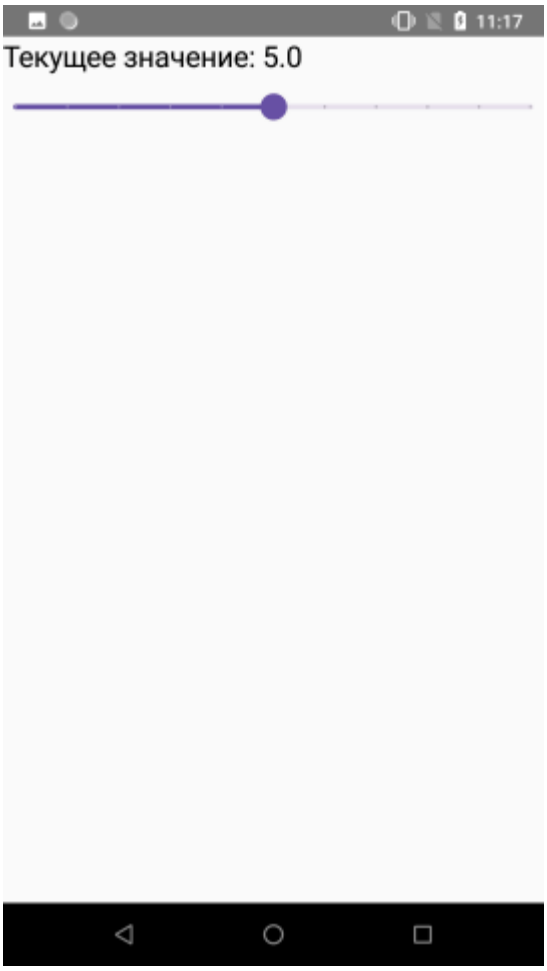


В примере выше не указан диапазон, поэтому по умолчанию мы можем выбирать значения на диапазоне от 0 до 1. Причем, так как не указано количество делений, это может быть любое значение, напимер, 0.51574075. Теперь явно укажем диапазон и количество делений:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
```

```
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Column
import androidx.compose.material.Slider
import androidx.compose.material.Text
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            var sliderPosition by remember{mutableStateOf(0f)}
            Column{
                Text(text = "Текущее значение: ${sliderPosition}", fontSize =
22.sp)
                Slider(
                    value = sliderPosition,
                    valueRange = 0f..10f,
                    steps = 9,
                    onValueChange = { sliderPosition = it }
                )
            }
        }
    }
}
```

В данном случае количество делений - 9, а диапазон - от 1 до 10. Поэтому на слайдере мы можем выбрать любое число из диапазона [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]



Цветовая гамма

По умолчанию слайдер использует цвета, устанавливаемые компонентом :

```
@Composable
fun colors(
    thumbColor: Color = MaterialTheme.colors.primary,
    disabledThumbColor: Color = MaterialTheme.colors.onSurface
            .copy(alpha = ContentAlpha.disabled)
            .compositeOver(MaterialTheme.colors.surface),
    activeTrackColor: Color = MaterialTheme.colors.primary,
    inactiveTrackColor: Color = activeTrackColor.copy(alpha = InactiveTrackAlpha),
    disabledActiveTrackColor: Color = MaterialTheme.colors.onSurface.copy(alpha =
DisabledActiveTrackAlpha),
    disabledInactiveTrackColor: Color = disabledActiveTrackColor.copy(alpha =
DisabledInactiveTrackAlpha),
    activeTickColor: Color = contentColorFor(activeTrackColor).copy(alpha =
TickAlpha),
    inactiveTickColor: Color = activeTrackColor.copy(alpha = TickAlpha),
    disabledActiveTickColor: Color = activeTickColor.copy(alpha =
DisabledTickAlpha),
    disabledInactiveTickColor: Color = disabledInactiveTrackColor
            .copy(alpha = DisabledTickAlpha)
): @Composable SliderColors
```

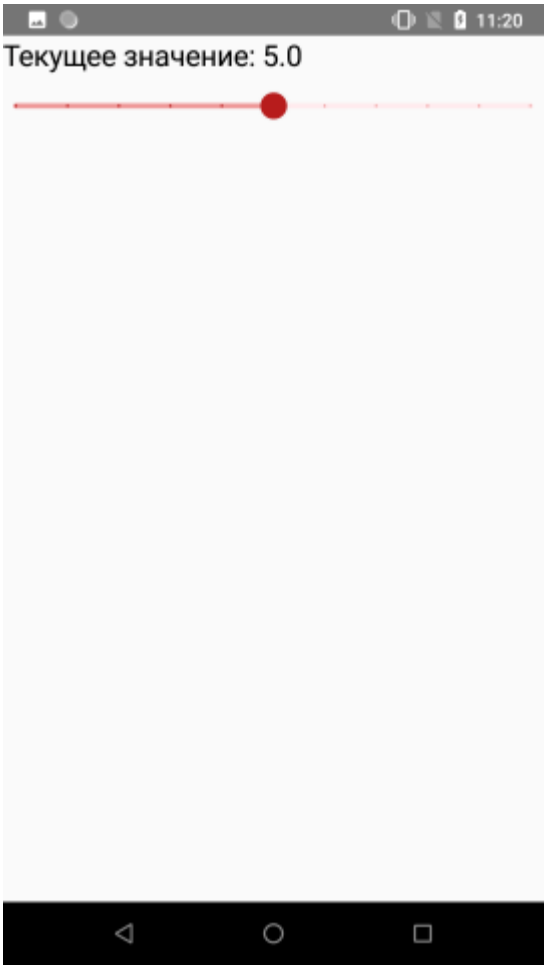
Параметры компонента:

- thumbColor: цвет ползунка
- disabledThumbColor: цвет ползунка, когда слайдер недоступен для взаимодействия
- activeTrackColor: цвет шкалы до ползунка
- inactiveTrackColor: цвет шкалы после ползунка
- disabledActiveTrackColor: цвет шкалы до ползунка, когда слайдер недоступен для взаимодействия
- disabledInactiveTrackColor: цвет шкалы после ползунка, когда слайдер недоступен для взаимодействия
- activeTickColor: цвет делений шкалы до ползунка
- inactiveTickColor: цвет делений шкалы после ползунка
- disabledActiveTickColor: цвет делений шкалы до ползунка, когда слайдер недоступен для взаимодействия
- disabledInactiveTickColor: цвет делений шкалы после ползунка, когда слайдер недоступен для взаимодействия

Изменим цветовую гамму слайдера:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Column
import androidx.compose.material.Slider
import androidx.compose.material.SliderDefaults
import androidx.compose.material.Text
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            var sliderPosition by remember{mutableStateOf(0f)}
            Column{
                Text(text = "Текущее значение: ${sliderPosition}", fontSize =
22.sp)
                Slider(
                    value = sliderPosition,
```

```
                    valueRange = 0f..10f,
                    steps = 9,
                    onValueChange = { sliderPosition = it },
                    colors = SliderDefaults.colors(
                        thumbColor = Color(0xFFB71C1C),
                        activeTrackColor = Color(0xFFEF9A9A),
                        inactiveTrackColor = Color(0xFFFFEBEE),
                        inactiveTickColor = Color(0xFFEF9A9A),
                        activeTickColor = Color(0xFFB71C1C)
                    )
                )
            }
        }
    }
}
```



Переключатель Switch

Компонент Switch представляет переключатель, который может находиться в двух состояниях: отмеченном и неотмеченном. Он имеет следующие параметры:

```
@Composable
fun Switch(
    checked: Boolean,
```

```
    onCheckedChange: (Boolean) -> Unit,
    modifier: Modifier = Modifier,
    enabled: Boolean = true,
    interactionSource: MutableInteractionSource = remember {
MutableInteractionSource() },
    colors: SwitchColors = SwitchDefaults.colors()
): @Composable Unit
```

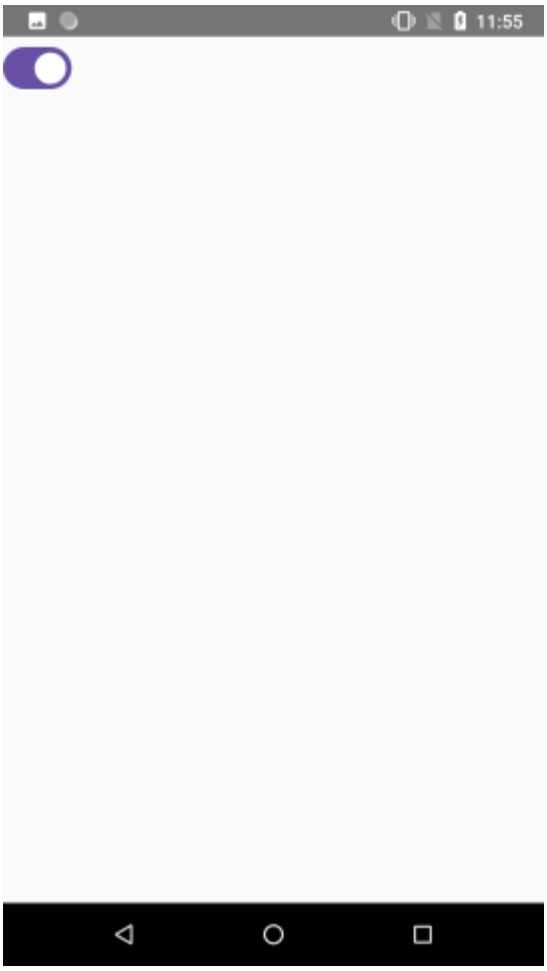
Параметры функции компонента:

- checked: указывает, отмечен ли переключатель. Если равно true, то отмечен
- onCheckedChange: функция обработки нажатия на компонент. Представляет функцию типа (Boolean) -> Unit, в которую в качестве параметра передается новое состояние переключателя
- modifier: объект типа Modifier, который задает модификаторы компонента
- enabled: устанавливает, будет ли слайдер доступен для ввода. Представляет значение типа Boolean. По умолчанию равно true, то есть поле доступно для ввода
- interactionSource: объект MutableInteractionSource, который задает поток взаимодействий для поля ввода. Значение по умолчанию - remember { MutableInteractionSource() }
- colors: объект SwitchColors, который задает цвета для поля ввода. Значение по умолчанию - SwitchDefaults.colors()

Простейший Switch:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.*
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val checkedState = remember { mutableStateOf(true) }
            Switch(
                checked = checkedState.value,
                onCheckedChange = { checkedState.value = it }
            )
        }
    }
}
```

В данном случае состояние компонента Switch привязано к значению переменной checkedState. В функции параметра onCheckedChange изменяем значение этой переменной, передавая ей новое значение.

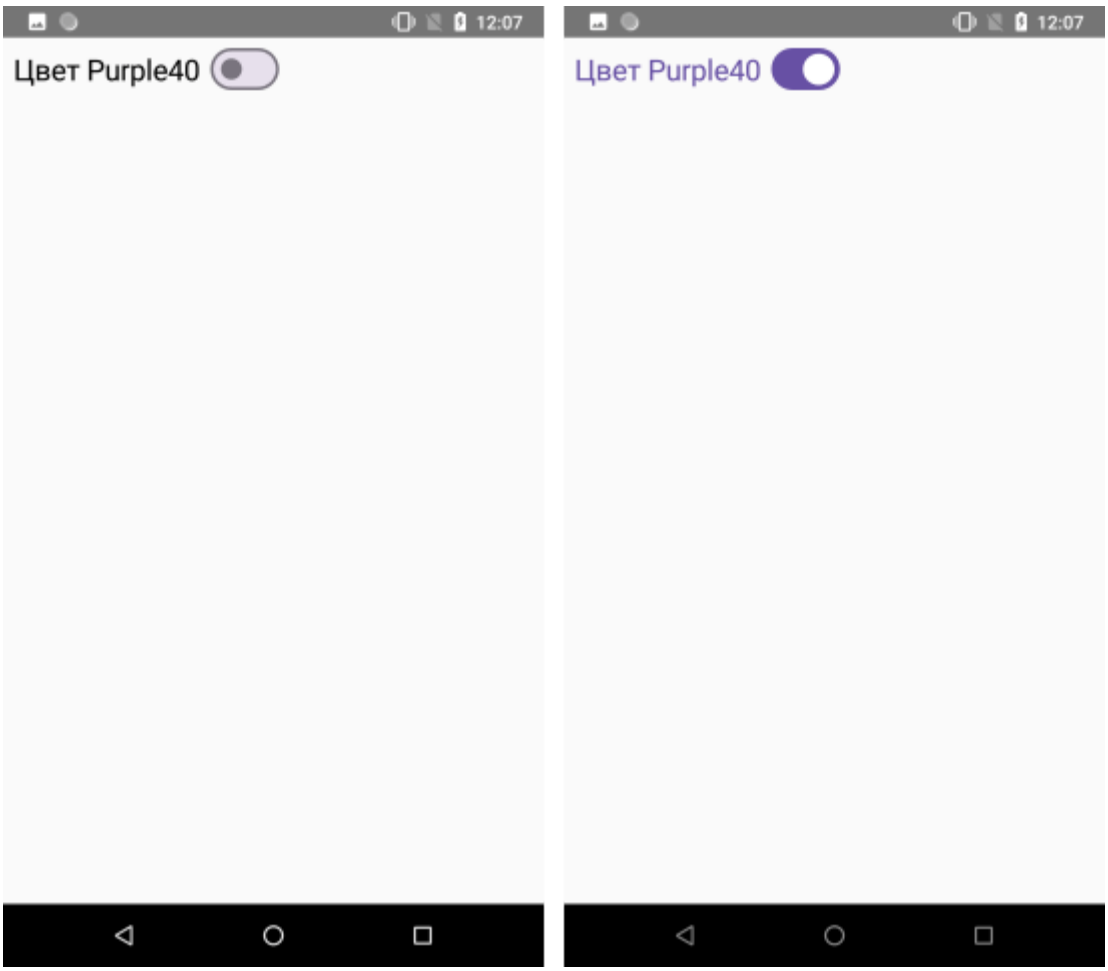


Обычно рядом со Switch идет текстовая метка, которая говорит о предназначении переключателя:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.material.*
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
```

```
            val checkedState = remember { mutableStateOf(false) }
            val textColor = remember { mutableStateOf(Color.Unspecified) }
            Column(modifier = Modifier.fillMaxSize().padding(10.dp)){
                Row (verticalAlignment = Alignment.CenterVertically ){
                    Text("Зеленый цвет", fontSize = 22.sp, color =
textColor.value)
                    Switch(
                        checked = checkedState.value,
                        onCheckedChange = {
                            checkedState.value = it
                            if(checkedState.value) textColor.value =
Color(0xff00695C)
                            else textColor.value = Color.Unspecified
                        }
                    )
                }
            }
        }
    }
}
```

В данном случае мы переключаем значение переменной textColor - если Switch находится в отмеченном состоянии, то она получает зеленый цвет. Для отображения изменения этой переменной ее значение привязано к параметру color компонента Text:



Цветовая гамма

По умолчанию Switch использует цвета, устанавливаемые компонентом SwitchDefaults.colors:

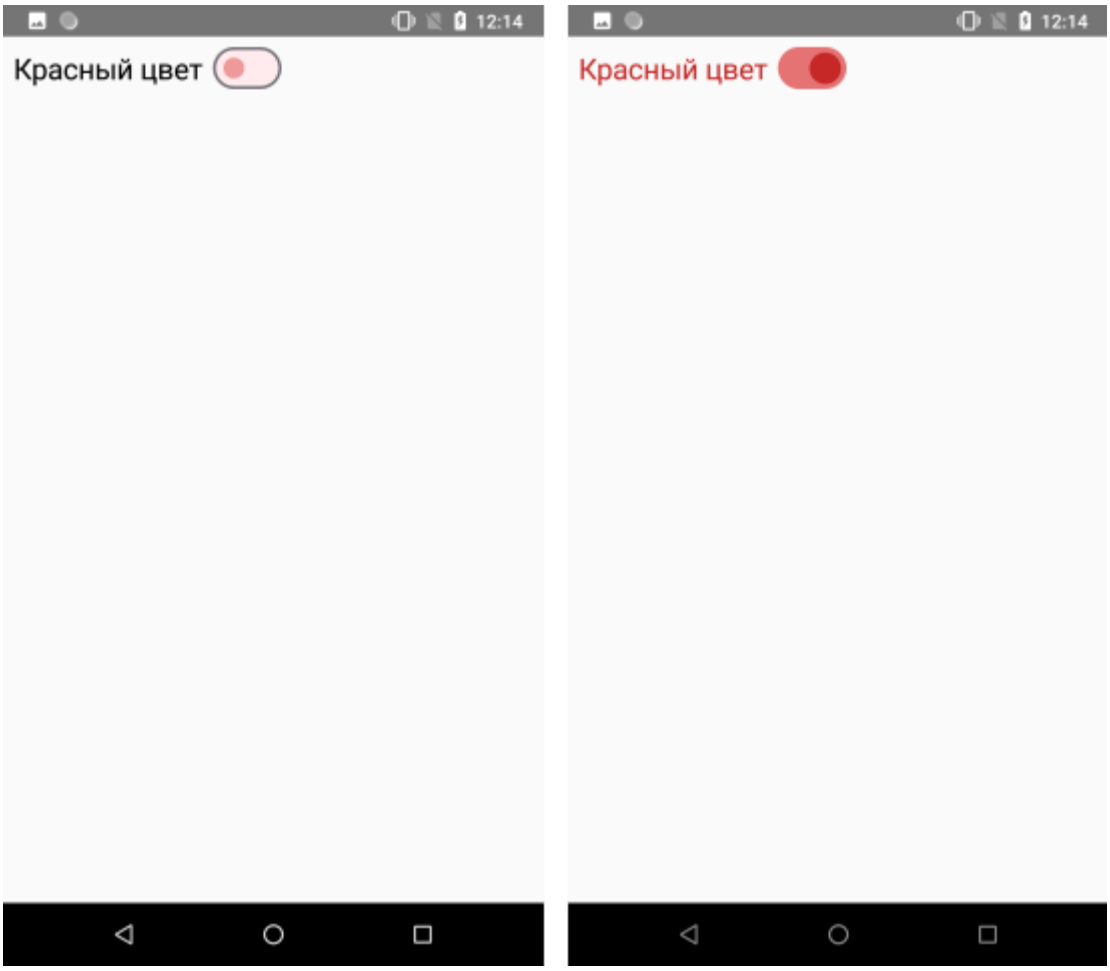
```
@Composable
fun colors(
    checkedThumbColor: Color = MaterialTheme.colors.secondaryVariant,
    checkedTrackColor: Color = checkedThumbColor,
    checkedTrackAlpha: Float = 0.54f,
    uncheckedThumbColor: Color = MaterialTheme.colors.surface,
    uncheckedTrackColor: Color = MaterialTheme.colors.onSurface,
    uncheckedTrackAlpha: Float = 0.38f,
    disabledCheckedThumbColor: Color = checkedThumbColor
            .copy(alpha = ContentAlpha.disabled)
            .compositeOver(MaterialTheme.colors.surface),
    disabledCheckedTrackColor: Color = checkedTrackColor
            .copy(alpha = ContentAlpha.disabled)
            .compositeOver(MaterialTheme.colors.surface),
    disabledUncheckedThumbColor: Color = uncheckedThumbColor
            .copy(alpha = ContentAlpha.disabled)
            .compositeOver(MaterialTheme.colors.surface),
    disabledUncheckedTrackColor: Color = uncheckedTrackColor
            .copy(alpha = ContentAlpha.disabled)
            .compositeOver(MaterialTheme.colors.surface)
): @Composable SwitchColors
```

Параметры компонента:

- checkedThumbColor: цвет бегунка в отмеченном состоянии
- checkedTrackColor: цвет переключателя в отмеченном состоянии
- checkedTrackAlpha: коэффициент прозрачности при отмеченном состоянии
- uncheckedThumbColor: цвет бегунка в неотмеченном состоянии
- uncheckedTrackColor: цвет переключателя в неотмеченном состоянии
- hecheckedTrackAlpha: коэффициент прозрачности при неотмеченном состоянии
- disabledCheckedTrackColor: цвет переключателя в отмеченном состоянии, когда он недоступен для взаимодействия
- disabledUncheckedTrackColor: цвет переключателя в неотмеченном состоянии, когда он недоступен для взаимодействия
- disabledCheckedThumbColor: цвет бегунка в отмеченном состоянии, когда переключатель недоступен для взаимодействия
- disabledUncheckedThumbColor: цвет бегунка в неотмеченном состоянии, когда переключатель недоступен для взаимодействия

Изменим цветовую гамму переключателя:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.*
import androidx.compose.material.*
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val checkedState = remember { mutableStateOf(false) }
            val textColor = remember { mutableStateOf(Color.Unspecified) }
            Column(modifier = Modifier.fillMaxSize().padding(10.dp)){
                Row (verticalAlignment = Alignment.CenterVertically ){
                    Text("Красный цвет", fontSize = 22.sp, color =
textColor.value)
                    Switch(
                        checked = checkedState.value,
                        onCheckedChange = {
                            checkedState.value = it
                            if(checkedState.value) textColor.value =
Color(0xFFC62828)
                            else textColor.value = Color.Unspecified
                        },
                        colors = SwitchDefaults.colors(
                            checkedThumbColor = Color(0xFFC62828),
                            checkedTrackColor = Color(0xFFE57373),
                            uncheckedThumbColor = Color(0xFFEF9A9A),
                            uncheckedTrackColor = Color(0xFFFFEBEE)
                        )
                    )
                }
            }
        }
    }
}
```



Диалоговые окна AlertDialog

Компонент AlertDialog представляет диалоговое окно, которое прерывает работу пользователя, отображая некоторую важную информацию. Этот компонент имеет следующее определение:

```
@Composable
fun AlertDialog(
    onDismissRequest: () -> Unit,
    buttons: () -> Unit,
    modifier: Modifier = Modifier,
    title: () -> Unit = null,
    text: () -> Unit = null,
    shape: Shape = MaterialTheme.shapes.medium,
    backgroundColor: Color = MaterialTheme.colors.surface,
    contentColor: Color = contentColorFor(backgroundColor),
    properties: DialogProperties = DialogProperties()
): @Composable Unit
```

Параметры компонента:

- onDismissRequest: представляет функцию типа () -> Unit, которая выполняется, когда пользователь пытается закрыть диалоговое окно, нажав на область вне окна или на кнопку Назад.

- buttons: представляет функцию типа () -> Unit, в которой устанавливается разметка для кнопок диалогового окна.
- modifier: объект типа Modifier, который задает модификаторы компонента
- title: устанавливает заголовок диалогового окна
- text: устанавливает текст диалогового окна
- shape: устанавливает форму диалогового окна в виде объекта Shape. Значение по умолчанию - MaterialTheme.shapes.medium
- backgroundColor: устанавливает фоновый цвет окна в виде объекта Color. Значение по умолчанию - MaterialTheme.colors.surface
- contentColor: устанавливает цвет текста окна в виде объекта Color. Значение по умолчанию задается вызовом функции contentColorFor(backgroundColor), которая для установки использует параметр backgroundColor
- properties: дополнительные свойства для настройки окна, которые представляют объект DialogProperties. Значение по умолчанию - вызов DialogProperties()

Для создания диалогового окна достаточно установить параметры onDismissRequest и buttons.

Определим простейшее диалоговое окно:

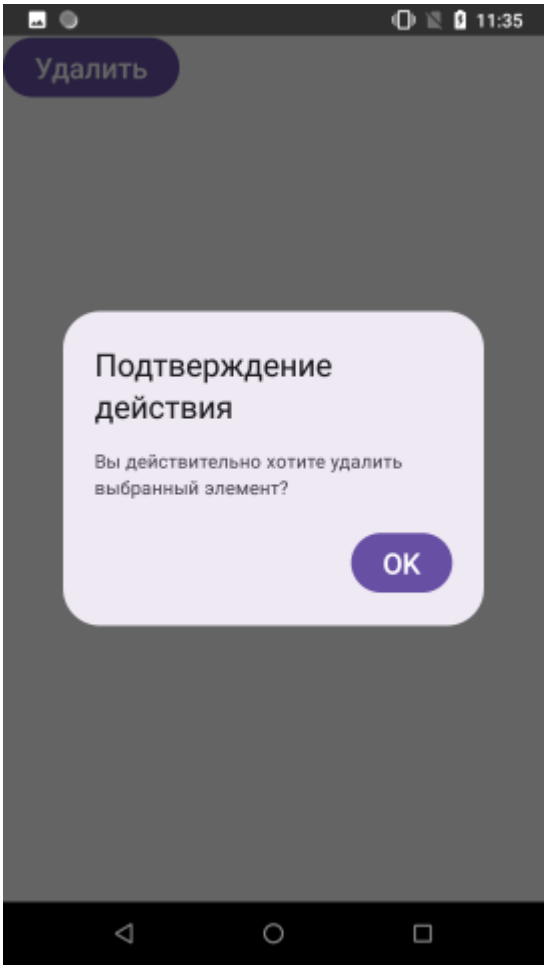
```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.*
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val openDialog = remember { mutableStateOf(false) }
            Button(
                onClick = { openDialog.value = true }
            ) {
                Text("Удалить", fontSize = 22.sp)
            }
            if (openDialog.value) {
                AlertDialog(
                    onDismissRequest = {
                        openDialog.value = false
                    },
                    title = { Text(text = "Подтверждение действия") },
                    text = { Text("Вы действительно хотите удалить выбранный
```

```
    элемент?") },
                    buttons = {
                        Button(
                            onClick = { openDialog.value = false }
                        ) {
                            Text("OK", fontSize = 22.sp)
                        }
                    }
                )
            }
        }
    }
}
```

Для управления отображением окна здесь определена переменная openDialog. Если она равна true, то диалоговое окно отображается. По умолчанию она равна false, поэтому при загрузке приложения мы не увидим окна на экране.

Нажав на кнопку с надписью "Удалить", мы изменим значение переменной openDialog на true и тем самым отобразим окно.

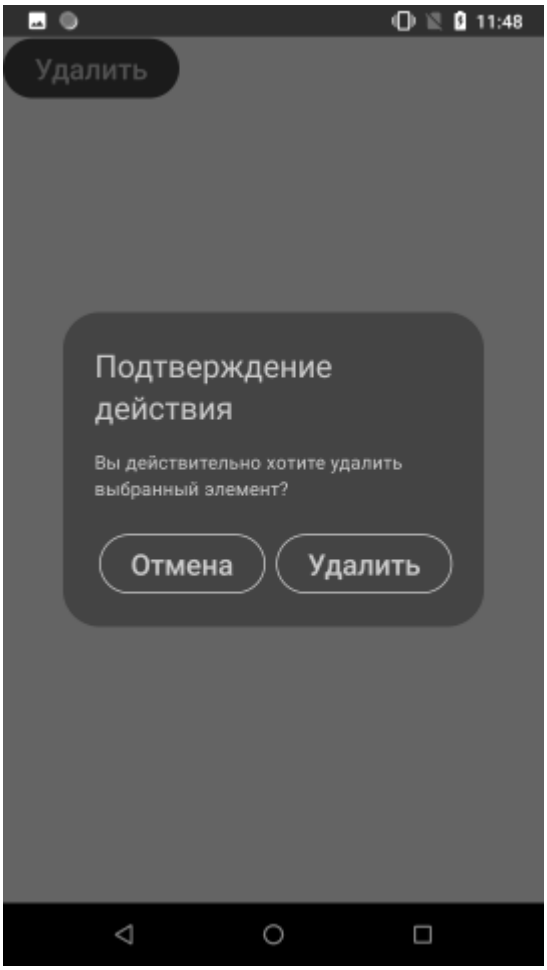
Нажав на кнопку "OK" внутри диалогового окна (срабатывает функция onClick кнопки) или на область вне окна (срабатывает функция из параметра onDismissRequest) значение переменной openDialog будет переключено обратно на false, и диалоговое окно исчезнет.



Однако, из скриншота можно увидеть, что кнопка диалогового окна по умолчанию помещается впритык в левом нижнем углу, что, возможно, не совсем эстетично выглядит. Кроме того, окно может содержать произвольное количество кнопок и необязательно только кнопок. Например, изменим панель кнопок следующим образом:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.padding
import androidx.compose.material.*
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val openDialog = remember { mutableStateOf(false) }
            Button(
                onClick = { openDialog.value = true }
            ) {
                Text("Удалить", fontSize = 22.sp)
            }
            if (openDialog.value) {
                AlertDialog(
                    onDismissRequest = {
                        openDialog.value = false
                    },
                    title = { Text(text = "Подтверждение действия") },
                    text = { Text("Вы действительно хотите удалить выбранный
элемент?") },
                    buttons = {
                        Row(
                            modifier = Modifier.padding(all = 8.dp),
                            horizontalArrangement = Arrangement.Center
                        ) {
                            Button(
                                modifier = Modifier.weight(1f),
                                onClick = { openDialog.value = false }
                            ) {
                                Text("Удалить")
                            }
                            Button(
                                modifier = Modifier.weight(1f),
                                onClick = { openDialog.value = false }
```

```
                            ) {
                                Text("Отмена")
                            }
                        }
                    }
                )
            }
        }
    }
}
```



Меню DropdownMenu

Компонент DropdownMenu позволяет компактным образом разместить ряд вариантов для выбора пользователем - некий аналог контекстного меню. DropdownMenu немного похож на всплывающее окно - он может быть отображаться, а может быть скрыт. DropdownMenu сам по себе не занимает места в разметке и отображается поверх остального контента. Обычно DropdownMenu размещается в контейнере Box, хотя это необязательное требование.

Функция компонента принимает следующие параметры:

```
@Composable
fun DropdownMenu(
```

```
    expanded: Boolean,
    onDismissRequest: () -> Unit,
    modifier: Modifier = Modifier,
    offset: DpOffset = DpOffset(0.dp, 0.dp),
    properties: PopupProperties = PopupProperties(focusable = true),
    content: ColumnScope.() -> Unit
): @Composable Unit
```

Параметры функции компонента:

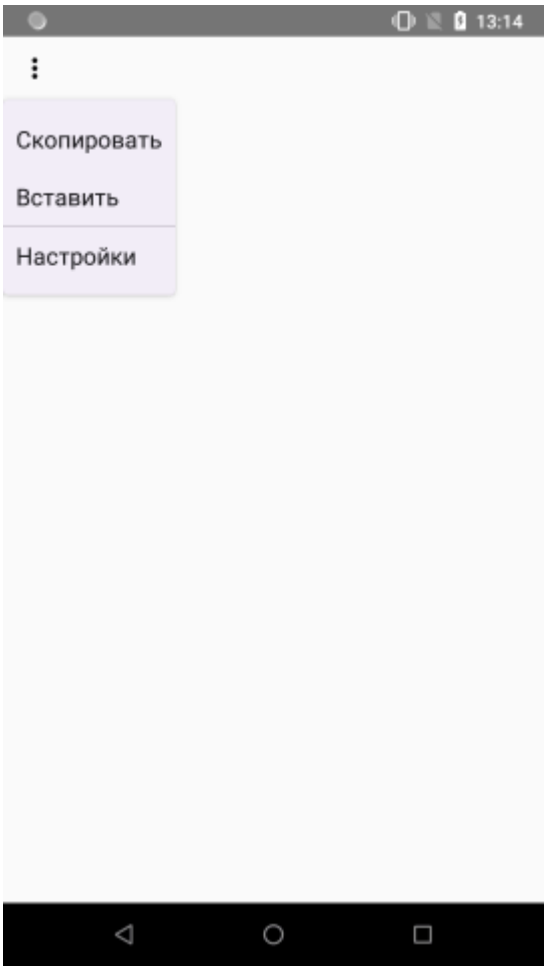
- onDismissRequest: представляет функцию-обработчик типа () -> Unit, которая вызывается, когда пользователь нажимает на область вне меню для его закрытия
- expanded: значение типа Boolean, которое устанавливает, будет ли меню отображаться (значение true) или будет скрыто (значение false)
- modifier: представляет объект Modifier, который определяет модификаторы кнопки
- offset: объект типа DpOffset, который определяет смещения позиции меню относительно положения по умолчанию. По умолчанию равно DpOffset(0.dp, 0.dp)
- properties: объект типа PopupProperties, который задает дополнительные свойства меню. По умолчанию равно PopupProperties(focusable = true)
- content: содержимое меню в виде столбца компонентов. Обычно представляет набор компонентов типа DropdownMenuItem.

Определим простейшее меню DropdownMenu:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.*
import androidx.compose.material.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.MoreVert
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
```

```
            var expanded by remember { mutableStateOf(false) }

            Box {
                IconButton(onClick = { expanded = true }) {
                    Icon(Icons.Default.MoreVert, contentDescription = "Показать
меню")
                }
                DropdownMenu(
                    expanded = expanded,
                    onDismissRequest = { expanded = false }
                ) {
                    Text("Скопировать", fontSize=18.sp, modifier =
Modifier.padding(10.dp).clickable(onClick={}))
                    Text("Вставить", fontSize=18.sp, modifier =
Modifier.padding(10.dp).clickable(onClick={}))
                    Divider()
                    Text("Настройки", fontSize=18.sp, modifier =
Modifier.padding(10.dp).clickable(onClick={}))
                }
            }
        }
    }
}
```

Здесь для управления отображением меню определена переменная expanded, и ее значение привязано к параметру expanded компонента DropdownMenu. При нажатии на кнопку IconButton эта переменная получает значение true, и меню отображается на экране.



Само меню состоит из трех компонентов Text. Между предпоследним и последним компонентами располагается элемент Divider, который просто для красоты разделяет пункты меню горизонтальной линией.

При нажатии пользователем на область вне меню, срабатывает функция параметра onDismissRequest компонента DropdownMenu. В этой функции переменная expanded получает значение false, и меню скрывается.

DropdownMenuItem

Обычно элементы меню DropdownMenu предоставляют компонент DropdownMenuItem:

```
@Composable
fun DropdownMenuItem(
    onClick: () -> Unit,
    modifier: Modifier = Modifier,
    enabled: Boolean = true,
    contentPadding: PaddingValues = MenuDefaults.DropdownMenuItemContentPadding,
    interactionSource: MutableInteractionSource = remember {
MutableInteractionSource() },
    content: RowScope.() -&t; Unit
): @Composable Unit
```

Параметры функции компонента:

- onClick: представляет функцию-обработчик нажатия меню
- modifier: предствляет объект Modifier, который определяет модификаторы компонента
- enabled: значение типа Boolean устанавливает, доступен ли компонент для нажатия (значение true) или нет (значение false)
- interactionSource: представляет объект типа MutableInteractionSource, который устанавливает поток взаимодействий для кнопки. Значение по умолчанию - remember { MutableInteractionSource() }
- contentPadding: объект типа PaddingValues, который устанавливает отступы между границами компонента и его содержимым. По умолчанию равно MenuDefaults.DropdownMenuItemContentPadding
- content: устанавливает содержимое компонента в виде строки

Используем в качестве пунктов меню DropdownMenuItem:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.*
import androidx.compose.material.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.MoreVert
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            var expanded by remember { mutableStateOf(false) }

            Box {
                IconButton(onClick = { expanded = true }) {
                    Icon(Icons.Default.MoreVert, contentDescription = "Показать
меню")
                }
                DropdownMenu(
                    expanded = expanded,
                    onDismissRequest = { expanded = false }
                ) {
                    DropdownMenuItem(onClick = { }) {
                        Text("Скопировать")
                    }
                    DropdownMenuItem(onClick = { }) {
```

```
                        Text("Вставить")
                    }
                    Divider()
                    DropdownMenuItem(onClick = { }) {
                        Text("Настройки")
                    }
                }
            }
        }
    }
}
```

Обработка выбора пунктов меню

Если пункт меню представлен компонентом DropdownMenuItem, то мы можем определить обработку выбора пункта с помощью параметра onClick:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.*
import androidx.compose.material.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.MoreVert
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.unit.DpOffset
import androidx.compose.ui.unit.dp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            var expanded by remember { mutableStateOf(false) }
            var selectedOption by remember { mutableStateOf("") }
            Column{
                Text("Выбран пункт: $selectedOption")
                Box {
                    IconButton(onClick = { expanded = true }) {
                        Icon(Icons.Default.MoreVert, contentDescription =
"Показать меню")
                    }
                    DropdownMenu(
                        expanded = expanded,
                        onDismissRequest = { expanded = false }
                    ) {
```

```
                        DropdownMenuItem(onClick = { selectedOption="Copy"}) {
                            Text("Скопировать")
                        }
                        DropdownMenuItem(onClick = { selectedOption="Paste"}) {
                            Text("Вставить")
                        }
                        Divider()
                        DropdownMenuItem(onClick = { selectedOption="Settings"}) {
                            Text("Настройки")
                        }
                    }
                }
            }

        }
    }
}
```

Здесь по нажатию на любой пункт меню в обработчике нажатия передаем переменной selectedOption соответствующее значение.

Определение позиции

Если нас не устраивает начальная позиция DropdownMenu, то мы можем ее сместить с помощью параметра offset. Он представляет объект DpOffset, который имеет два свойства: x (смещение по горизонтали) и y (смещение по вертикали). Оба свойства принимают значения в единицах dp:

```
var expanded by remember { mutableStateOf(false) }
Box {
    IconButton(onClick = { expanded = true }) {
        Icon(Icons.Default.MoreVert, contentDescription = "Показать меню")
    }
    DropdownMenu(
        expanded = expanded,
        onDismissRequest = { expanded = false },
        offset = DpOffset(x = 20.dp, y = 10.dp)
    ) {
        DropdownMenuItem(onClick = {  }) { Text("Скопировать") }
        DropdownMenuItem(onClick = { }) { Text("Вставить") }
        Divider()
        DropdownMenuItem(onClick = {  }) { Text("Настройки") }
    }
}
```

Индикаторы прогресса

CircularProgressIndicator

Компонент CircularProgressIndicator представляет круговой индикатор процесса. Он имеет следующее определение:

```
@Composable
fun CircularProgressIndicator(
    progress: Float,
    modifier: Modifier = Modifier,
    color: Color = MaterialTheme.colors.primary,
    strokeWidth: Dp = ProgressIndicatorDefaults.StrokeWidth
): @Composable Unit
```

Параметры функции компонента:

- progress: значение индикатора в виде объекта Float. Это значение находится в диапазаоне от 0.0 (процесс еще не начался) до 1.0 (завершение процесса).
- modifier: представляет объект Modifier, который определяет модификаторы компонента
- color: цвет индикатора
- strokeWidth: ширина индикатора

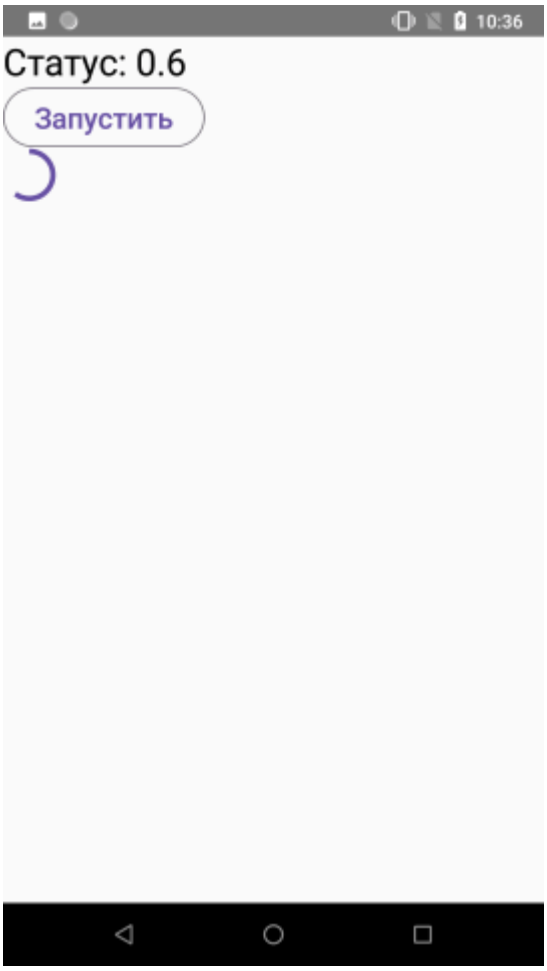
Определим индикацию процесса с помощью CircularProgressIndicator:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.*
import androidx.compose.material.*
import androidx.compose.runtime.*
import androidx.compose.ui.unit.sp
import kotlinx.coroutines.delay
import kotlinx.coroutines.launch

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            var progress by remember { mutableStateOf(0.0f) }
            val scope = rememberCoroutineScope()

            Column{
                Text("Статус: $progress", fontSize = 22.sp)
                OutlinedButton(
                    onClick = {
                        scope.launch {
                            while (progress < 1f) {
                                progress += 0.1f
                                delay(1000L)
```

```
                            }
                        }
                    }
                ) {
                    Text("Запустить", fontSize = 22.sp)
                }
                CircularProgressIndicator(progress = progress)
            }
        }
    }
}
```

В данном случае значение компонента CircularProgressIndicator привязано к переменной progress. Для имитации некоторого процесса с помощью функции rememberCoroutineScope определяем контекст корутины и по нажатию на кнопку с его с помощью создаем и заускаем корутину. В этой корутине увеличиваем значение прогресса progress на 0.1f пока не дойдем до значения 1.0. Для имитации долговременной работы применяется задержка на одну секунду с помощью вызова delay():

```
onClick = {
    scope.launch {
        while (progress < 1f) {
            progress += 0.1f
            delay(1000L)
        }
    }
}
```



Также компонент имеет еще одну версию:

```
@Composable
fun CircularProgressIndicator(
    modifier: Modifier = Modifier,
    color: Color = MaterialTheme.colors.primary,
    strokeWidth: Dp = ProgressIndicatorDefaults.StrokeWidth
): @Composable Unit
```

Эта версия предоставляет бесконечно прокручиваемый индикатор процессе.

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.*
import androidx.compose.material.*

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Box{ CircularProgressIndicator() }
```

```
        }
    }
}
```

LinearProgressIndicator

Компонент LinearProgressIndicator представляет линейный индикатор процесса. Он имеет следующее определение:

```
@Composable
fun LinearProgressIndicator(
    progress: Float,
    modifier: Modifier = Modifier,
    color: Color = MaterialTheme.colors.primary,
    backgroundColor: Color = color.copy(alpha = IndicatorBackgroundOpacity)
): @Composable Unit
```

Параметры функции компонента:

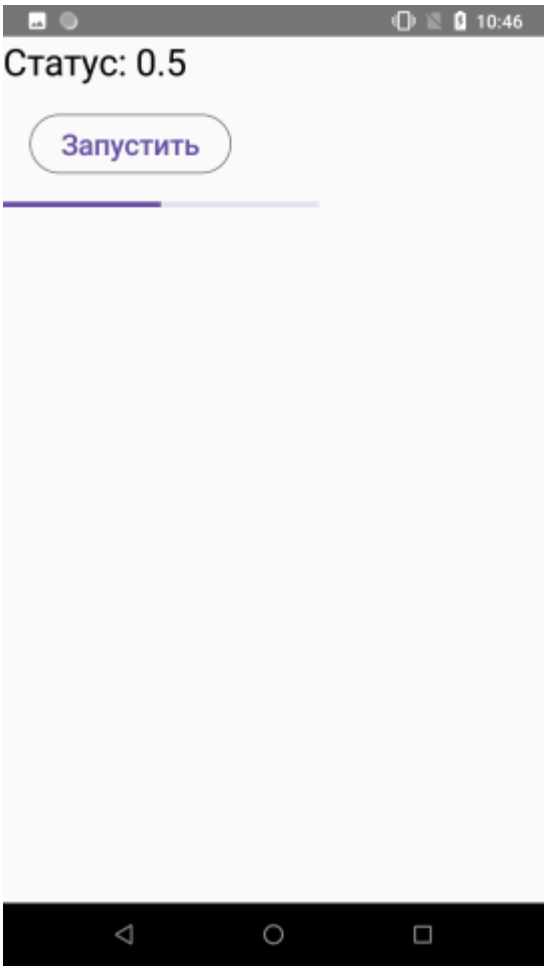
- progress: значение индикатора в виде объекта Float. Это значение находится в диапазаоне от 0.0 (процесс еще не начался) до 1.0 (завершение процесса).
- modifier: представляет объект Modifier, который определяет модификаторы компонента
- color: цвет закрашенной части индикатора (пройденная часть)
- backgroundColor: цвет незакрашенной части индикатора (еще непройденная часть)

Определим индикацию процесса с помощью LinearProgressIndicator:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.*
import androidx.compose.material.*
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.sp
import androidx.compose.ui.unit.dp
import kotlinx.coroutines.delay
import kotlinx.coroutines.launch


class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
```

```
            var progress by remember { mutableStateOf(0.0f) }
            val scope = rememberCoroutineScope()

            Column{
                Text("Статус: $progress", fontSize = 22.sp)
                OutlinedButton(
                    modifier = Modifier.padding(20.dp),
                    onClick = {
                        scope.launch {
                            while (progress < 1f) {
                                progress += 0.1f
                                delay(1000L)
                            }
                        }
                    }
                ) {
                    Text("Запустить", fontSize = 22.sp)
                }

                LinearProgressIndicator(progress = progress)
            }
        }
    }
}
```

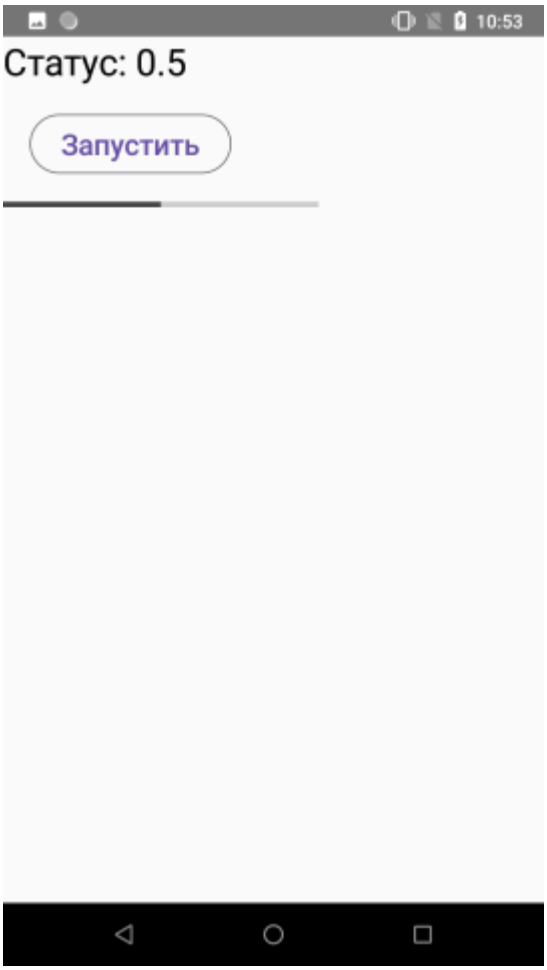
Здесь параметр progress компонента LinearProgressIndicator привязан к значению переменной progress. Для имитации некоторого процесса с помощью функции rememberCoroutineScope определяем контекст корутины и по нажатию на кнопку с его с помощью создаем и заускаем корутину. В этой корутине увеличиваем значение прогресса progress на 0.1f пока не дойдем до значения 1.0. Для имитации долговременной работы применяется задержка на одну секунду с помощью вызова delay().



Параметры color и backgroundColor позволяют настроить цветовую схему индикатора:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.*
import androidx.compose.material.*
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.sp
import androidx.compose.ui.unit.dp
import kotlinx.coroutines.delay
import kotlinx.coroutines.launch


class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            var progress by remember { mutableStateOf(0.0f) }
            val scope = rememberCoroutineScope()

            Column{
```

```
                OutlinedButton(
                    modifier = Modifier.padding(20.dp),
                    onClick = {
                        scope.launch {
                            while (progress < 1f) {
                                progress += 0.1f
                                delay(1000L)
                            }
                        }
                    }
                ) {
                    Text("Запустить", fontSize = 22.sp)
                }

                LinearProgressIndicator(
                    progress = progress,
                    color = Color(0xFFD32F2F),
                    backgroundColor = Color(0xFFEF9A9A)
                )
            }
        }
    }
}
```



Также компонент имеет еще одну версию:

```
@Composable
fun LinearProgressIndicator(
    modifier: Modifier = Modifier,
    color: Color = MaterialTheme.colors.primary,
    backgroundColor: Color = color.copy(alpha = IndicatorBackgroundOpacity)
): @Composable Unit
```

Эта версия предоставляет бесконечно прокручиваемый индикатор процессе. Пример ее применения:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.*
import androidx.compose.material.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Column{
                LinearProgressIndicator(modifier = Modifier.padding(20.dp))
            }
        }
    }
}
```