



UNIDADE II

Desenvolvimento para
Dispositivos Móveis

Prof. MSc. Olavo Ito

Introdução ao Android Studio

Introdução ao Android Studio

O que é o Android Studio?

- IDE oficial para criação de aplicativos Android.
- Desenvolvido e distribuído pelo Google.

Introdução ao Android Studio

Requisitos do Sistema:

- **Sistemas Operacionais:**
 - *Linux, macOS (10.14 Mojave ou mais recente);*
 - *ChromeOS;*
 - *Windows (8, 10, 11) de 64 bits.*

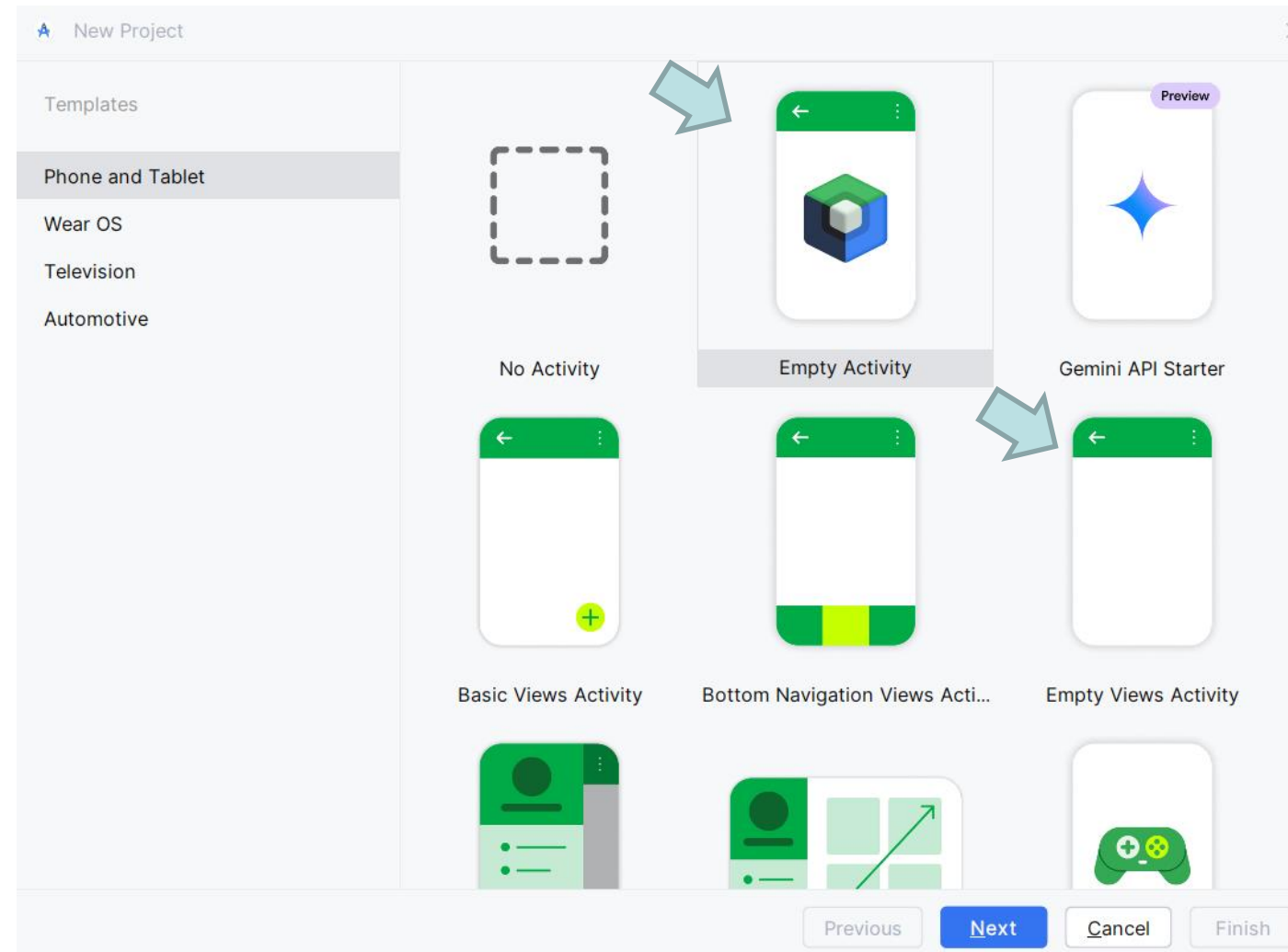
Hardware:

- CPU: Intel Core de segunda geração ou mais recente, ou CPU AMD com suporte a Hypervisor do Windows.
- RAM: Pelo menos 8 GB de RAM livre.
- Espaço em Disco: Mínimo de 8 GB disponível.
- Resolução de Tela: Mínimo de 1.280 x 800.
- **Acesso à Internet:** Necessário para download e atualizações.

Iniciando um Projeto no Android Studio

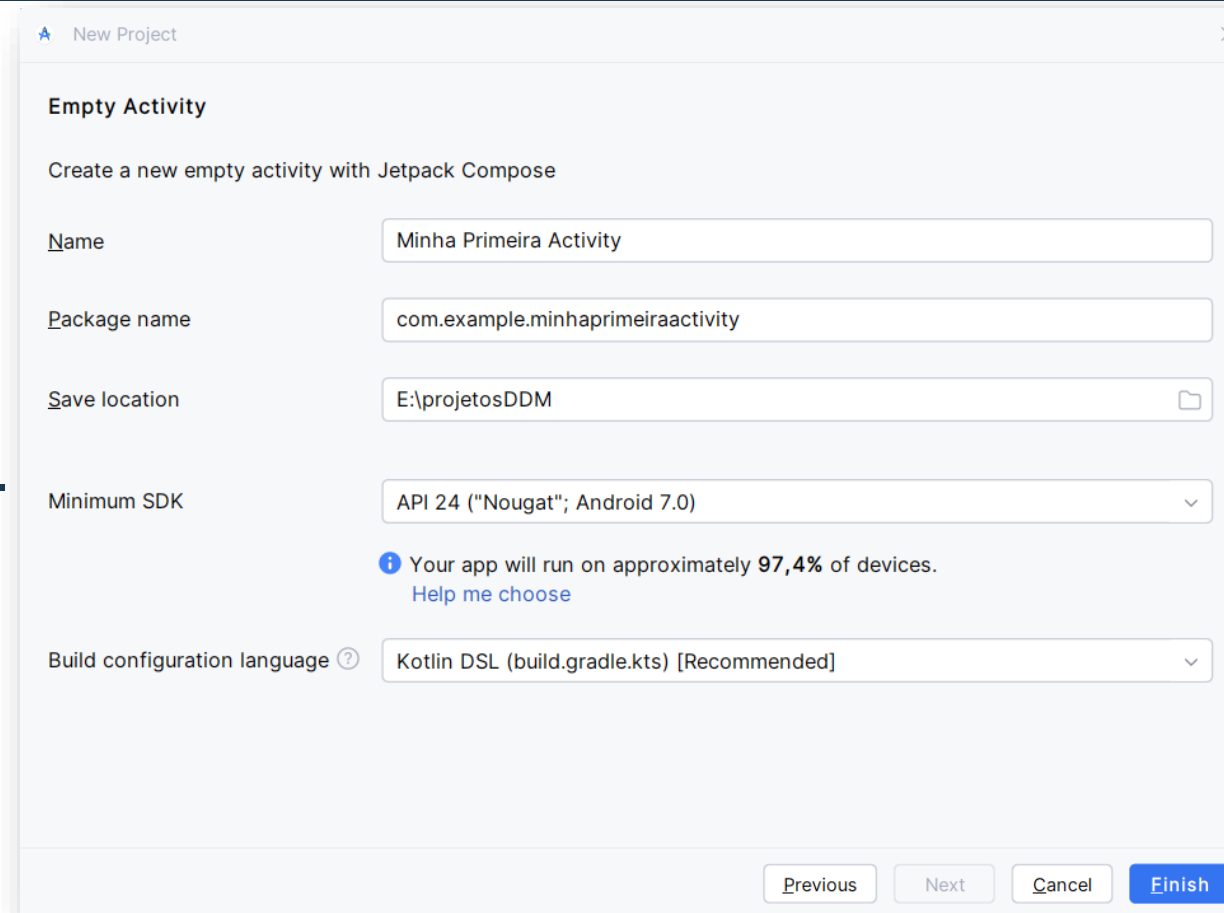
Iniciando um Projeto no Android Studio

- Tela Inicial: Welcome to Android Studio
 - Clique em “New Project”.
- Escolha do Template
 - Tela de definição do tipo de template
 - Templates indicados: “Empty Activity” e “Empty Views Activity”.
 - Escolha “Empty Activity” e clique em “Next”.



Iniciando um Projeto no Android Studio

- Configuração Inicial do Projeto
 - Nome: “Minha Primeira Activity”.
 - Package name: `com.example.minhaprimeiraactivity``.
 - Save location: Mantenha como está.
 - Minimum SDK: API 24: Android 7.0 (Nougat).
 - Build configuration language: Kotlin DSL (build.gradle.kts) [Recommended].
 - Clique em “Finish”.



The screenshot shows the 'New Project' dialog in Android Studio, specifically the 'Empty Activity' screen. The dialog is titled 'New Project' and has a close button (X) in the top right corner. Below the title, it says 'Empty Activity' and 'Create a new empty activity with Jetpack Compose'. The form contains the following fields:

- Name:** 'Minha Primeira Activity'
- Package name:** 'com.example.minhaprimeiraactivity'
- Save location:** 'E:\projetosDDM' (with a folder icon on the right)
- Minimum SDK:** 'API 24 ("Nougat"; Android 7.0)' (with a dropdown arrow on the right)
- Build configuration language:** 'Kotlin DSL (build.gradle.kts) [Recommended]' (with a dropdown arrow on the right)

Below the 'Minimum SDK' field, there is an information icon (i) and a message: 'Your app will run on approximately 97,4% of devices.' with a link 'Help me choose'.

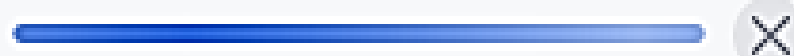
At the bottom right, there are four buttons: 'Previous', 'Next', 'Cancel', and 'Finish' (which is highlighted in blue).

Iniciando um Projeto no Android Studio

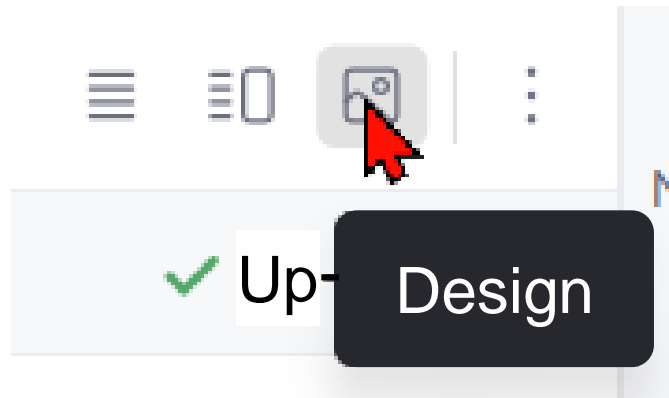
- Montagem do Projeto

- Aguarde a montagem do projeto.

Importing 'projetosDDM' Gradle Project

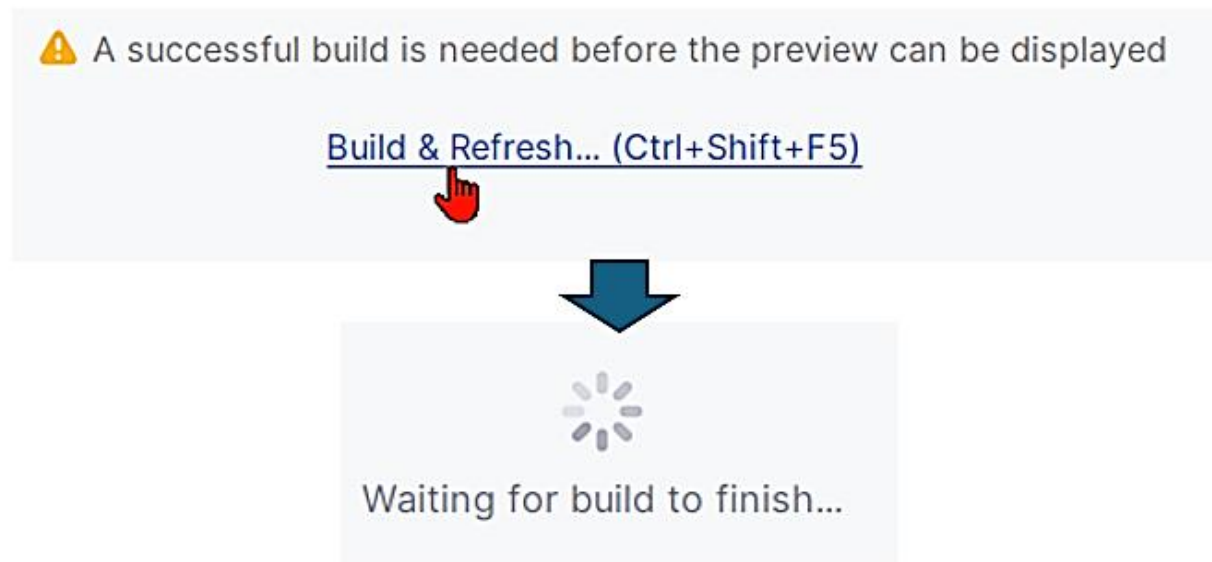


- Verifique se não há barras de carregamento ou algo rodando.
 - Configuração do Layout do Editor
 - Se a tela de design não aparecer, clique no botão de layout do editor e selecione “Design” .

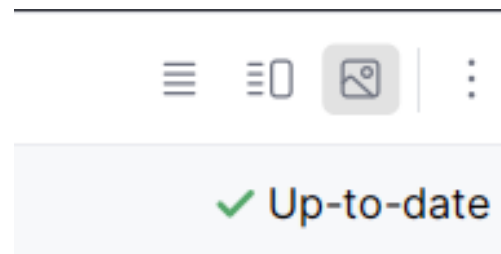


Iniciando um Projeto no Android Studio

- Ativação da pré-visualização
 - Clique em 'Build & Refresh' para realizar uma compilação completa.



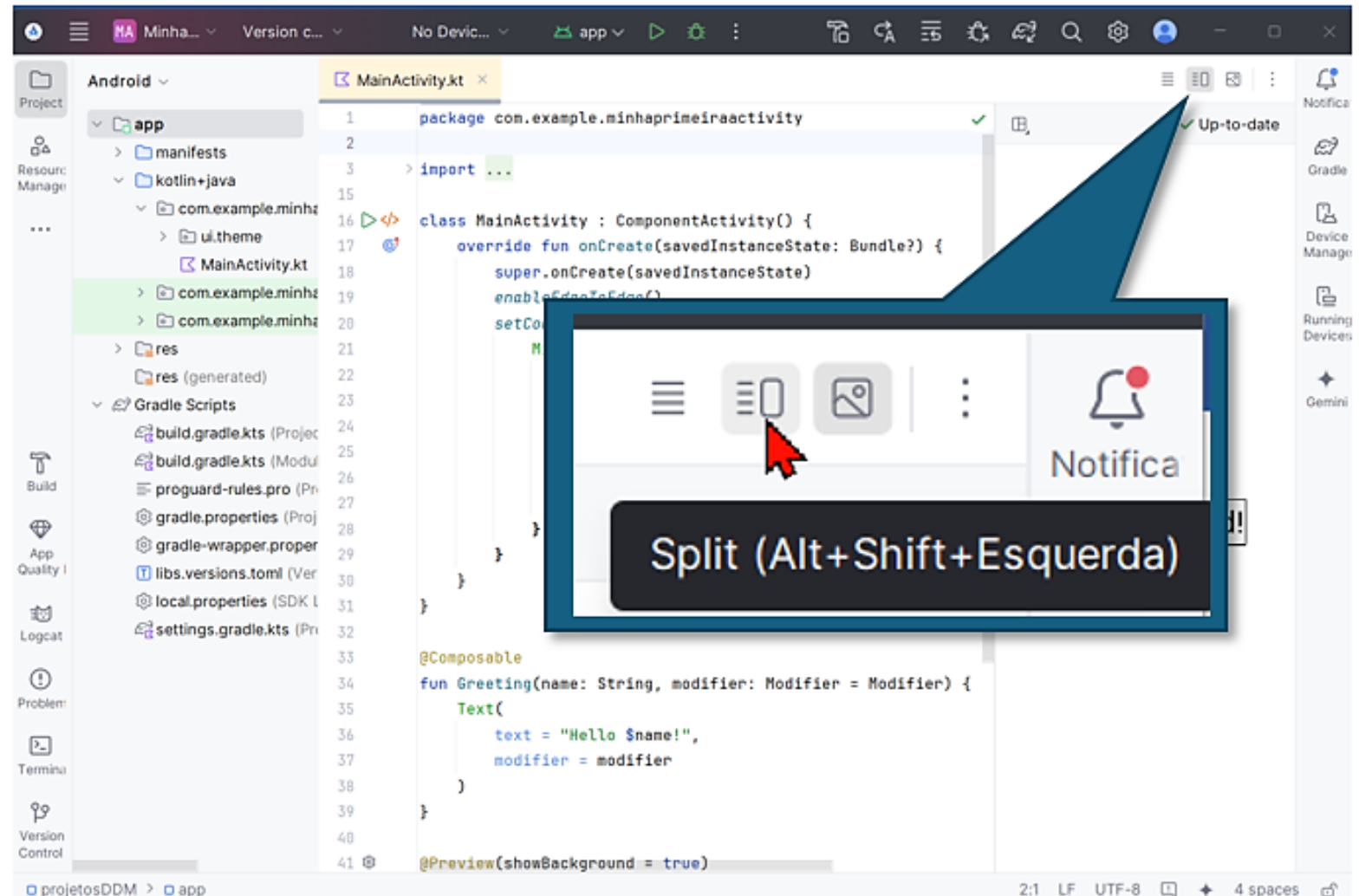
- Aguarde a construção terminar e o estado mudar para "Up to Date".



Interface do Android Studio

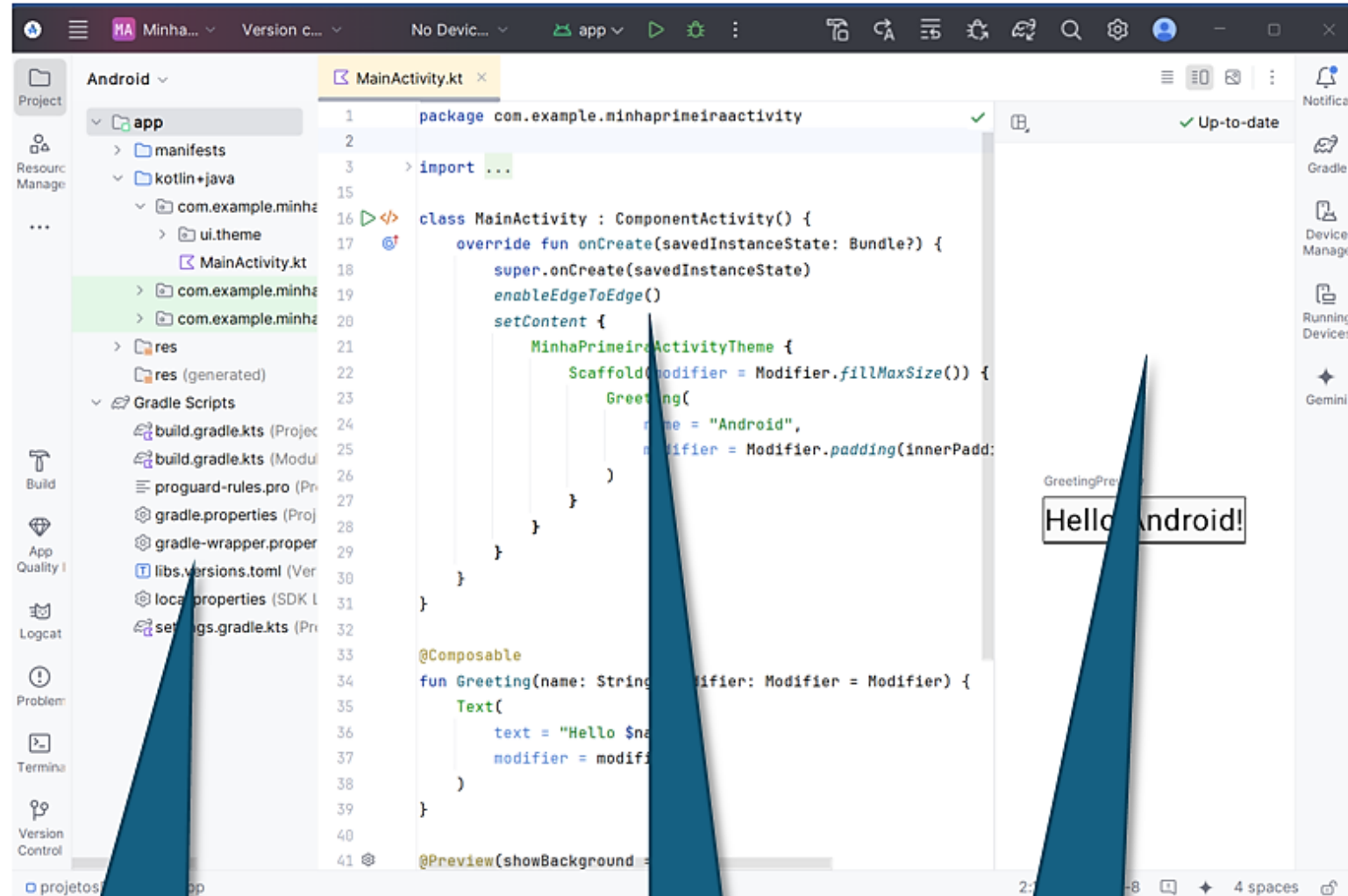
Interface do Android Studio

- Visualização Padrão
 - Clique em “Split” no botão de layout do editor para a visualização padrão.



Interface do Android Studio

- Três Telas Principais
 - Visualização do Projeto: Exibe arquivos e pastas do projeto.
 - Visualização do Código: Onde você edita o código.
 - Visualização de Design: Permite ver a aparência do aplicativo.



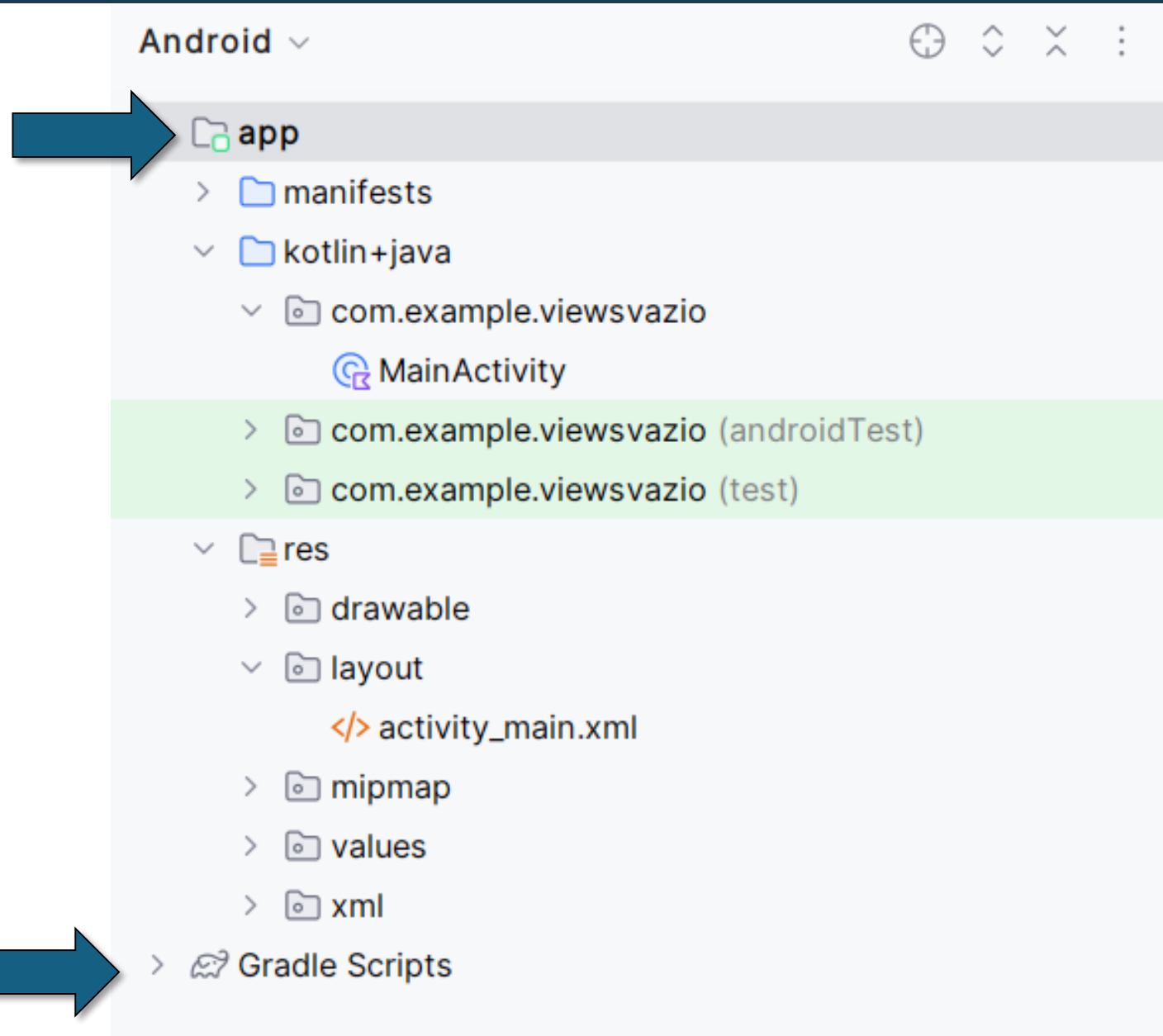
Projeto

Código

Design

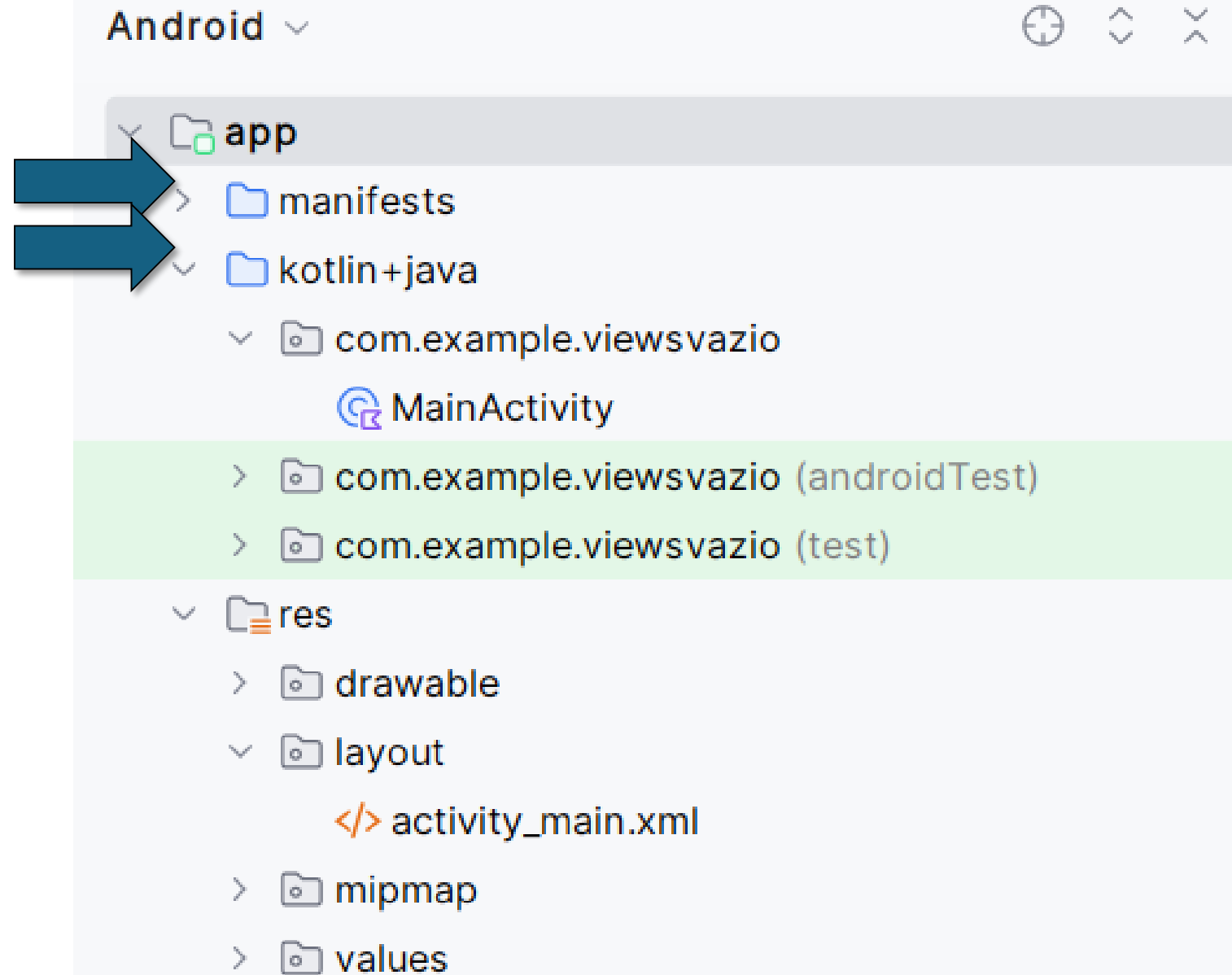
Interface do Android Studio

- Estrutura do Projeto
- Visualização de Projetos Android: Organizada por módulos.
- Gradle Scripts: Arquivos de build no nível superior.



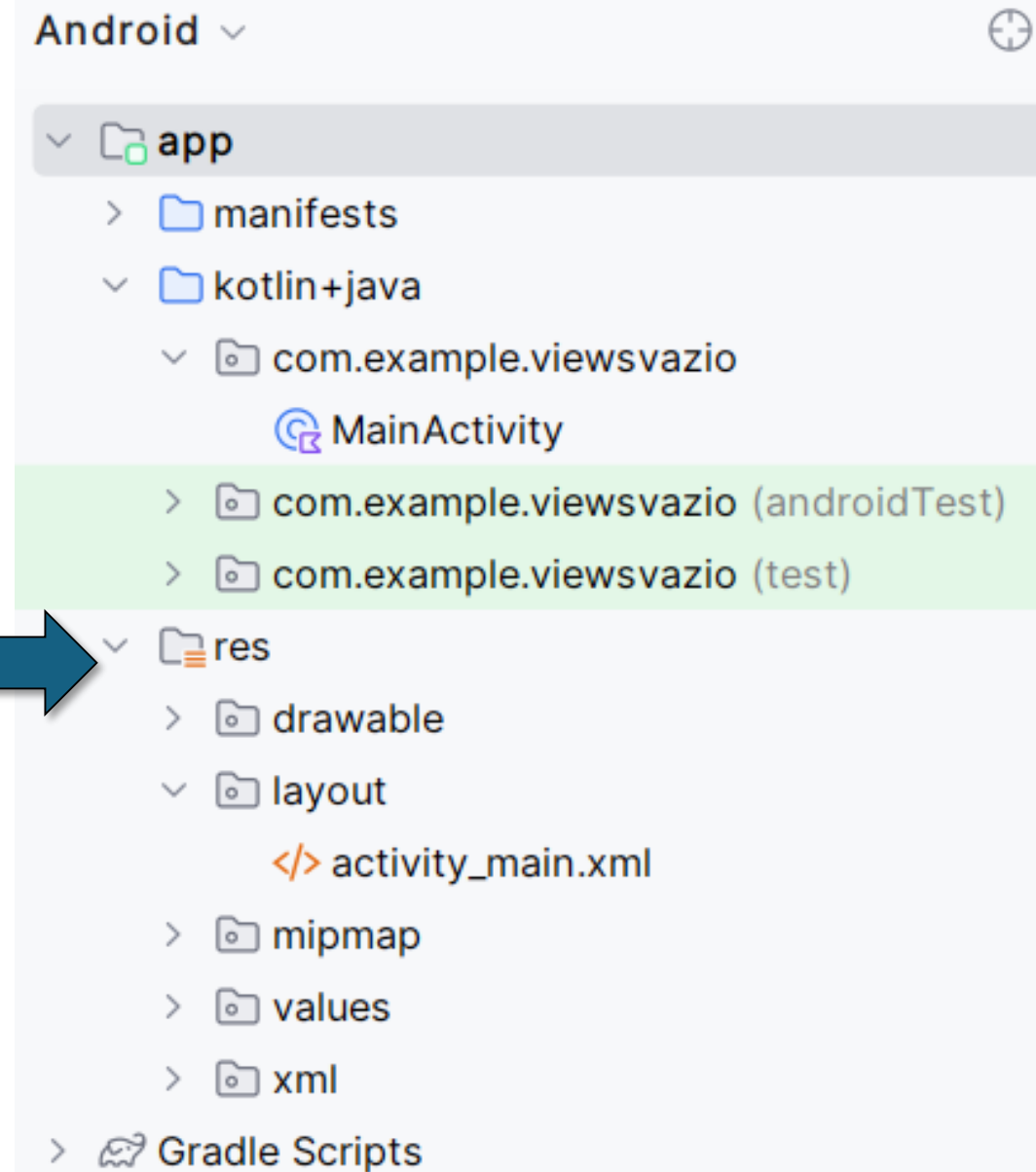
Interface do Android Studio

- Pastas do Módulo de Aplicativo:
 - manifests: *Contém `AndroidManifest.xml`, declara componentes e permissões.*
 - kotlin + java: Arquivos de código-fonte em Kotlin e Java.



Interface do Android Studio

- Pastas do Módulo de Aplicativo (Cont.):
 - *res*: Recursos não relacionados a código.
 - *drawable*: Imagens e recursos gráficos.
 - *layout*: Arquivos XML dos layouts.
 - *mipmap*: Ícones do aplicativo.
 - *values*: Strings, cores, dimensões e estilos.
 - *menu*: Arquivos XML dos menus.
 - *anim*: Animações.
 - *raw*: Arquivos brutos, como áudio ou fontes.



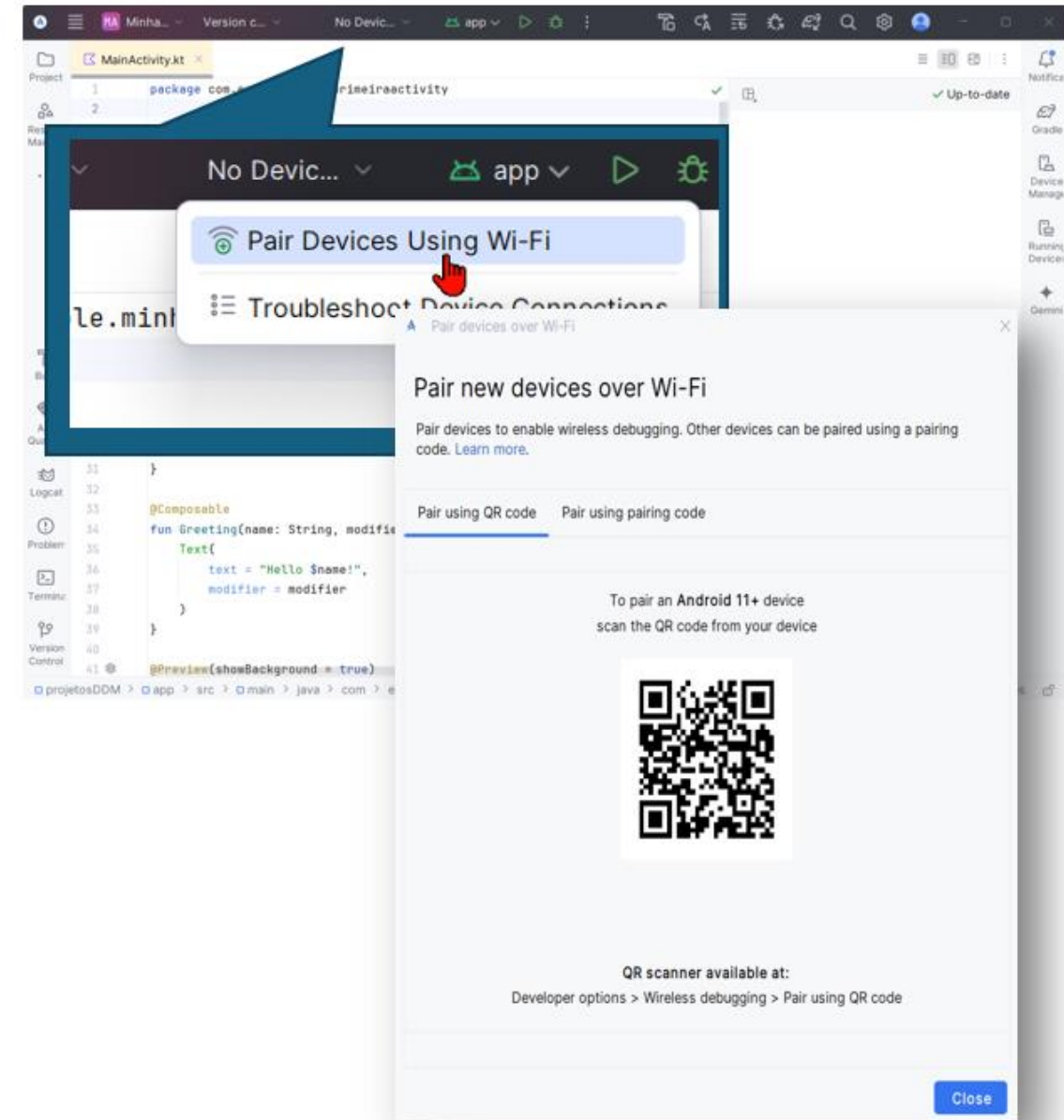
Utilizando o Emulador no Android Studio

Utilizando o Emulador no Android Studio

- O que é o Emulador?
 - Ferramenta essencial para desenvolvedores Android.
 - Simula diversos dispositivos Android no computador.
 - Funciona como um smartphone virtual.
- Vantagens do Emulador
 - Testa aplicativos em diferentes dispositivos e versões do Android.
 - Simula vários tamanhos de tela, resoluções e fabricantes.
 - Facilita a depuração do aplicativo.
 - Métodos de Emulação
 - *Espelhamento de Dispositivo Físico:*
 - *Conectado via USB ou Wi-Fi.*
 - *Economiza memória e CPU do computador.*
 - Android Virtual Device (AVD):
 - *Configuração que define as características de um emulador.*

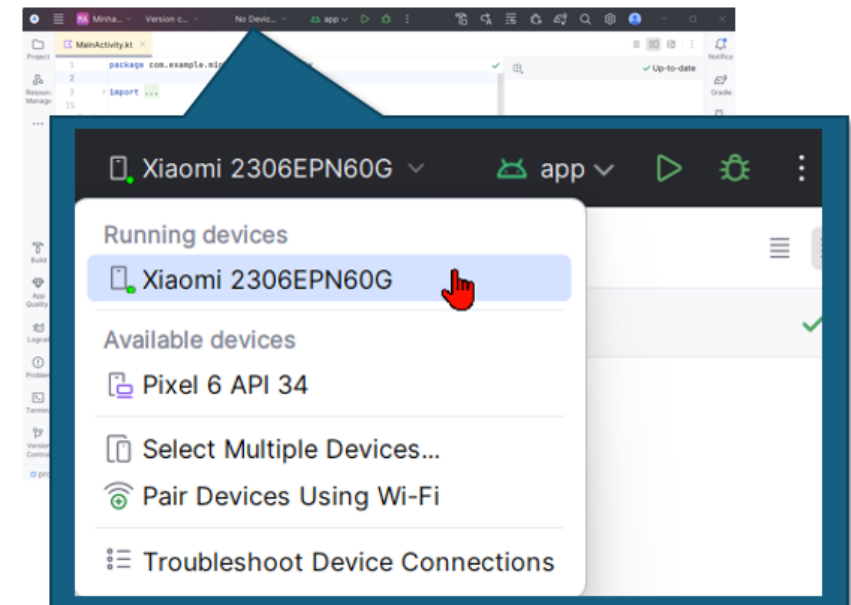
Espelhamento via Wi-Fi

- Ambos dispositivos conectados à mesma rede Wi-Fi.
- Escolha “Pair Devices Using Wi-Fi” e escaneie o QR code.



Espelhamento via Cabo USB

- Necessário driver de conexão e autorização para depuração USB.
 - Ativar modo desenvolvedor no dispositivo:
 - 1. Abra as Configurações.
 - 2. Vá para “Sobre o Telefone”.
 - 3. Toque repetidamente em “Número da Versão” até ativar o modo desenvolvedor.
 - 4. Volte para Configurações e busque “Depurador USB”.
 - Conecte o dispositivo ao computador e permita o uso de dados.
 - Dispositivo disponível na lista de dispositivos.



Espelhamento via Android Virtual Device (AVD)

- O que é o AVD?
 - Configuração que define as características de um emulador Android.
 - Permite simular diferentes dispositivos e testar aplicativos em diversos cenários.



Interatividade

Qual botão deve ser clicado para deixar o editor de código e o layout simultaneamente na visualização padrão no Android Studio?

- a) Design.
- b) Code.
- c) Split.
- d) Project.
- e) Layout.

Resposta

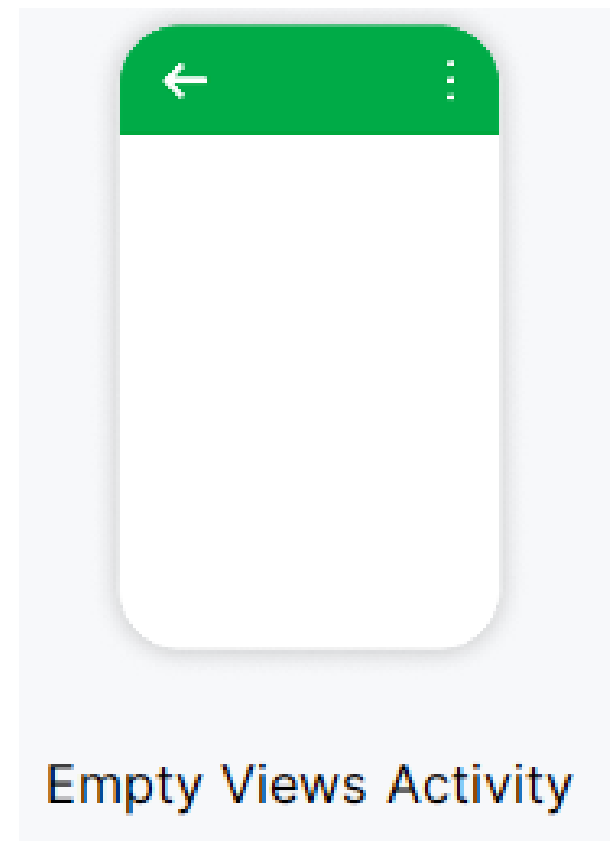
Qual botão deve ser clicado para deixar o editor de código e o layout simultaneamente na visualização padrão no Android Studio?

- a) Design.
- b) Code.
- c) Split.
- d) Project.
- e) Layout.

Interface Gráfica XML no Android Studio

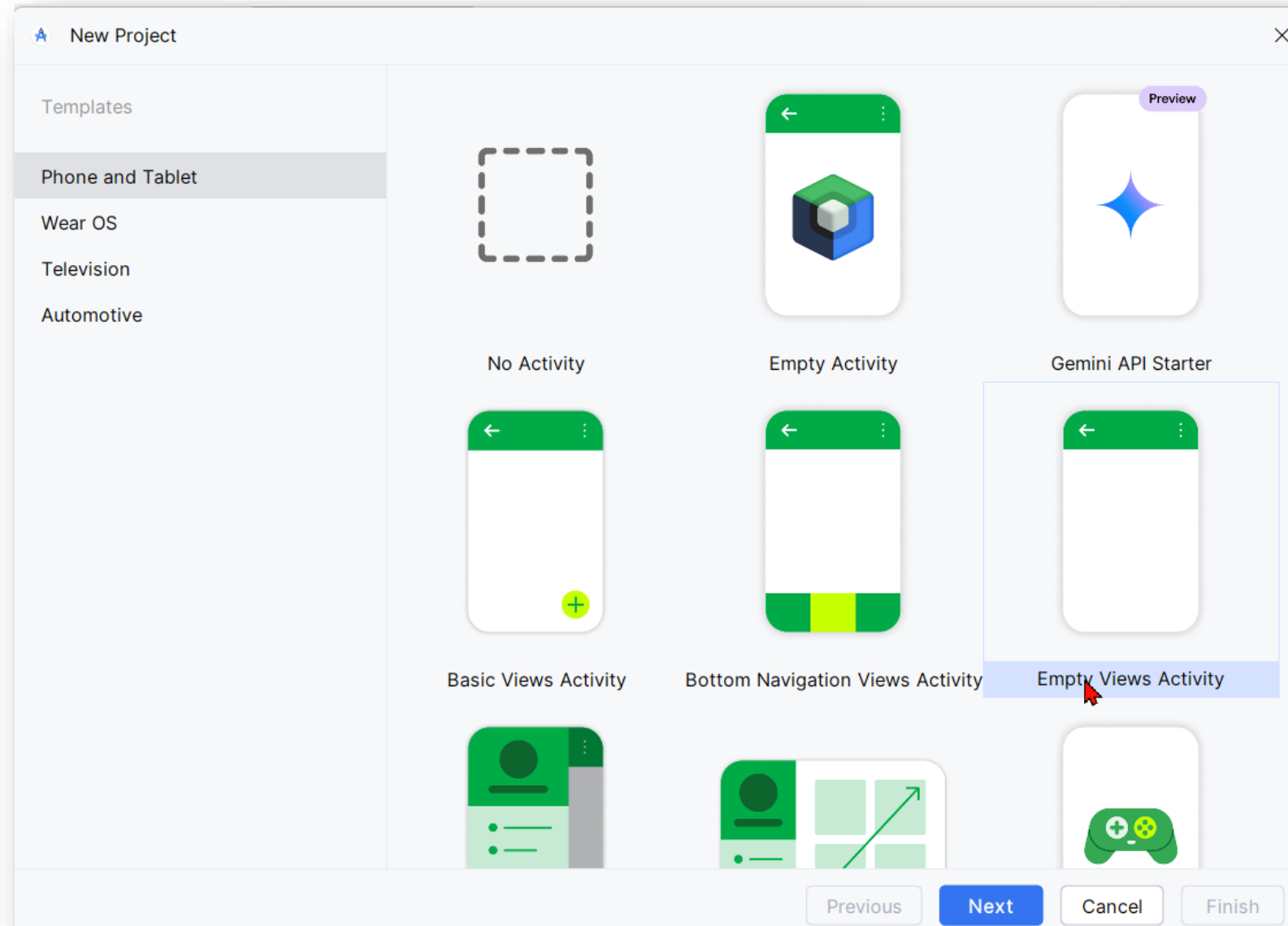
Interface Gráfica XML

- Views em Aplicativos Android:
 - *Elementos visuais com os quais o usuário interage.*
 - Criadas e manipuladas visualmente (arquivos XML) e programaticamente (Java/Kotlin).
 - Permitem construir interfaces complexas e personalizadas.



Interface Gráfica XML

- Iniciando um Projeto com Views
 - Inicie um novo projeto em “new Project”.
 - Selecione o template “Empty Views Activity”



Interface Gráfica XML

- Configuração Inicial:
 - Preencha a tela de propriedades iniciais.
 - Altere o nome para 'Minha Activity Views'.
 - Deixe os outros campos inalterados.
 - Clique em 'finish'.
- Sincronização do Gradle:
 - Aguarde o Gradle terminar a sincronização.

New Project

Empty Views Activity

Creates a new empty activity

Name: Minha Activity Views

Package name: com.example.minhaactivityviews

Save location: E:\projetosDDM\minhaactivityviews

Language: Kotlin

Minimum SDK: API 24 ("Nougat"; Android 7.0)

Information: Your app will run on approximately 97,4% of devices. [Help me choose](#)

Build configuration language: Kotlin DSL (build.gradle.kts) [Recommended]

Previous Next Cancel Finish

activity_main.xml MainActivity.kt

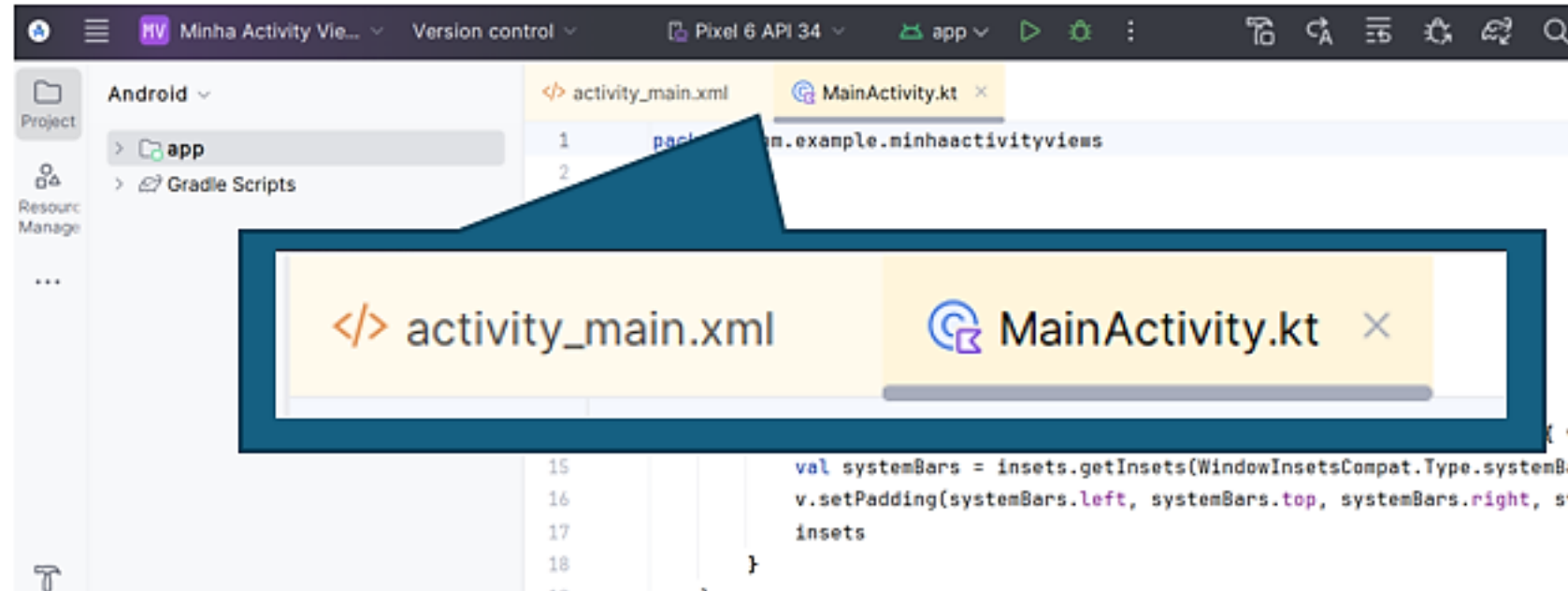
Gradle project sync in progress...

```
1 package com.example.minhaactivityviews
```

Interface no Android Studio do View

Interface no Android Studio

- Arquivo activity_main.xml.
- Estrutura da interface gráfica.
- Arquivo MainActivity.kt.
- Lógica do aplicativo.



Introdução ao MainActivity.kt

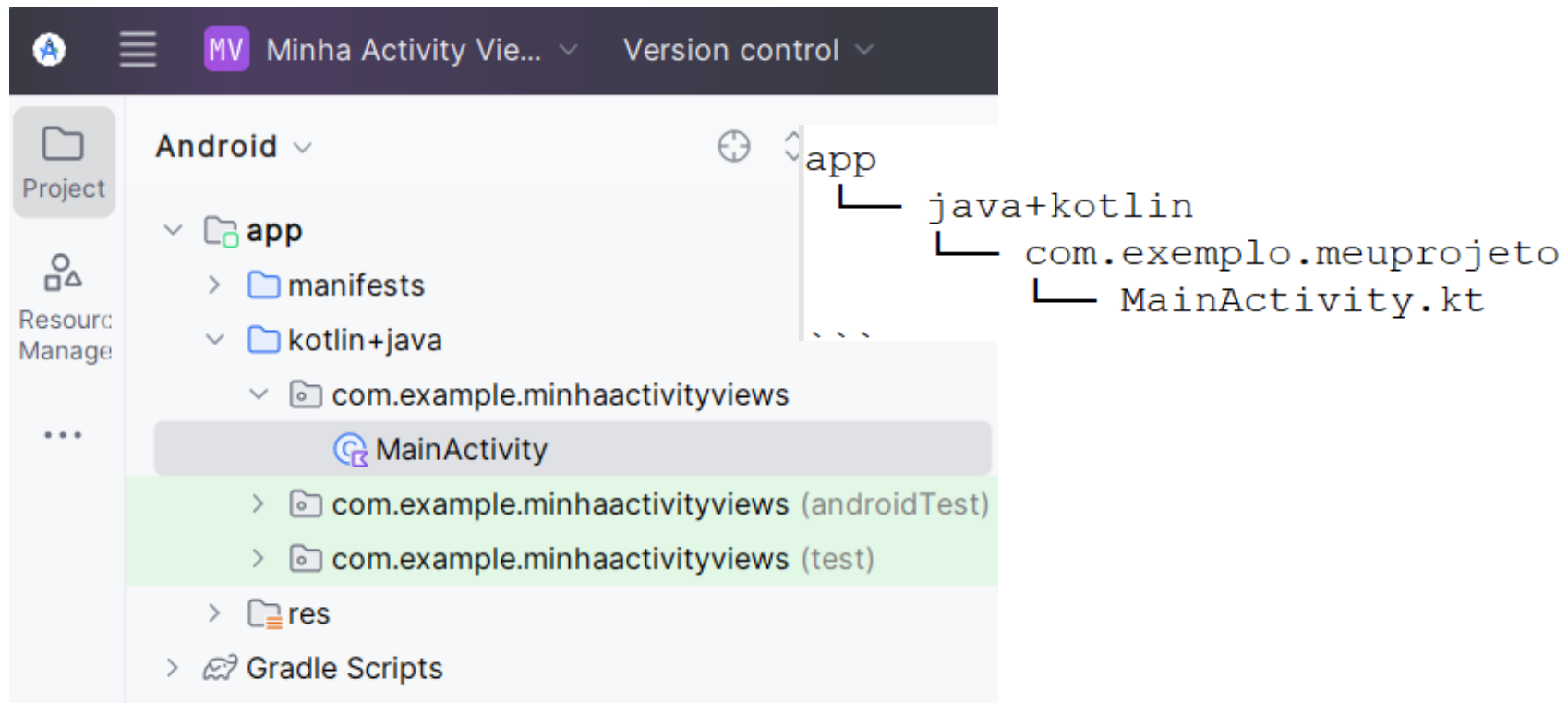
MainActivity.kt

```
</> activity_main.xml  MainActivity.kt x
1  package com.example.minhaactivityviews
2
3  > import ...
8
9  class MainActivity : AppCompatActivity() {
10     override fun onCreate(savedInstanceState: Bundle?) {
11         super.onCreate(savedInstanceState)
12         enableEdgeToEdge()
13         setContentView(R.layout.activity_main)
14         ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v, insets ->
15             val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
16             v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)
17             insets
18         }
19     }
20 }
```

- Arquivo fundamental em projetos Android
 - Representa a atividade principal do aplicativo.
 - Conecta a interface do usuário à lógica do código.
 - Localizado no diretório do projeto no Android Studio.

Acessando MainActivity.kt

- Localização no diretório do projeto.
- Duplo clique para abrir na árvore do projeto.



Introdução ao activity_main.xml

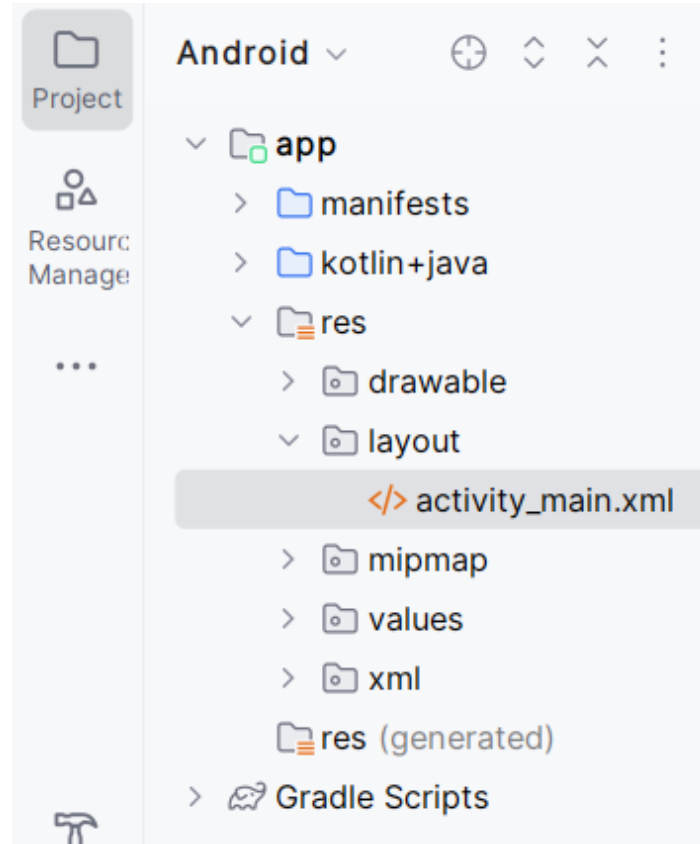
activity_main.xml

- Define a interface do usuário para a atividade principal.
- Escrito em XML.
- Inclui elementos como botões, textos, imagens.



activity_main.xml

- Localizado na pasta layout dos recursos do projeto.



```
app
├── res
│   └── layout
│       └── activity_main.xml
```

Modos de Edição do XML

- Code: Exibe apenas o código XML.



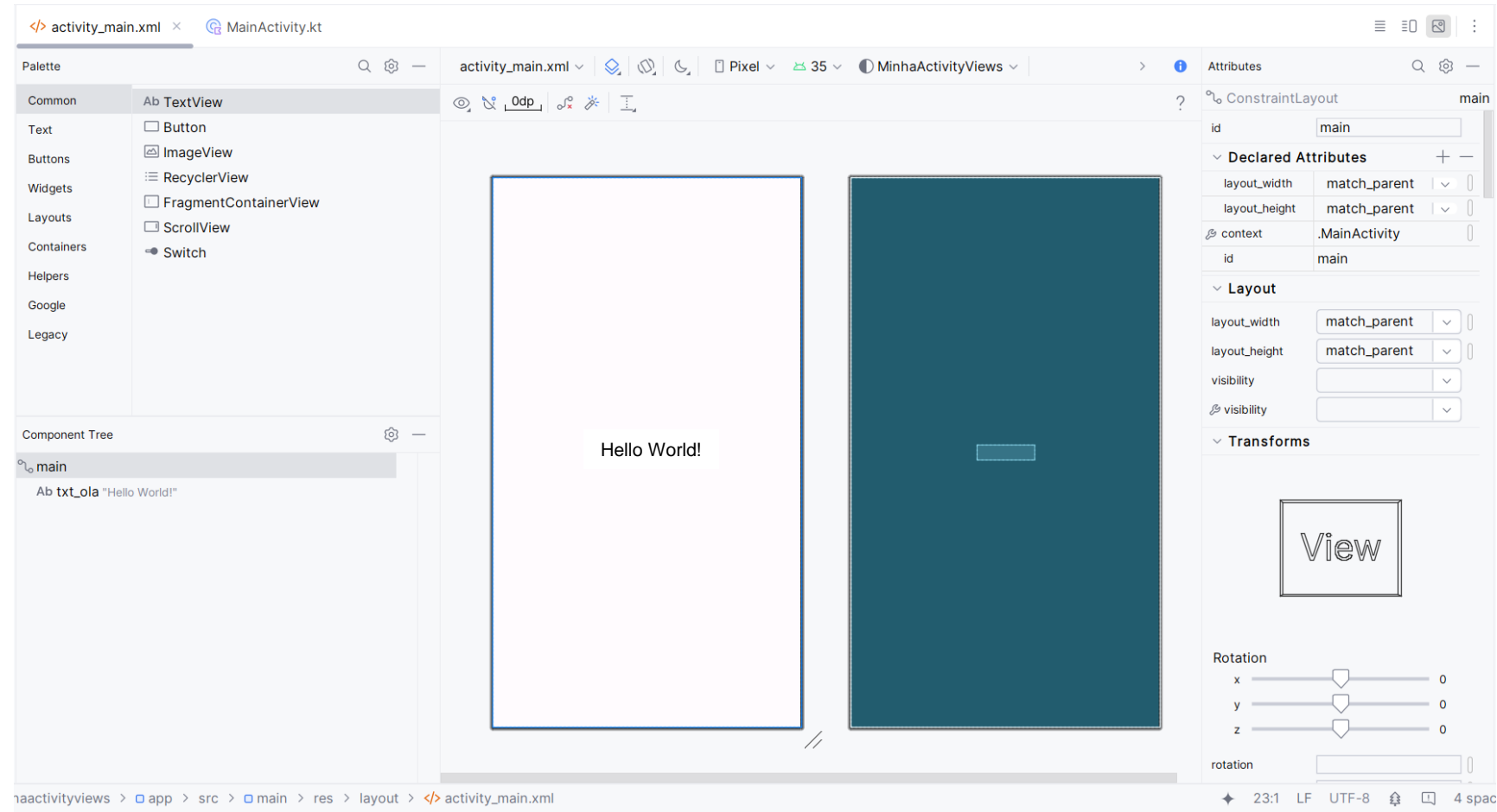
The screenshot shows an IDE window with two tabs: 'activity_main.xml' and 'MainActivity.kt'. The 'activity_main.xml' tab is active, displaying XML code in a 'Code' mode. The code is as follows:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:app="http://schemas.android.com/apk/res-auto"
4     xmlns:tools="http://schemas.android.com/tools"
5     android:id="@+id/main"
6     android:layout_width="match_parent"
7     android:layout_height="match_parent"
8     tools:context=".MainActivity">
9
10     <TextView
11         android:id="@+id/txt_ola"
12         android:layout_width="wrap_content"
13         android:layout_height="wrap_content"
14         android:text="Hello World!"
15         app:layout_constraintBottom_toBottomOf="parent"
16         app:layout_constraintEnd_toEndOf="parent"
17         app:layout_constraintStart_toStartOf="parent"
18         app:layout_constraintTop_toTopOf="parent" />
19
20 </androidx.constraintlayout.widget.ConstraintLayout>
21
22
23
```

On the right side of the IDE, there is a toolbar with icons for undo, redo, and search. Below these icons is a 'Code' button, which is highlighted with a red mouse cursor. The 'Code' button is a dark grey rectangle with the word 'Code' in white text.

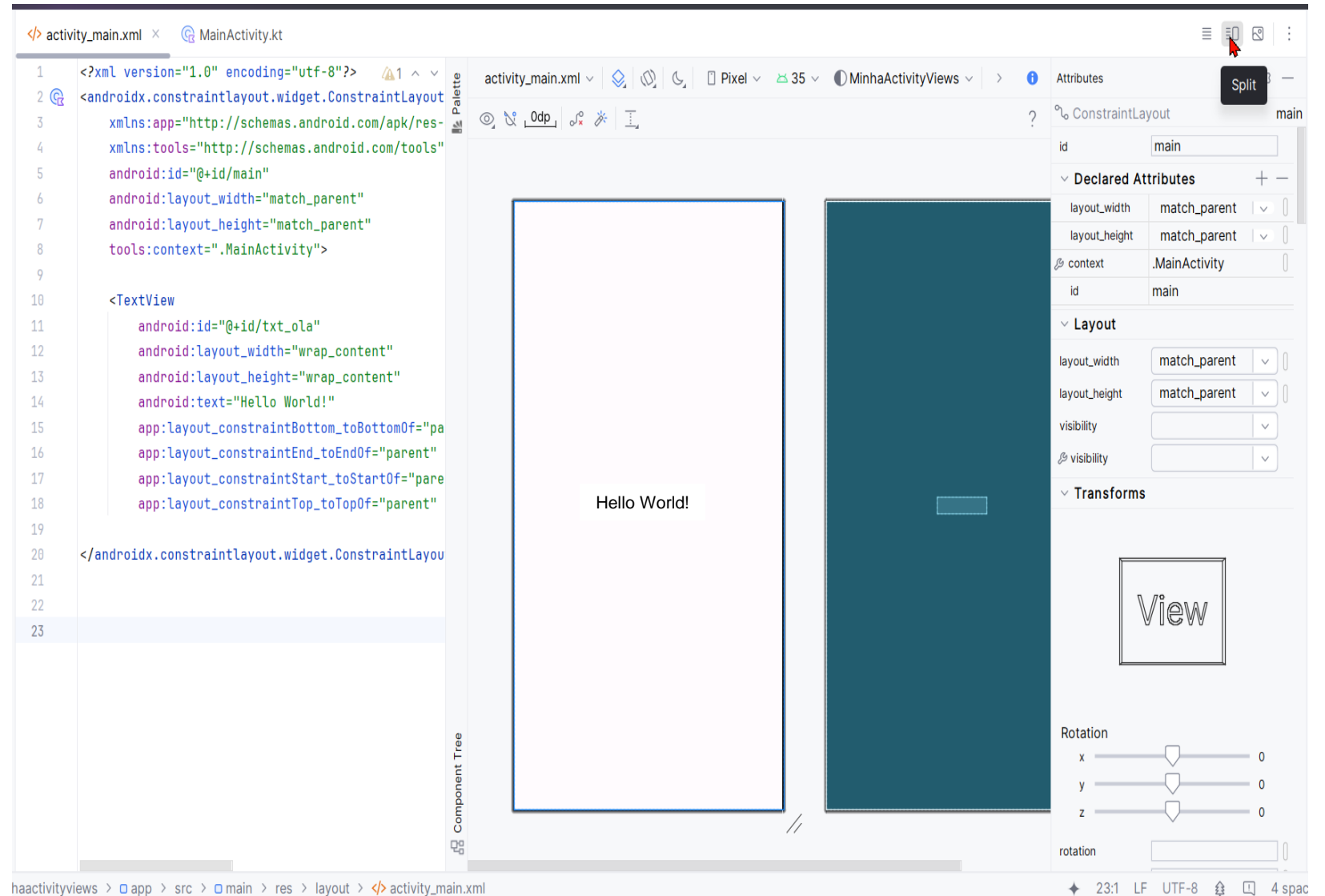
Modos de Edição do XML

- Design: Interface visual para arrastar e soltar componentes.



Modos de Edição do XML

- Split: Combina Code e Design.



Estrutura do activity_main.xml

- Declaração da versão do XML.
- Tag mais externa define o layout (ConstraintLayout).
- Elementos internos como TextView.

```
</> activity_main.xml x MainActivity.kt
1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res-auto"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:id="@+id/main"
6      android:layout_width="match_parent"
7      android:layout_height="match_parent"
8      tools:context=".MainActivity">
9
10     <TextView
11         android:layout_width="wrap_content"
12         android:layout_height="wrap_content"
13         android:text="Hello World!"
14         app:layout_constraintBottom_toBottomOf="parent"
15         app:layout_constraintEnd_toEndOf="parent"
16         app:layout_constraintStart_toStartOf="parent"
17         app:layout_constraintTop_toTopOf="parent" />
18
19 </androidx.constraintlayout.widget.ConstraintLayout>
```

Estrutura do activity_main.xml

- Tag mais externa define o layout (ConstraintLayout).

```
<?xml version="1.0" encoding="utf-8"?>  
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android:  
    xmlns:app="http://schemas.android.com/apk/res-auto"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:id="@+id/main"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    tools:context=".MainActivity">
```

Conteúdo do Constraint Layout

```
</androidx.constraintlayout.widget.ConstraintLayout>
```

Estrutura do activity_main.xml

- Elementos internos como TextView.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:an
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/main"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".MainActivity">
```

```
<TextView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Hello World!"
  app:layout_constraintBottom_toBottomOf="parent"
  app:layout_constraintEnd_toEndOf="parent"
  app:layout_constraintStart_toStartOf="parent"
  app:layout_constraintTop_toTopOf="parent" />
```

```
</androidx.constraintlayout.widget.ConstraintLayout>
```


Views em Android

Views

- View: Ponto de partida para criar a interface de um aplicativo Android.
- Widgets: Tipos especializados de Views (ex.: botões, caixas de texto).
- Layouts: ViewGroups que organizam outras Views (ex.: LinearLayout, RelativeLayout).
- Hierarquia: Permite criar interfaces complexas e personalizadas.

Componentes Essenciais no Desenvolvimento Android

Componentes Essenciais

- **TextView:**
 - Exibe texto na tela.
 - Personalizável em fonte, tamanho, cor e alinhamento.
 - Suporta formatação de texto, links clicáveis e ajuste automático do tamanho da fonte.

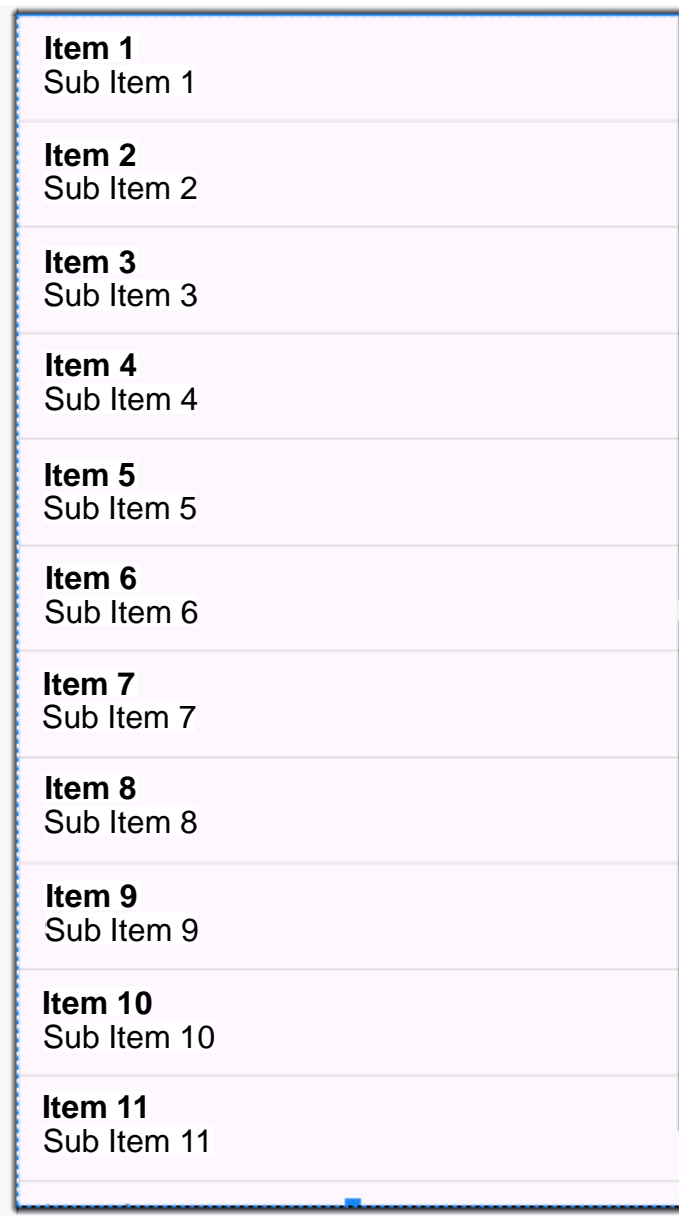


```
txtTexto.text = "Texto alterado"
```



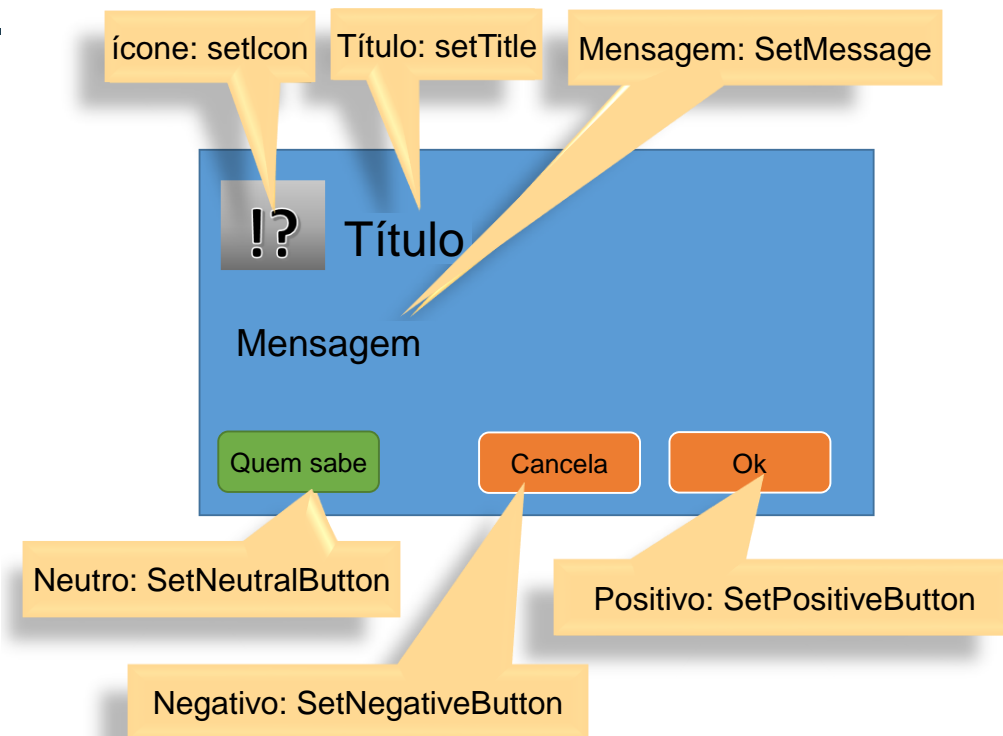
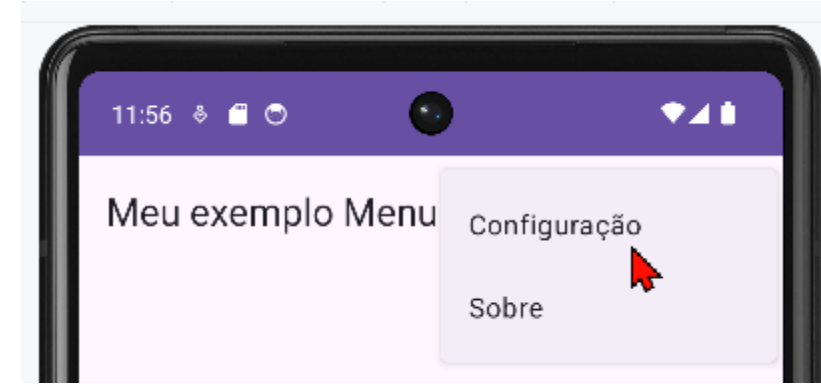
Componentes Essenciais

- ListView:
 - Exibe listas de itens de maneira organizada.
 - Personalizável, permite definir layout, eventos de clique e rolagem.
 - Usa Adapter para inserir itens automaticamente.



Componentes Essenciais

- Menu:
 - Cria menus personalizados e organizados.
 - Tipos: Menu de Opções, Menu de Contexto, Menu Popup.
 - Usa MenuInflater para transformar XML em objetos Menu.
- AlertDialog:
 - *Cria caixas de diálogo personalizadas.*
 - Não requer layout XML específico.
 - Configurável diretamente no código.



Componentes Essenciais

- **LinearLayout:**
 - *Organiza elementos de forma linear (horizontal ou vertical).*
 - Atributos: `android:orientation`, `android:layout_weight`.
 - Ideal para layouts simples e intuitivos.
- **RelativeLayout:**
 - Posiciona elementos em relação a outros elementos ou aos limites do layout pai.
 - Atributos: `android:layout_above`, `android:layout_below`, `android:layout_toLeftOf` etc.
 - Flexível para layouts complexos e personalizados.
 - **TableLayout:**
 - Organiza elementos em uma estrutura tabular.
 - Divide o conteúdo em linhas (`TableRow`) e colunas.
 - Cada célula pode conter diversos tipos de elementos visuais.

Interatividade

Qual é a classe base para todos os elementos visuais em um aplicativo Android?

- a) Activity.
- b) View.
- c) ViewGroup.
- d) Widget.
- e) Layout.

Resposta

Qual é a classe base para todos os elementos visuais em um aplicativo Android?

- a) Activity.
- b) View.**
- c) ViewGroup.
- d) Widget.
- e) Layout.

Jetpack Compose no Desenvolvimento Android

Jetpack Compose no Desenvolvimento Android

- Definição: Nova forma de criar interfaces do usuário para Android.
- Vantagens:
 - Menos código, mais resultados.
 - Abordagem declarativa e componentes pré-construídos.
 - Facilita a criação de UIs personalizadas e responsivas.
 - Benefícios:
 - Simplificação do ciclo de desenvolvimento.
 - Foco na lógica central.
 - Redução do tempo de teste e depuração.
 - Menor complexidade, facilitando manutenção e colaboração.
 - Comparação com Abordagem Antiga:
 - Atualização manual e complexa da interface.
 - Maior propensão a erros e dificuldade na manutenção.

Estrutura do Código

```
class MainActivity : ComponentActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        enableEdgeToEdge()  
        setContent {  
            MeuExemploComposeLayout1Theme {  
                Scaffold(modifier = Modifier.fillMaxSize()) {  
                    innerPadding -> Greeting(  
                        "Android",  
                        Modifier.padding(innerPadding)  
                    )  
                }  
            }  
        }  
    }  
}
```

chamada da função

```
@Composable  
fun Greeting(name: String, modifier: Modifier = Modifier) {  
    Text(  
        "Hello $name!",  
        modifier  
    )  
}
```

função Composable

```
@Preview(showBackground = true)  
@Composable  
fun GreetingPreview() {  
    MeuExemploComposeLayout1Theme {  
        Greeting("Android")  
    }  
}
```

função Preview

Estrutura do Código

- Função Composable Greeting



```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Text(
        text = "Hello $name!",
        modifier = modifier
    )
}
```

Introdução ao Jetpack Compose

O Paradigma de Programação Declarativa

- Definição:
 - *Descreve “o que” o programa deve fazer, em vez de “como” fazer.*
 - Define propriedades e estado desejado.
- Comparação:
 - Imperativo: Descreve passo a passo como executar uma tarefa.
 - Declarativo:
 - Descreve o que a interface de usuário deve ser.
 - Define o estado desejado da tela.

Material Design

- Definição: Sistema de design criado pelo Google.
- Objetivo:
 - Unificar e padronizar a experiência do usuário.
 - Oferecer diretrizes, componentes e ferramentas para a criação de interfaces de usuário.
- Princípios:
 - Física do mundo real.
 - Tipografia clara.
 - Layouts responsivos.

Componentes do Material Design no Compose

- Componentes Básicos:
 - Text, Image, Icon, Box, Surface.
- Layouts:
 - Column, Row, LazyColumn, LazyRow, Grid.
- Componentes de Interação:
 - Button, TextField, Checkbox, RadioButton, Slider, Switch.
- Componentes de Navegação:
 - BottomNavigation, NavigationRail, TopAppBar.
 - Componentes de Diálogo e Alerta:
 - AlertDialog, Snackbar, DropdownMenu.

Componentes do Material Design no Compose

- Componentes de Progresso:
 - CircularProgressIndicator, LinearProgressIndicator.
- Outros Componentes:
 - Divider, Spacer, DropdownMenuItem, ExposedDropdownMenuBox, OutlinedTextField, Text composable.

Widgets Composable no Jetpack Compose

Widgets Composable

- Definição: Funções em Jetpack Compose para criar interfaces de usuário de forma declarativa.
- Características:
- Anotadas com `@Composable`.
 - Combináveis para criar interfaces complexas e reutilizáveis.
- Parâmetros:
 - Configuram e personalizam comportamento e aparência.
 - Incluem propriedades, como texto, cores, tamanhos, ações de clique.
 - Conteúdo entre Chaves `{}`:
 - Define o comportamento interno do widget.
 - Pode incluir outros widgets composable, funções lambda, ou qualquer código necessário.

Parâmetros dos Widgets Composable

- Configuração e Personalização:
 - texto;
 - cores;
 - tamanhos;
 - ações de clique.

```
widget ( modificador, eventos, localização, etc ... )  
{  
  ações  
outros Composables  
etc  
}
```

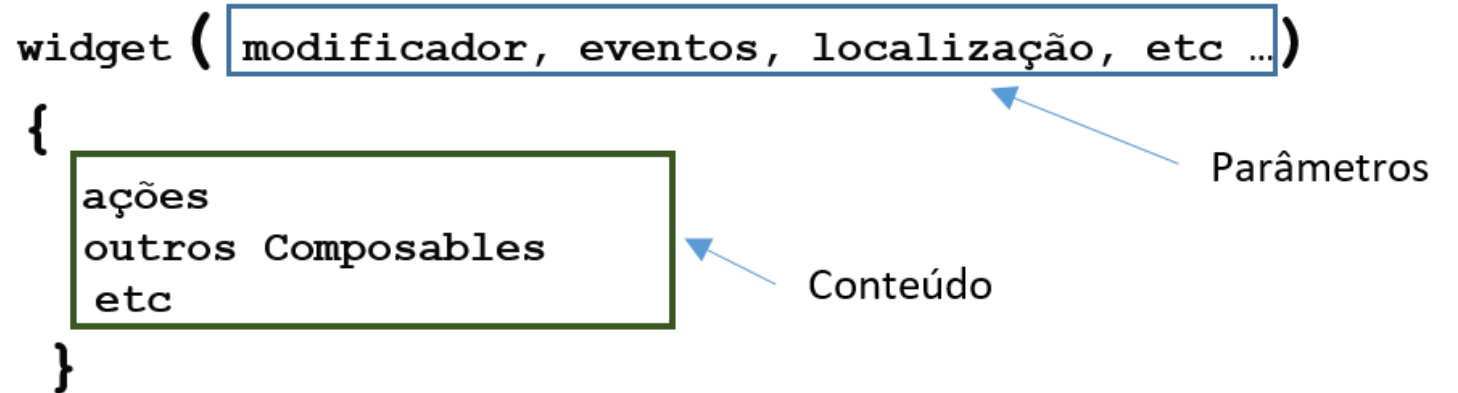


Diagram illustrating the structure of a widget definition. The parameters (modificador, eventos, localização, etc ...) are enclosed in a blue box and labeled "Parâmetros". The content (ações, outros Composables, etc) is enclosed in a green box and labeled "Conteúdo".

```
widget ( modificador,  
eventos,  
localização,  
etc )  
...  
{  
  ações  
outros Composables  
etc  
}
```

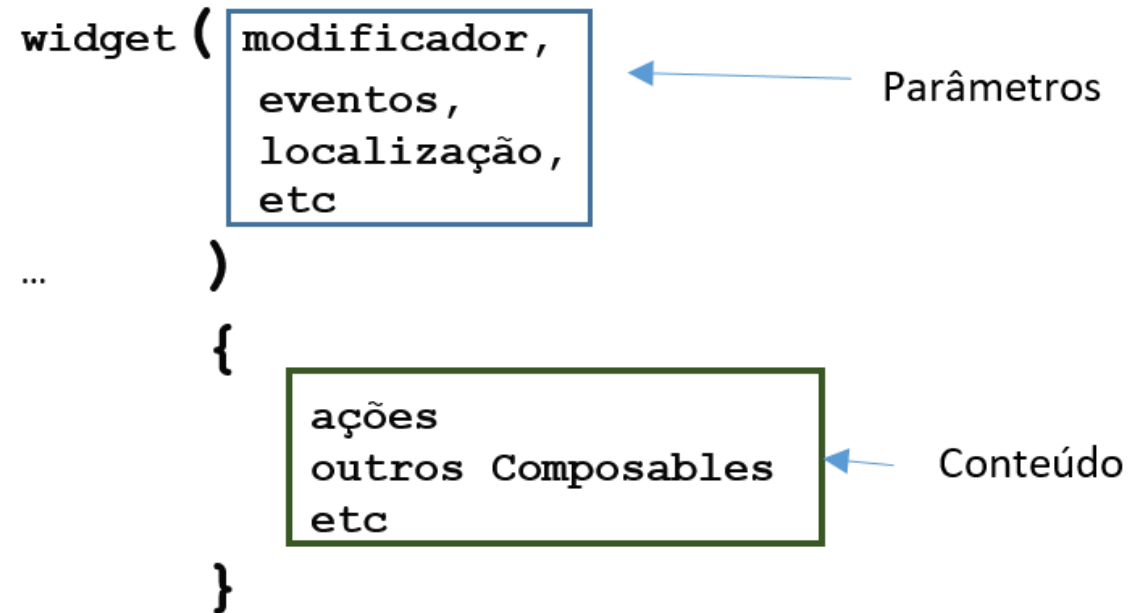


Diagram illustrating the structure of a widget definition. The parameters (modificador, eventos, localização, etc) are enclosed in a blue box and labeled "Parâmetros". The content (ações, outros Composables, etc) is enclosed in a green box and labeled "Conteúdo".

Conteúdo entre Chaves {}

- Definição:
 - Define o comportamento interno do widget.
 - Pode incluir outros widgets composável, funções lambda, ou qualquer código necessário.

```
Card(  
    modifier = Modifier.padding(16.dp),  
    elevation = CardDefaults.cardElevation(defaultElevation = 8.dp)  
) {  
    Column {  
        Text(text = "Título do Card",  
            style = MaterialTheme.typography.titleLarge)  
        Text(text = "Conteúdo do Card")  
        Button(onClick = { /* Lógica do botão */ }) {  
            Text(text = "Clique aqui")  
        }  
    }  
}
```

Interatividade

Qual é a função da anotação `@Composable` no Jetpack Compose?

- a) Define o tema do aplicativo.
- b) Aplica um padding interno aos elementos.
- c) Indica que uma função é composável.
- d) Centraliza os elementos na tela.
- e) Cria uma estrutura básica para a tela.

Resposta

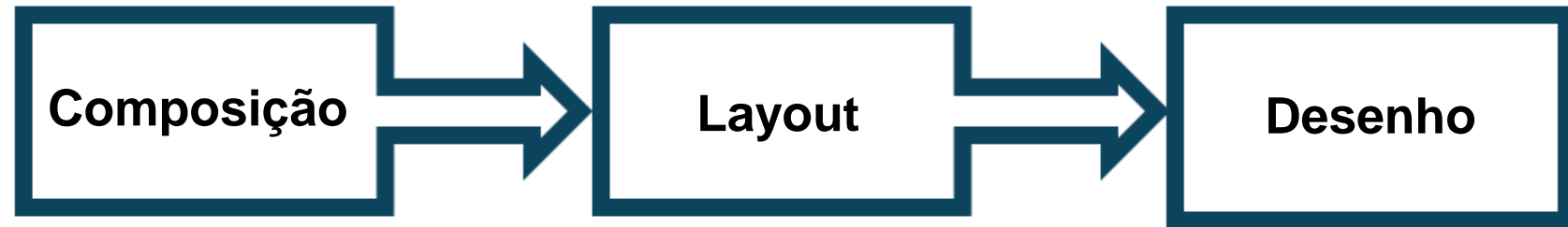
Qual é a função da anotação `@Composable` no Jetpack Compose?

- a) Define o tema do aplicativo.
- b) Aplica um padding interno aos elementos.
- c) Indica que uma função é composável.
- d) Centraliza os elementos na tela.
- e) Cria uma estrutura básica para a tela.

Fundamentos dos Layouts

Fundamentos dos Layouts

- Jetpack Compose: Abordagem poderosa e flexível para criar interfaces de usuário declarativas no Android.
- Pilares do Framework:
 - Composables: Funções que descrevem a UI.
 - Layout: Organiza composables em uma estrutura visual.
 - Modificadores: Alteram o comportamento de um composable.



Funções Composáveis



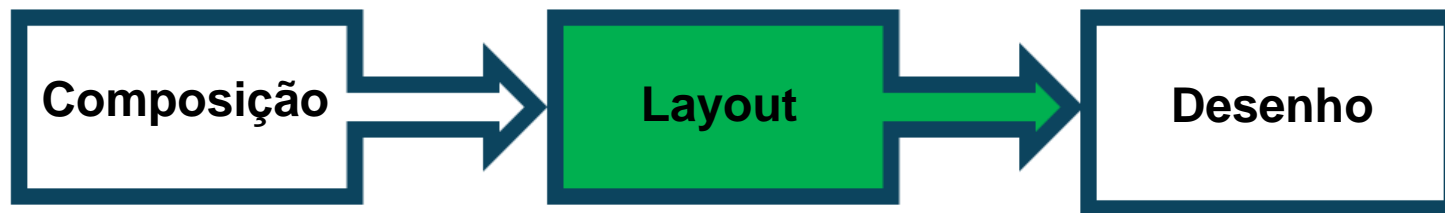
- Definição:
 - Funções composáveis são fundamentais para construir interfaces de usuário no Android Studio Compose.
 - Estrutura da UI definida de forma declarativa, sem XML.
- Blocos de Construção:
 - Funções composáveis combinadas para criar interfaces complexas.

Funções Composáveis

- Exemplo:
 - Função `Greeting` que recebe uma `String` e um `Modifier`, gerando um widget `Text`.

```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Text(
        text = "Hello $name!",
        modifier = modifier
    )
}
```

Principais Layouts em Compose



- Para compreender como as funções Composables funcionam, vamos analisar um exemplo no Android Studio.

```
class MainActivity : ComponentActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        enableEdgeToEdge()  
        setContent {  
            MeuExemploComposeLayout1Theme {  
                Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->  
                    Greeting(  
                        name = "Android",  
                        modifier = Modifier.padding(innerPadding)  
                    )  
                }  
            }  
        }  
    }  
}
```



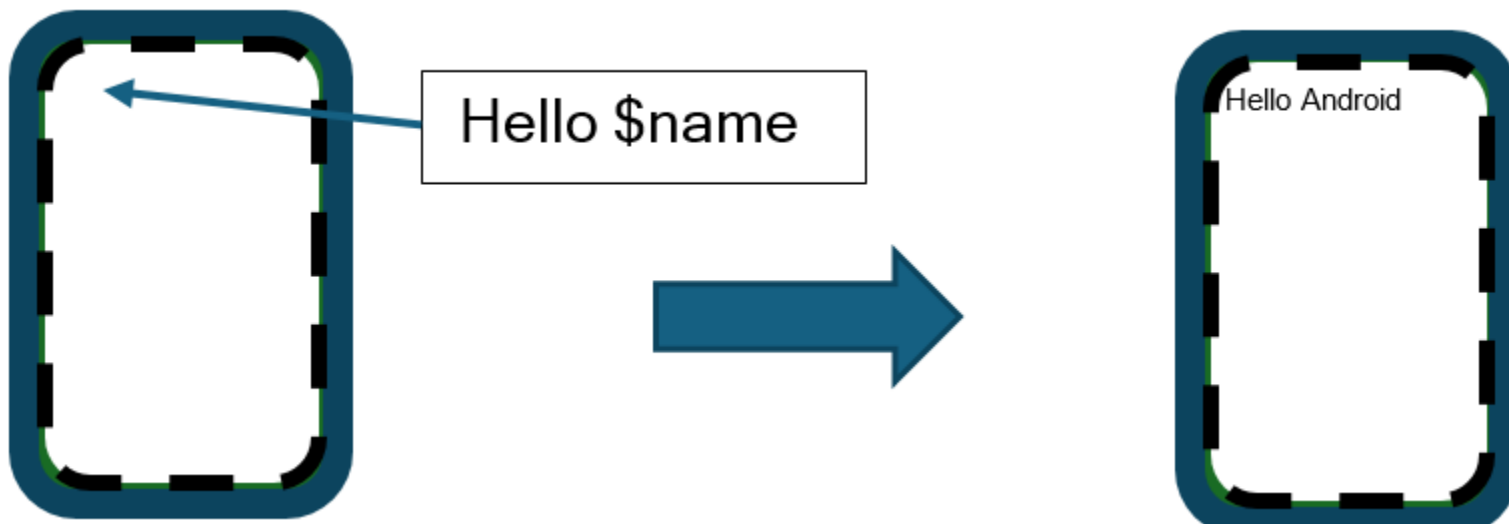
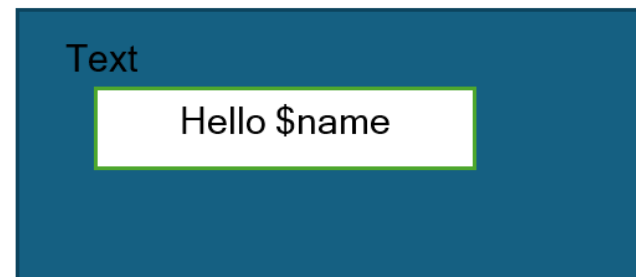
Widget no Layout



`@Composable`

```
fun Greeting(name: String, modifier: Modifier = Modifier) {  
    Text(  
        text = "Hello $name!",  
        modifier = modifier  
    )  
}
```

Greeting



Alteração da Função `Greeting`

```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Text(
        text = "Hello $name!",
        modifier = modifier
    )
    Text(
        text = "Outro texto",
        modifier = modifier
    )
    Text(
        text = "e um terceiro texto",
        modifier = modifier
    )
}
```

GreetingPreview

Hello Anchor!texto

Principais Layouts em Compose

Principais Layouts em Compose

- Column:
 - Componente essencial para layouts verticais.
 - Organiza elementos em uma pilha vertical.
 - Oferece opções de personalização como alinhamento e espaçamento.



```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Column {
        Text(
            text = "Hello $name!",
            modifier = modifier
        )
        Text(
            text = "Outro texto",
            modifier = modifier
        )
        Text(
            text = "e um terceiro texto",
            modifier = modifier
        )
    }
}
```

GreetingPreview

Hello Android!
Outro texto
e um terceiro texto

Principais Layouts em Compose

- Row:
 - Componente para layouts horizontais.
 - Organiza elementos em uma linha.
 - Oferece opções de personalização como alinhamento e espaçamento.



```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Row {
        Text(
            text = "Hello $name!",
            modifier = modifier
        )
        Text(
            text = "Outro texto",
            modifier = modifier
        )
        Text(
            text = "e um terceiro texto",
            modifier = modifier
        )
    }
}
```

GreetingPreview

Hello Android!Outro textoe um terceiro texto

Trabalhando com Layouts – Decompondo em Widgets

```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    linha(name = name, modifier = modifier)
}
```

GreetingPreview

Hello Android! Outro texto
e um terceiro texto

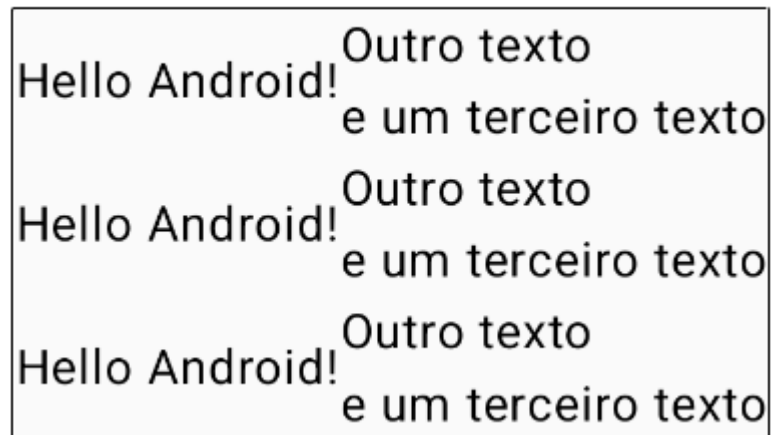
```
@Composable
fun linha(name: String, modifier: Modifier = Modifier) {
    Row(
        verticalAlignment = Alignment.CenterVertically
    ) {
        Text(
            text = "Hello $name!",
            modifier = modifier
        )
        Column {
            Text(
                text = "Outro texto",
                modifier = modifier
            )
            Text(
                text = "e um terceiro texto",
                modifier = modifier
            )
        }
    }
}
```

Trabalhando com Layouts – Decompondo em Widgets

```
@Composable
```

```
fun Greeting(name: String, modifier: Modifier = Modifier) {  
    Column {  
        linha(name = name, modifier = modifier)  
        linha(name = name, modifier = modifier)  
        linha(name = name, modifier = modifier)  
    }  
}
```

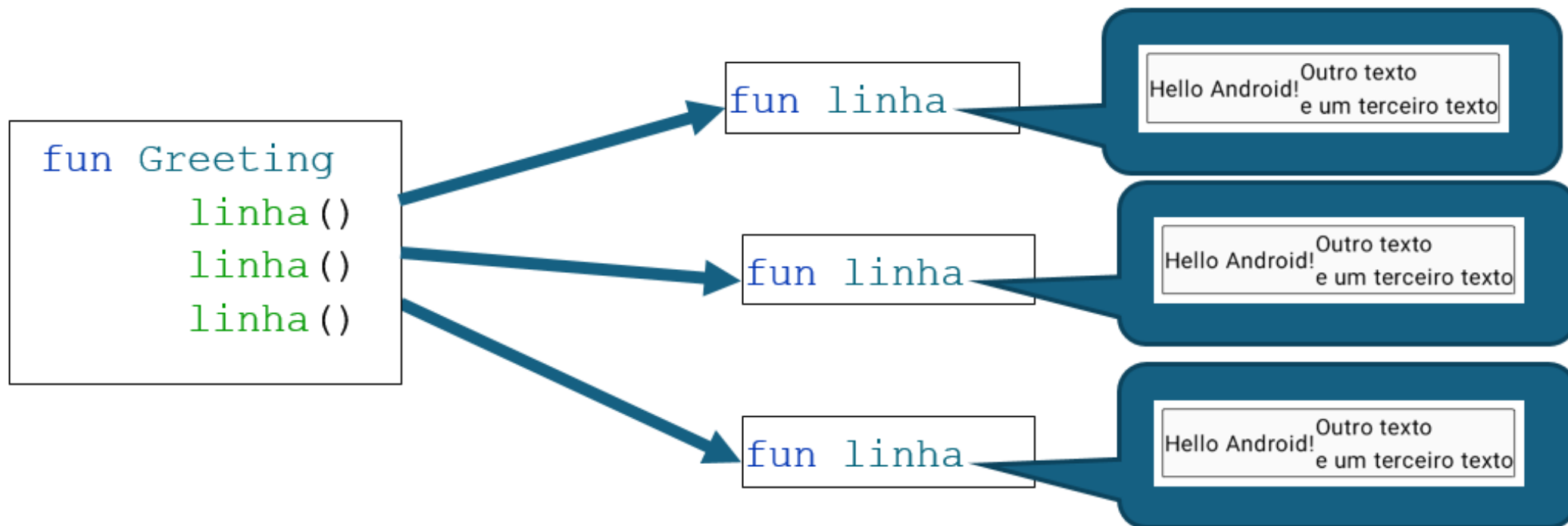
GreetingPreview



The preview shows a rectangular box containing three rows of text. Each row consists of two parts: a greeting 'Hello Android!' followed by a description 'Outro texto e um terceiro texto'. The text is left-aligned and the rows are stacked vertically with some spacing between them.

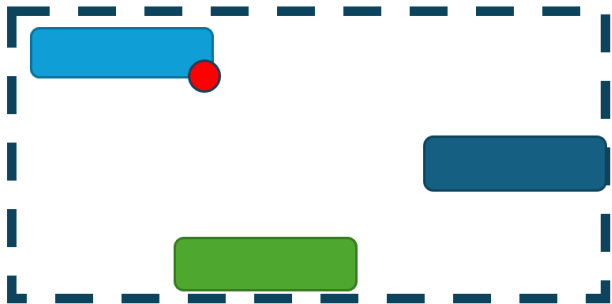
Hello Android!	Outro texto e um terceiro texto
Hello Android!	Outro texto e um terceiro texto
Hello Android!	Outro texto e um terceiro texto

Trabalhando com Layouts – Decompondo em Widgets



Layouts Flexíveis e Otimizados no Jetpack Compose

- Box:
 - Layout flexível para organizar elementos de forma livre.
 - Permite definir margens, preenchimentos, alinhamentos e sobreposição de elementos.



```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Box(
        modifier = Modifier.fillMaxSize()
            .padding(32.dp)
            .background(Color.LightGray)
    ) {
        Text(text = "Texto superior esquerdo",
            modifier = Modifier.align(Alignment.TopStart)
        )
        Text(text = "Texto centralizado",
            modifier = Modifier.align(Alignment.Center))
        Text(text = "Texto inferior direito",
            modifier = Modifier.align(Alignment.BottomEnd))
    }
}
```

GreetingPreview



Layouts Flexíveis e Otimizados no Jetpack Compose

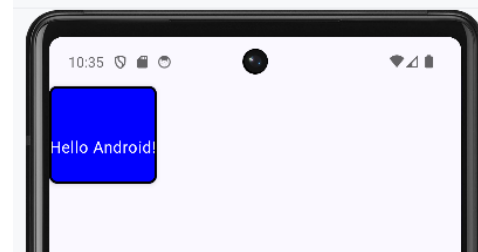
- LazyColumn e LazyRow:
 - Otimizados para listas longas.
 - Renderizam apenas os itens visíveis na tela.
 - `LazyColumn`: lista vertical.
 - `LazyRow`: lista horizontal.
- LazyVerticalGrid:
 - Organiza itens em uma grade verticalmente rolável.
 - Ideal para listas grandes ou de comprimento desconhecido.



Componentes do Jetpack Compose

- Surface:

- Representa uma superfície material com estilos de Material Design.
- Propriedades: elevação, sombra, cor e forma.
- Adaptação automática ao tema do sistema.
- Exemplo: Contêiner estilizado com cor de fundo azul, cantos arredondados, elevação da sombra, cor do conteúdo branca e borda preta.



- Scaffold:

- Estrutura padrão para criação de telas.
- *Disponibiliza 'slots' para barra superior, barra inferior, botão de ação flutuante e corpo principal.*
- Exemplo: Barra superior azul, barra inferior verde, botão de ação flutuante com ícone de adição.



Modificadores de Layout

Modificadores de Layout

- Definição:
 - Objetos que ajustam comportamento, aparência ou layout de composables.
 - Não alteram o estado interno.
- Personalização:
 - Tamanho, preenchimento, alinhamento, cor de fundo, ações de clique.
- Combinação:
 - Imutáveis e podem ser combinados para múltiplos efeitos.
 - Funções:
 - Customizar aparência.
 - Definir comportamento.
 - Melhorar acessibilidade.
 - Gerenciar layout.

Estrutura Básica de um Modificador

- Aplicação:
 - Usado com o parâmetro 'modifier' de um composável.
 - Exemplo: `modifier = modifier.padding(16.dp)`
 - Resultado:
 - Margem de 16 pontos.
 - Alterando para 50.dp, a margem aumenta.

```
Text(  
    text = "Hello $name!",  
    modifier = modifier.padding( 16. dp)  
)
```

GreetingPreview

Hello Android!

```
modifier = modifier.padding( 50. dp)
```

GreetingPreview

Hello Android!

Estrutura Básica de um Modificador

- Aplicação:
 - Usado com o parâmetro 'modifier' de um composable.
 - Exemplo: ``modifier = modifier.padding(16.dp)``
- Modificadores de Layout:
 - `padding()`: Adiciona preenchimento dentro do composable.
 - `fillMaxSize()`: *Faz o composable ocupar todo o espaço disponível.*
 - `wrapContentSize()`: Ajusta o tamanho do composable com base no seu conteúdo.
 - `size()`: Define um tamanho fixo para o composable.
 - `offset()`: Desloca a posição do composable.
 - `align()`: Define o alinhamento dentro do layout pai.
 - `weight()`: Distribui o espaço em ``Row`` ou ``Column`` com base no peso.

Modificadores de Design

Modificadores de Design

- Definição:
 - Determinam como um composable é visualmente renderizado.
 - Definem aspectos como cor de fundo, bordas, sombras e recortes.
- Exemplos:
- `background()`: Adiciona cor de fundo ou forma.
 - Exemplo: `.background(Color.Blue, shape = RoundedCornerShape(16.dp))``
- `border()`: Adiciona uma borda ao redor do composable.
 - Exemplo: `border(4.dp, Color.Black)``
 - `clip()`: Recorta o composable em um formato especificado.
 - Exemplo: `clip(CircleShape)``
 - `alpha()`: Ajusta a transparência do composable.
 - Exemplo: `alpha(0.5f)``
 - `shadow()`: Adiciona um efeito de sombra.
 - Exemplo: `shadow(18.dp)``

Modificadores de Interação

Modificadores de Interação

- Definição:
 - Gerenciam entradas do usuário, como cliques, gestos e rolagens.
 - Definem como o composable responde às ações do usuário.
- Exemplos:
 - `clickable()`: Responde a eventos de clique.
 - `toggleable()`: Adiciona comportamento de alternância.
 - `scrollable()`: Permite que o composable seja rolável.
 - Outros Modificadores:
 - Animação: ``animateContentSize()``, ``graphicsLayer()``.
 - Gestos: ``pointerInput()``, ``nestedScroll()``.
 - Foco: ``focusable()``.

Gerenciamento de Espaço e Alinhamento

Gerenciamento de Espaço e Alinhamento

- Arrangement: Define a distribuição dos componentes no eixo principal (vertical em Column, horizontal em Row) e o alinhamento no eixo perpendicular.
- Parâmetros:
 - ``verticalArrangement`` e ``horizontalArrangement``: Espaçamento no eixo principal.
 - ``horizontalAlignment`` e ``verticalAlignment``: Alinhamento no eixo perpendicular.
- Implementações Comuns:
 - Start: Elementos na margem inicial.
 - End: Elementos na margem final.
 - Center: Elementos centralizados.
 - SpaceBetween: Espaço igual entre elementos.
 - SpaceAround: Espaço extra nas extremidades.
 - SpaceEvenly: Espaço igual entre todos os elementos.

Gerenciamento de Espaço e Alinhamento

GreetingPreview

Arrangement.Start



Arrangement.Center



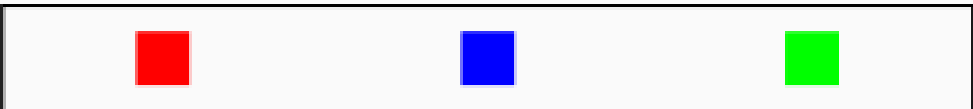
Arrangement.End



Arrangement.SpaceBetween



Arrangement.SpaceAround



Arrangement.SpaceEvenly



Interatividade

Qual modificador faz com que um composable ocupe todo o espaço disponível em seu elemento pai?

- a) `padding()`
- b) `fillMaxSize()`
- c) `wrapContentSize()`
- d) `size()`
- e) `offset()`

Resposta

Qual modificador faz com que um composable ocupe todo o espaço disponível em seu elemento pai?

- a) padding()
- b) fillMaxSize()**
- c) wrapContentSize()
- d) size()
- e) offset()

ATÉ A PRÓXIMA!