

Comparativo >>>>

XLM vs Compose

NA CRIAÇÃO E MANUTENÇÃO DE
UIS NO ANDROID



XML

É a forma como, por mais de uma década, os layouts de tela foram construídos no Android.

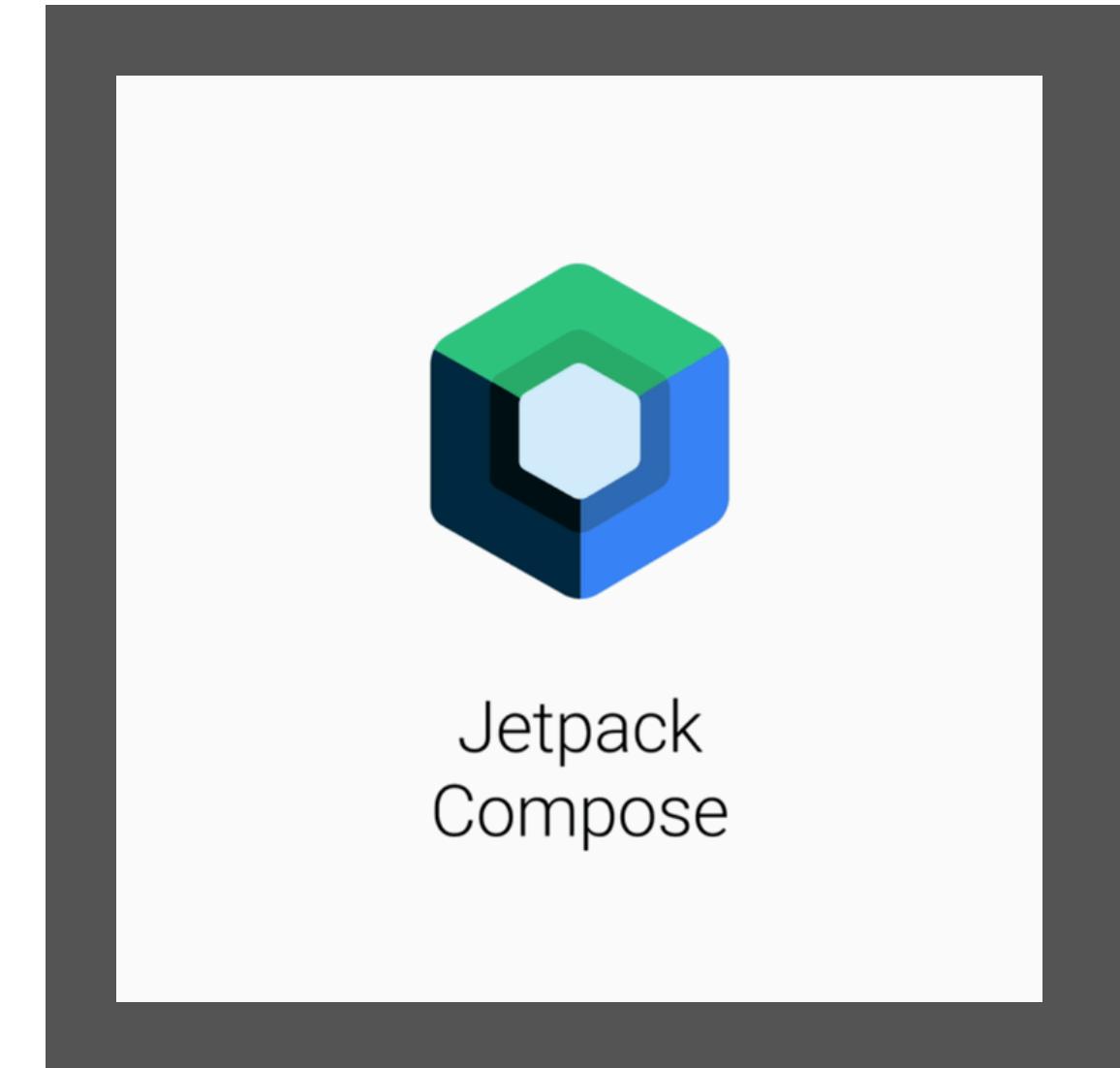
A View XML e o seu comportamento vivem em arquivos separados



Compose

É a nova forma, recomendada pelo Google, de criar interfaces no Android.

A UI é construída diretamente no código Kotlin, o que permite criar componentes reutilizáveis e lógicas complexas.



A UI reage automaticamente a mudanças de estado.

Imperativo vs Declarativo



XML

Exemplo de construção de botões na View Xml

The screenshot shows the Android Studio XML layout editor. The top bar displays the file name "activity_home.xml". The left pane contains the XML code for the layout. The right pane shows the visual representation of the screen, which includes a logo, a title, and three green buttons labeled "Calcule", "Seus resultados", and "Referência". Below the screen preview is a component tree showing an ImageView.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <!-- Logo -->
    <com.example.imc.RoundedImageview
        android:id="@+id/logo_imc"
        android:layout_width="150dp"
        android:layout_height="150dp"
        android:src="@mipmap/ic_launcher"
        android:layout_marginBottom="16dp"/>
    <!-- Title -->
    <com.example.imc.TextView
        android:id="@+id/titulo_imc"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Calculadora de Índice de Massa Corpórea"
        android:fontStyle="italic"
        android:fontFamily="sans-serif"
        android:layout_marginTop="16dp"/>
    <!-- Botão Calcule -->
    <com.example.imc.Button
        android:id="@+id/button_calcule_home"
        android:layout_width="203dp"
        android:layout_height="54dp"
        android:text="Calcule"
        android:background="@color/colorPrimary"
        android:fontStyle="bold"
        android:fontFamily="sans-serif"
        android:layout_marginTop="16dp"/>
    <!-- Botão Seus resultados -->
    <com.example.imc.Button
        android:id="@+id/btn_resultados"
        android:layout_width="203dp"
        android:layout_height="54dp"
        android:text="Seus resultados"
        android:background="@color/colorPrimary"
        android:fontStyle="bold"
        android:fontFamily="sans-serif"
        android:layout_marginTop="16dp"/>
    <!-- Botão Referência -->
    <com.example.imc.Button
        android:id="@+id/btn_referencias"
        android:layout_width="203dp"
        android:layout_height="54dp"
        android:text="Referência"
        android:background="@color/colorPrimary"
        android:fontStyle="bold"
        android:fontFamily="sans-serif"
        android:layout_marginTop="16dp"/>
    <!-- Guideline -->
    <androidx.constraintlayout.widget.Guideline
        android:id="@+id/guideline4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:layout_constraintTop_toTopOf="parent"
        android:layout_constraintBottom_toBottomOf="parent"/>

```

Construção de um botão no XML

Obs: Para efeitos adicionais, como um **RoundedCorner** personalizado é necessário criar um arquivo de recurso xml, e atribui-lo ao botão

```
<Button  
    android:id="@+id/button_calcule_home"  
    android:layout_width="203dp"  
    android:layout_height="54dp"  
    android:text="Calcule"  
    app:layout_constraintBottom_toTopOf="@+id/btn_resultados"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintHorizontal_bias="0.5"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintTop_toBottomOf="@+id/guideline4" />
```

Conectando a view xml e adicionando funções aos botões

```
class HomeActivity : AppCompatActivity() {  
  
    lateinit var btnCalcule: Button  
    lateinit var btnVerResultados: Button  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_home)  
  
        btnCalcule = findViewById(R.id.button_calcule_home)  
        btnVerResultados = findViewById(R.id.btn_resultados)  
  
        btnCalcule.setOnClickListener {  
            val abrirCalculadora = Intent(this, MainActivity::class.java)  
            startActivity(abrirCalculadora)  
        }  
  
        btnVerResultados.setOnClickListener {  
            val resultadosAnteriores = Intent(this, ResultadosAnterioresActivity::class.java)  
            startActivity(resultadosAnteriores)  
        }  
    }  
}
```

Definindo a View XML

Mapeando os botões
pelo ID do
componente XML

```
<Button  
    android:id="@+id/button_calcule_home"  
  
<Button  
    android:id="@+id/btn_resultados"
```

Definindo a ação de
click do btoão

Compose

Exemplo de construção de botões

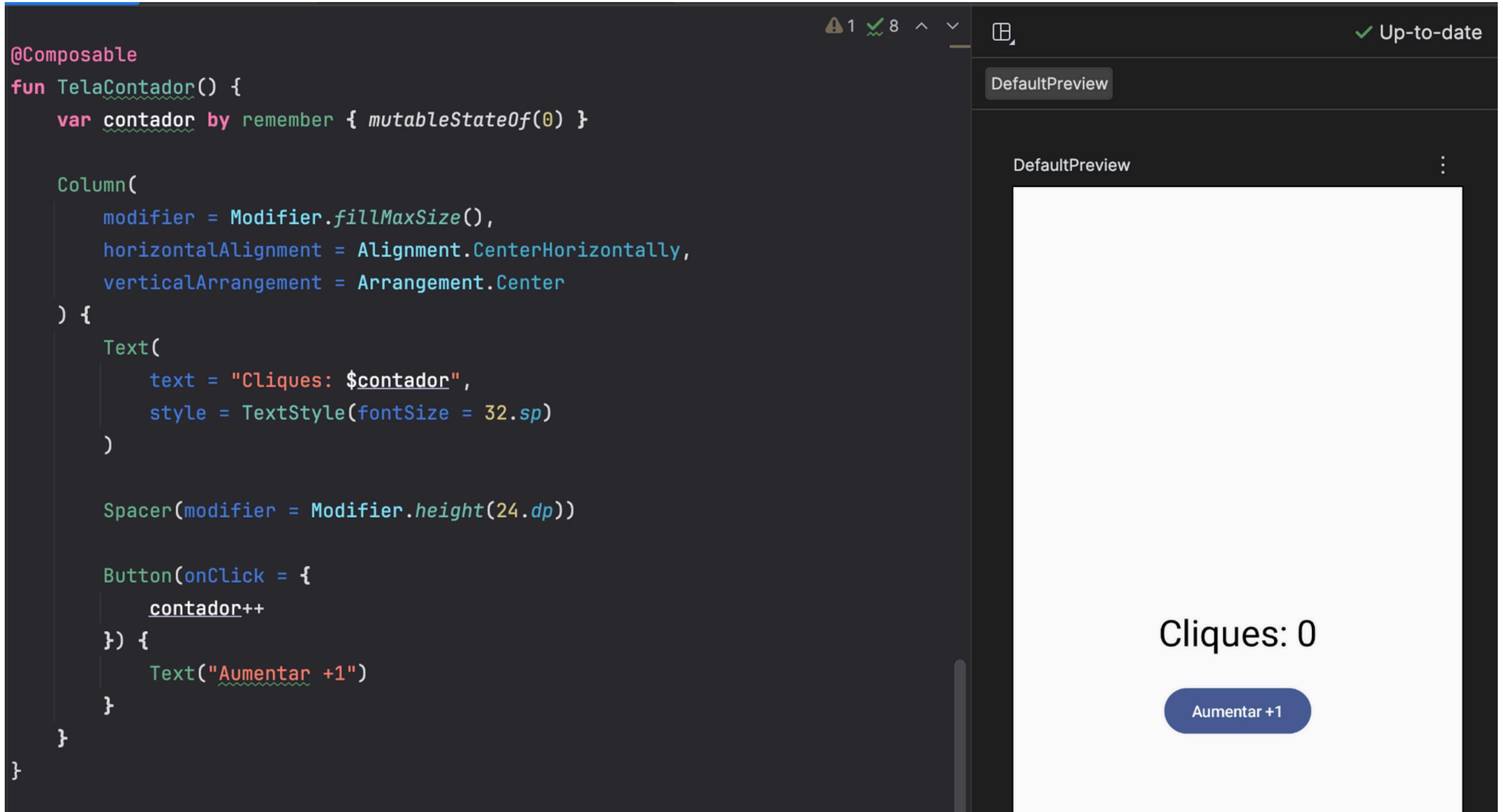
```
@Composable  
@ComposableInferredTarget  
public fun Button(  
    onClick: () -> Unit,  
    modifier: Modifier = Modifier,  
    enabled: Boolean = true,  
    shape: Shape = ButtonDefaults.shape,  
    colors: ButtonColors = ButtonDefaults.buttonColors(),  
    elevation: ButtonElevation? = ButtonDefaults.buttonElevation(),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

Por padrão, o Botão tem todos esses parâmetros, sendo o mais comum o **onClick()**

Para construir um botão usando compose, basta chamar a função Button() e passar os parâmetros que deseja. Exemplo a seguir

Compose

Tudo centralizado



The screenshot shows the Android Studio preview window for Jetpack Compose. The code on the left defines a composable function `TelaContador()`. It uses `remember` to create a mutable state `contador` initialized to 0. The UI consists of a centered `Text` displaying the value of `contador`, a vertical `Spacer`, and a centered `Button` labeled "Aumentar +1" which increments the state. The preview on the right shows the resulting UI with the text "Cliques: 0" and the button centered below it.

```
@Composable
fun TelaContador() {
    var contador by remember { mutableStateOf(0) }

    Column(
        modifier = Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        Text(
            text = "Cliques: $contador",
            style = TextStyle(fontSize = 32.sp)
        )

        Spacer(modifier = Modifier.height(24.dp))

        Button(onClick = {
            contador++
        }) {
            Text("Aumentar +1")
        }
    }
}
```

Criando a tela Compose

```
@Composable  
fun TelaContador() {  
    var contador by remember { mutableStateOf(0) }  
  
    Column(  
        modifier = Modifier.fillMaxSize(),  
        horizontalAlignment = Alignment.CenterHorizontally,  
        verticalArrangement = Arrangement.Center  
    ) {  
        Text(  
            text = "Cliques: $contador",  
            style = TextStyle(fontSize = 32.sp)  
        )  
  
        Spacer(modifier = Modifier.height(24.dp))  
  
        Button(onClick = {  
            contador++  
        }) {  
            Text("Aumentar +1")  
        }  
    }  
}
```

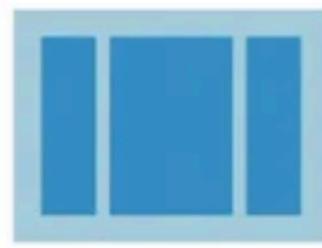
Telas compose são funções
@Composable

Chama a função do Botão, passando
como parâmetro apenas o que ele
deve fazer ao ser clicado

Valor de “contador” é atualizado
automaticamente na tela, graças ao
mutableStateOf()

Tipos de layout

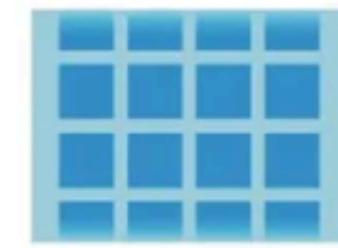
XML



LinearLayout



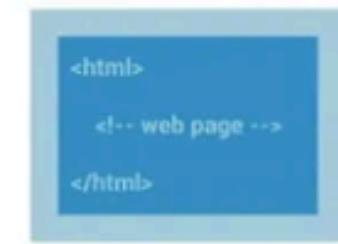
RelativeLayout



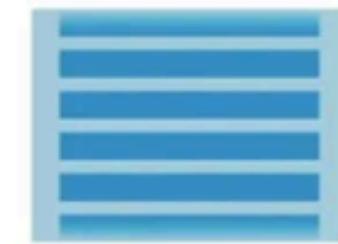
GridLayout



TableLayout

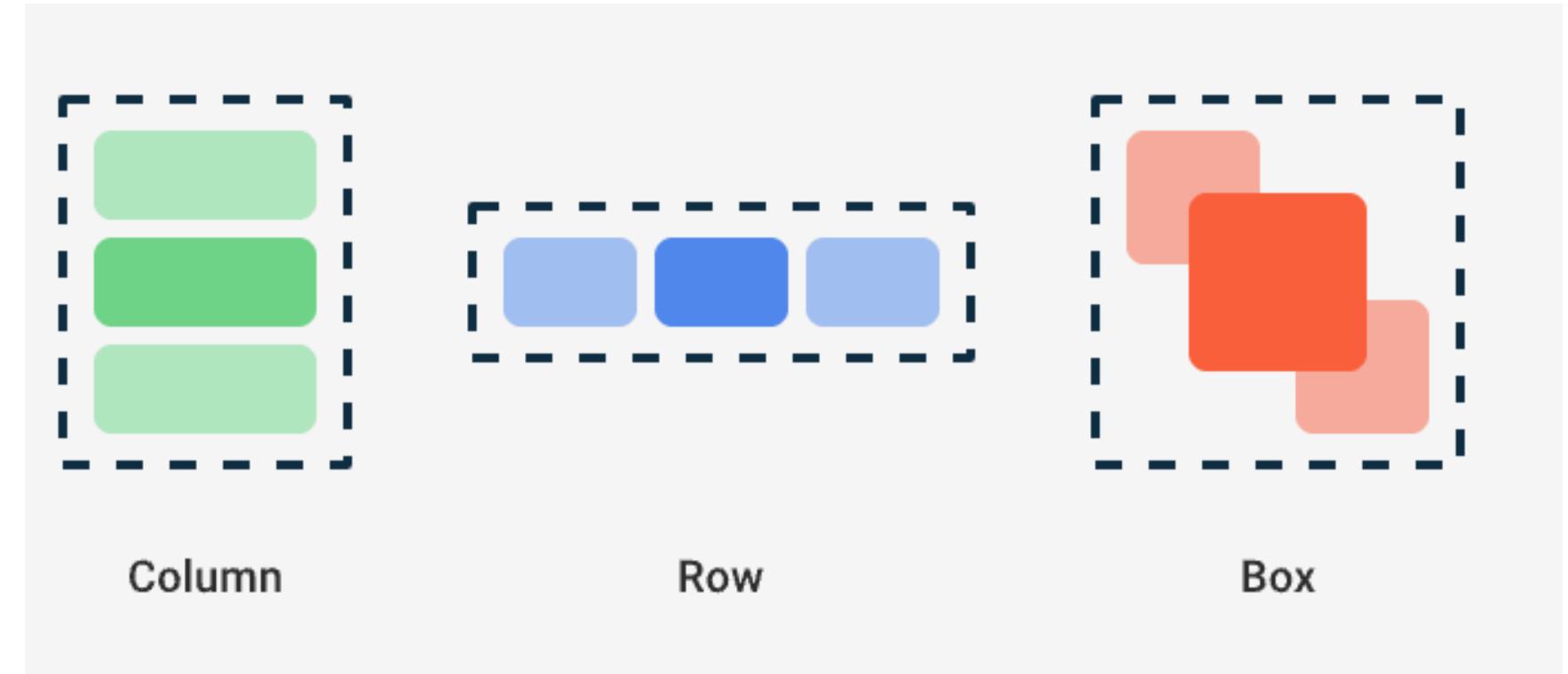


WebView



ListView/RecyclerView

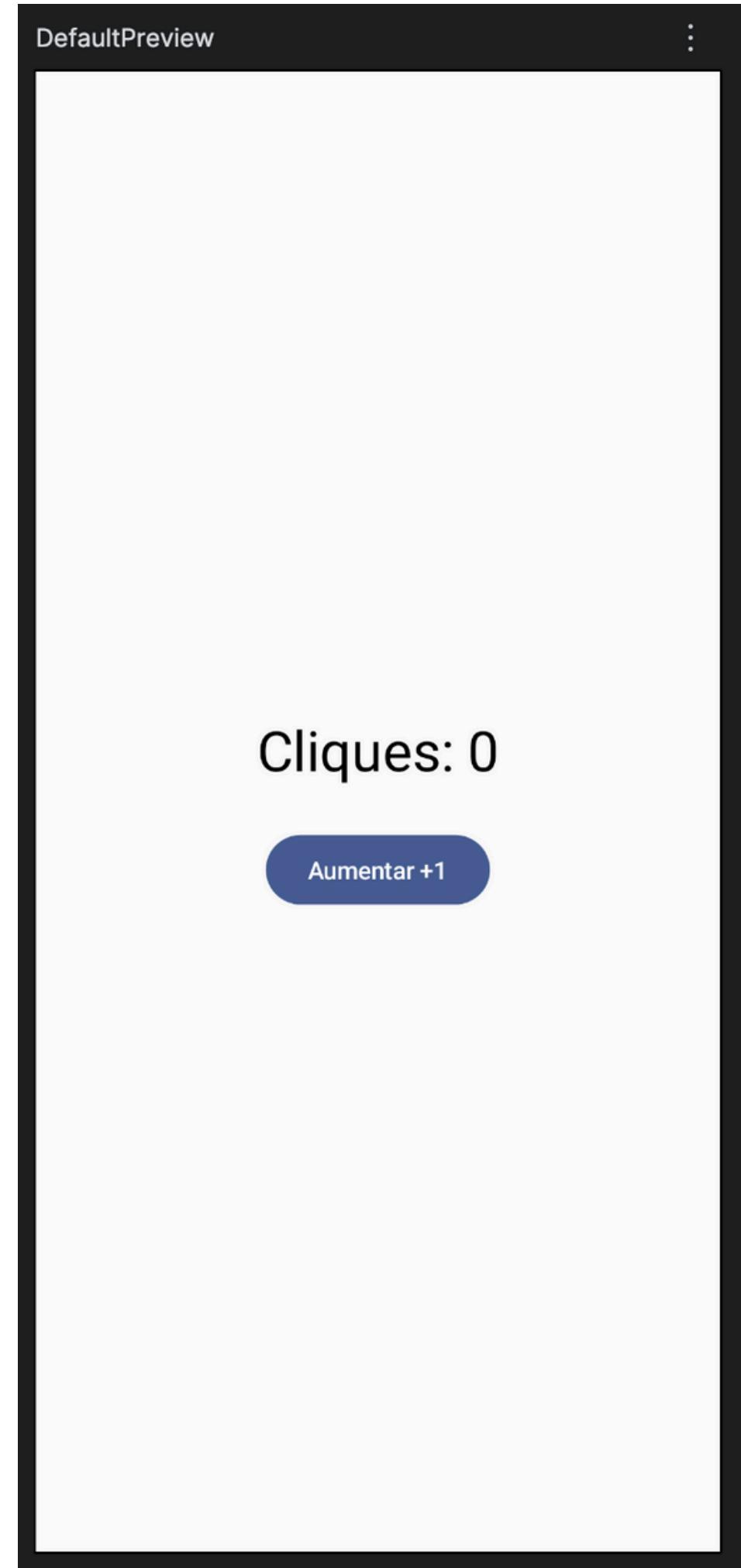
Compose



Passo a passo:

Criação de uma view simples de um Contador

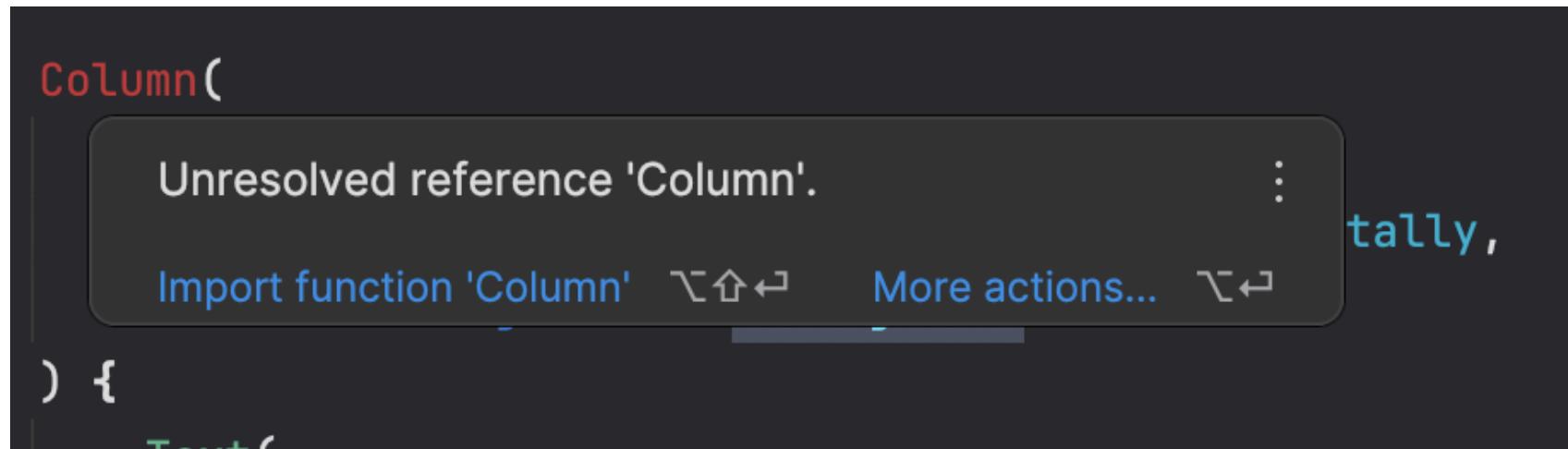
Compose



Passo a passo: COMPOSE

1. No seu pacote, crie uma nova Kotlin Class/File
2. Escolha o nome da sua tela
3. Faça as importações necessárias

É mais comum importar a medida que você vai construindo o layout, o próprio AndroidStudio dá a sugestão:



Passo a passo: COMPOSE

4. Escreva o código da sua tela:

5. Crie a função com o marcador @Preview e chame a função da sua tela

```
@Preview(showBackground = true)
@Composable
fun DefaultPreview() {
    TelaContador()
}
```

The screenshot shows the Android Studio Preview window. At the top, there are two tabs: "DefaultPreview" and "DefaultPreview". The first tab is selected, showing the code for the composable function. The second tab shows the resulting UI preview. The UI consists of a large text "Cliques: 0" in a bold, black font, centered on the screen. Below it is a blue button with the text "Aumentar +1".

```
@Composable
fun TelaContador() {
    var contador by remember { mutableStateOf(0) }

    Column(
        modifier = Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        Text(
            text = "Cliques: $contador",
            style = TextStyle(fontSize = 32.sp)
        )

        Spacer(modifier = Modifier.height(24.dp))

        Button(onClick = {
            contador++
        }) {
            Text("Aumentar +1")
        }
    }
}
```

Passo a passo: COMPOSE

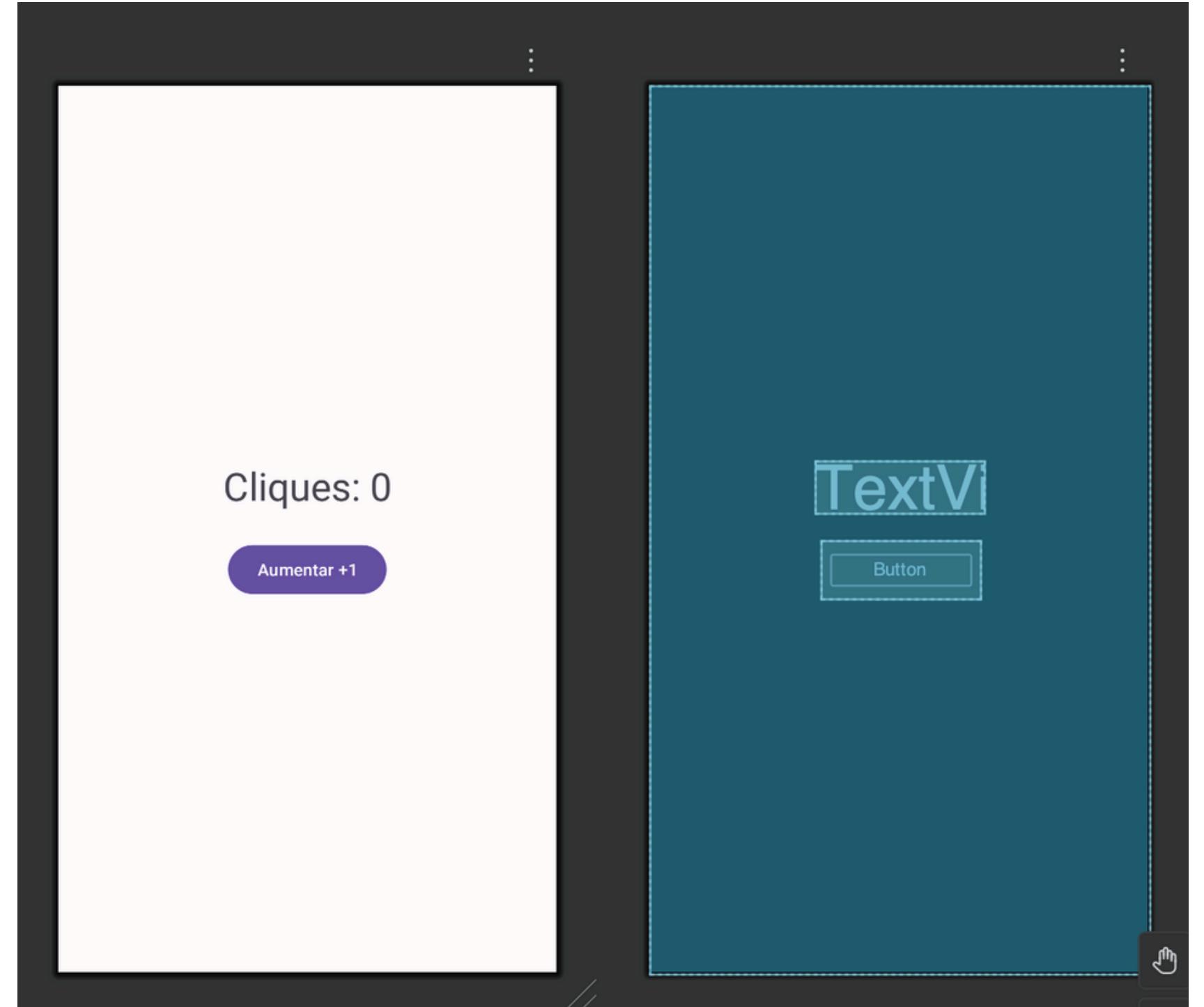
6. E está pronto!! Basta chamar a função da sua tela no seu MainActivity.kt

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

Passo a passo:

Criação de uma view simples de um Contador

XML



Passo a passo:

Criação de uma view simples de um Contador

1. Navegue até res/layout e acesse o activity main.xml
2. Lá, você poderá construir seu layout XML escrevendo o XML, ou utilizando o Layout Editor
3. Escreva o código ou Arraste os elementos para a tela

3. Escreva o código ou Arraste os elementos para a tela

The screenshot shows the Android Studio interface with the following components:

- Left Panel (Code Editor):** Displays the XML code for `activity_main.xml`. The code defines a `ConstraintLayout` containing a `TextView` and a `Button`. The `TextView` has the text "Cliques: 0". The `Button` has the text "Aumentar +1". Both elements are constrained to the parent layout.
- Top Bar:** Shows tabs for `activity_main.xml` and `MainActivity.kt`, along with icons for search, refresh, and other tools.
- Toolbar:** Includes icons for file operations, navigation, and settings.
- Palette:** A sidebar on the right containing icons for various UI components like `Text View`, `Image`, `Card View`, etc.
- Layout Preview:** Two side-by-side preview windows showing the UI. The left preview shows the actual UI with a white background, displaying "Cliques: 0" and a button labeled "Aumentar +1". The right preview shows the same UI but with a dark teal background.

Passo a passo:

Criação de uma view simples de um Contador

4. Acesse Gradle Scripts e vá para o arquivo build.gradle.kts (Module : app)

4.1 Adicione a funcionalidade de binding

4.2 Caso não exista um buildFeatures, basta cria-lo.

4.3 Na barra azul superior, sincronize o Gradle.

```
1  plugins {
2      alias(libs.plugins.android.application)
3      alias(libs.plugins.kotlin.android)
4  }
5
6  android {
7      namespace = "com.example.teste"
8      compileSdk = 35
9
10     buildFeatures {
11         viewBinding = true
12     }
13 }
```

Passo a passo:

Criação de uma view simples de um Contador

Uso do **Biding**:

O View Binding, é a abordagem moderna recomendada pelo Google para interagir com views XML de forma segura.

Ele cria uma ponte segura e direta entre o código Kotlin e o arquivo de layout XML.

Para cada arquivo de layout XML (ex: activity_main.xml), o sistema de build do Android gera automaticamente uma classe de vinculação (ex: ActivityMainBinding).

Essa classe gerada contém referências diretas e já com o tipo correto para todas as Views do seu layout que possuem um ID.

Passo a passo:

Criação de uma view simples de um Contador

5. Agora, conecte o layout XML à lógica em Kotlin com o Binding

5.1 Acesse o MainActivity.kt

5.2 Escreva o código e faça o link aos elementos XML

(Texto do contador e botão)

```
class MainActivity : AppCompatActivity() {  
  
    // 1. Declara a variável de "binding" que dará acesso seguro às Views  
    private lateinit var binding: ActivityMainBinding  
  
    // 2. A variável que guardará o estado do contador  
    private var contador = 0  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
  
        // 3. "Infla" o layout XML e prepara o objeto binding para uso  
        binding = ActivityMainBinding.inflate(layoutInflater)  
        setContentView(binding.root)  
  
        // 4. Define o que acontece quando o botão é clicado  
        binding.buttonAumentar.setOnClickListener {  
            // 5. Incrementa a variável do contador  
            contador++  
  
            // 6. Atualiza manualmente o texto na tela  
            // Esta é a principal diferença: a atualização não é automática!  
            binding.textViewContador.text = "Cliques: $contador"  
        }  
    }  
}
```

Conclusão

O que você vai usar para construir os Layouts depende muito do que você se sente mais confortável fazendo. Ainda existem milhões de apps que utilizam **XML**, e até a união entre **XML** e **Compose**.

Caso você ainda não conheça nenhuma das duas formas, eu recomendaria começar pelo **Compose**. Atualmente é a forma recomendada pelo Android, por ser declarativa, torna o código mais comprehensível, e pessoalmente, acho menos trabalhoso.

Comparativo final

Linguagem e Sintaxe



Linguagem de marcação separada.
Requer troca de contexto constante entre arquivos XML e código Kotlin/Java.

Quantidade de Código

Mais verboso. Requer Adapters para `Recyclers`, `findViewById`, View/Data Binding, múltiplos arquivos de layout para diferentes estados



A UI é construída com a mesma linguagem da lógica de negócios, permitindo mais poder e flexibilidade

Significativamente reduzido. Listas são criadas com `LazyColumn/LazyRow` de forma simples. A lógica condicional para exibir componentes é feita com um simples `if/else`

Comparativo final

Gerenciamento de Estado



Complexo. Atualização manual das Views para evitar vazamentos de memória e manter a consistência.

Reutilização de componentes

Feita através de Custom Views ou layouts com `<include>`, o que pode ser complexo e verboso.



Simplificado. Usa padrões reativos e integrados como `mutableStateOf` para gerenciar o ciclo de vida do estado de forma eficiente.

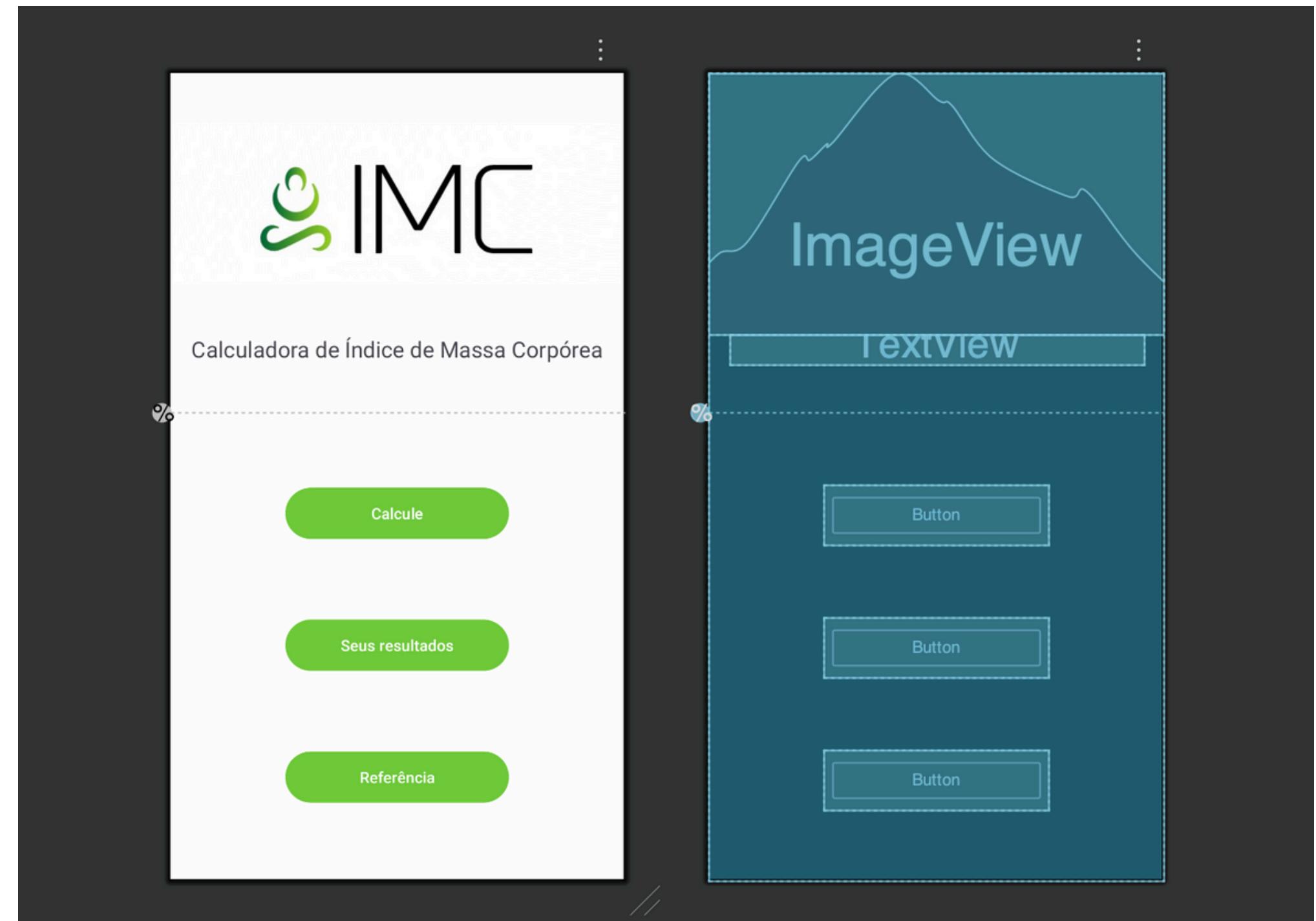
Qualquer função `@Composable` é um componente reutilizável. Basta chamar a função com parâmetros diferentes.

Comparativo final

Preview



O Layout Editor mostra a representação da UI, mas não é possível interagir em tempo real

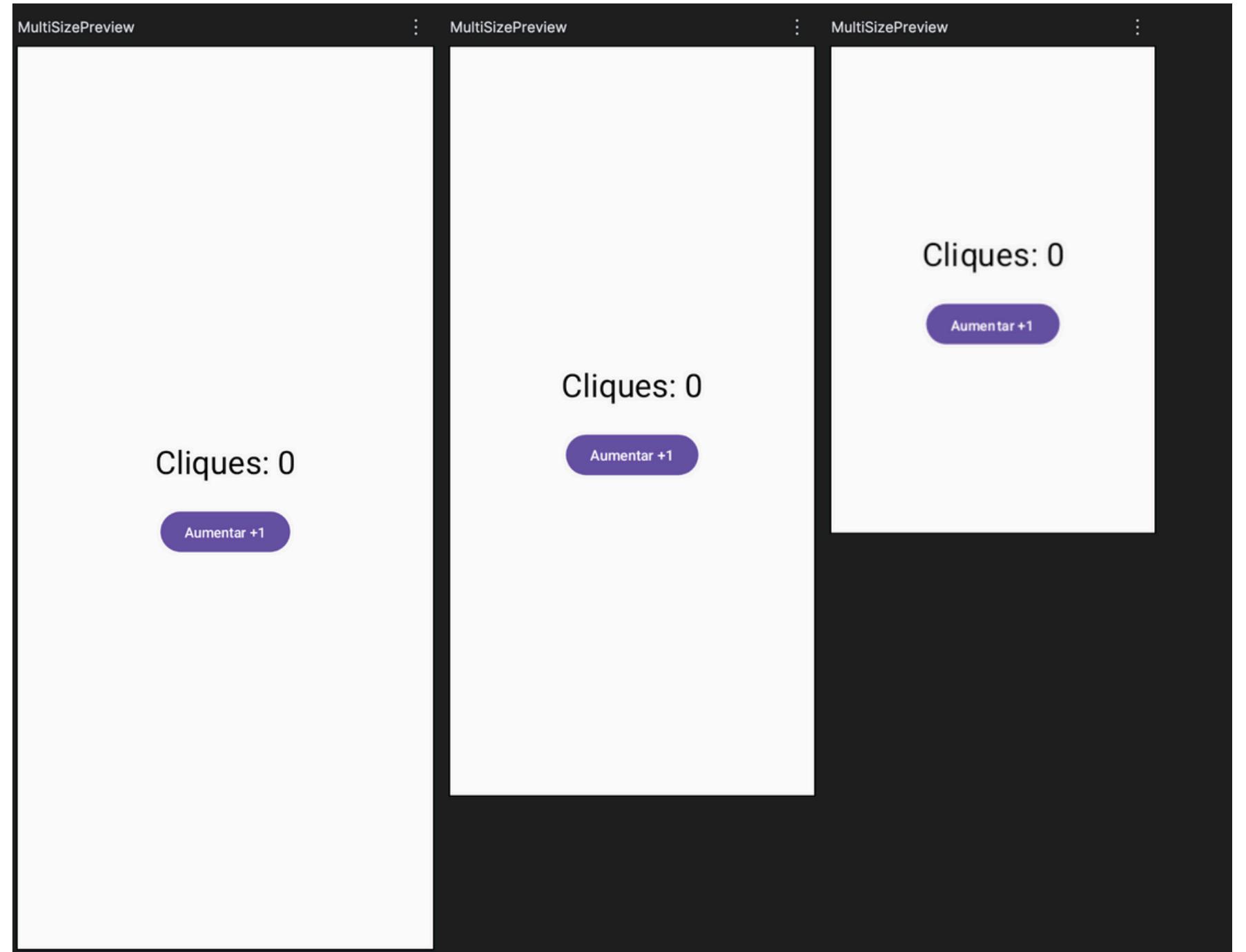


Comparativo final

Preview



Previews em tempo real (@Preview)
Permitem visualizar componentes
isoladamente, em diferentes temas
(modo noturno), com dados variados
e até mesmo de forma interativa,
acelerando o ciclo de
desenvolvimento.



Obrigado!

