

**CURSOS
TÉCNICOS**

**DESENVOLVIMENTO DE
APLICATIVOS I**

Eixo Informática para Internet

UNIDADE 3

SUMÁRIO

3.	INICIANDO E ENTENDENDO UM PROJETO ANDROID	3
3.1	Classe MainActivity	4
3.2	Funções com anotações	5
4.	JETPACK COMPOSE	10
4.1	Trabalhando com o Compose	11
4.2	Mudando o tamanho, altura da linha e alinhamento do texto:.....	14
4.3	Adicionando uma imagem no projeto.....	14
4.4	Adicionando uma imagem combinável - trabalhando com recursos (res/) no projeto	16
5.	A Classe R.....	17
5.1	Adicionando um layout de caixa	18
5.2	Alinhar e organizar o texto usando Column	20
6.	Referências.....	22

UNIDADE 3

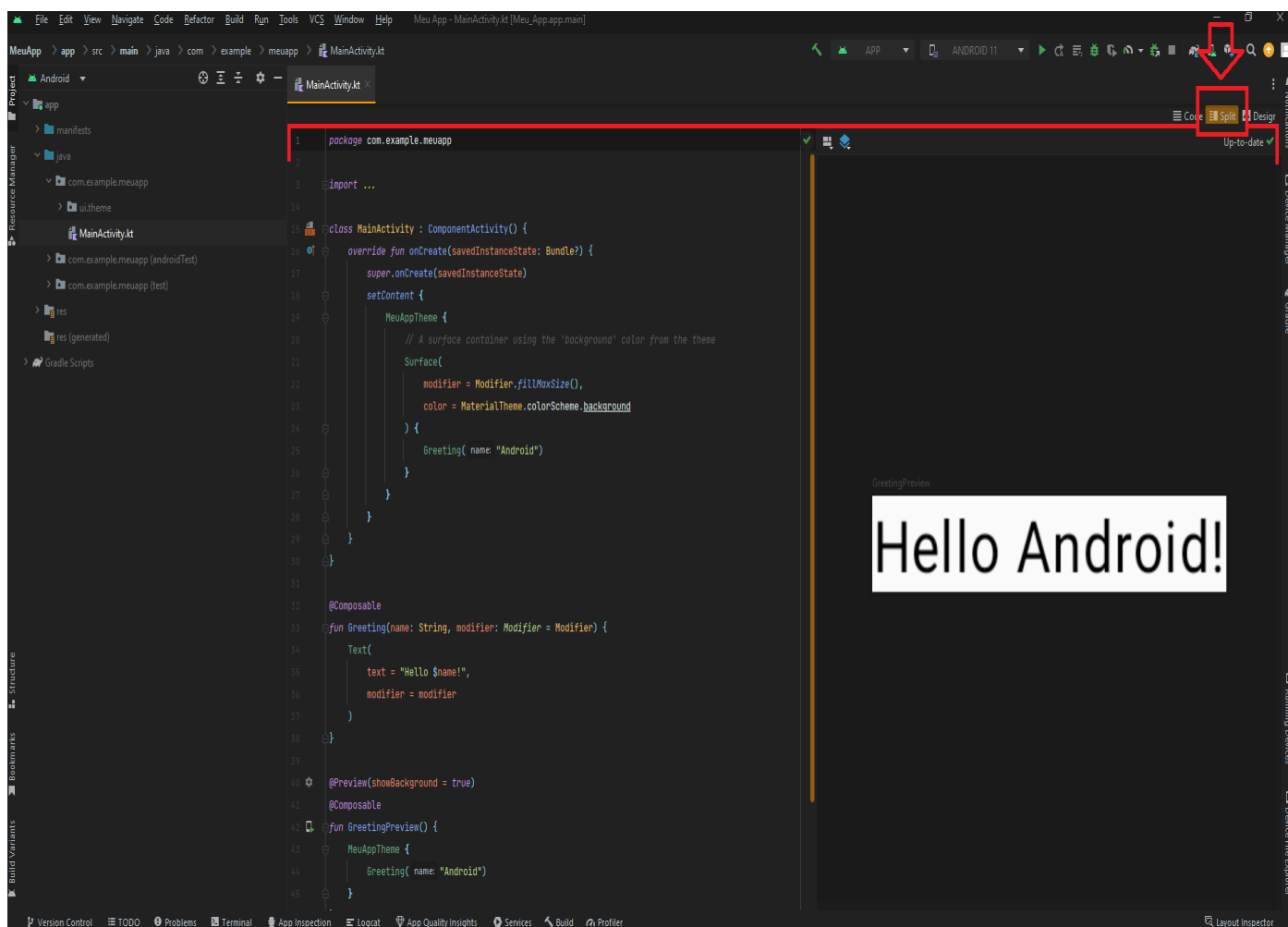
3. INICIANDO E ENTENDENDO UM PROJETO ANDROID

Na unidade anterior, estudamos como iniciar um novo projeto no Android Studio. Tivemos uma visão geral do projeto, com as especificações de alguns tópicos bem como a criação e configuração de máquinas virtuais.

Nesta unidade, iremos nos aprofundar nesta estrutura e desenvolver um primeiro código-base, entendendo cada etapa inicial e fundamental para a continuidade de nossos estudos no desenvolvimento de aplicativos nativos.

Observe a imagem abaixo e deixe seu Android Studio da mesma forma.

- Desta maneira poderemos verificar o que estamos alterando.
- Clique no **botão Split** para que a visualização da aplicação possa ser vista.



3.1 Classe MainActivity

Já tivemos na unidade anterior de estudo, uma introdução à **classe MainActivity**, agora iremos nos familiarizar mais com a estrutura deste arquivo.

Primeiramente, vamos analisar o que é uma Classe: Uma classe no Kotlin (e em qualquer linguagem de programação orientada a objetos) é um conceito fundamental que permite a criação de objetos. **No desenvolvimento Android com Kotlin, uma classe é uma estrutura que define propriedades (variáveis) e métodos (funções) que descrevem o comportamento e as características de um objeto específico.**

Para definir uma classe, usamos a palavra reservada “class” e suas propriedades podem ser listadas em sua declaração ou corpo.

```
class Rectangle(var height: Double, var length: Double)
{ var perimeter = (height + length) * 2 }
```

Quando se trata das padronizações de estilo de codificação do Kotlin no contexto do Android, é de suma importância fazer referência constante à documentação oficial. Nesse sentido, no seguinte link ([Guia de estilo do Kotlin | Android Developers](#)), é possível acessar a definição completa dos padrões de codificação estabelecidos pelo Google para a linguagem Kotlin utilizada no desenvolvimento Android.

A classe MainActivity é o compilador que inicializará a nossa aplicação, pois é o ponto de entrada do aplicativo. **Quando o aplicativo é iniciado, o sistema operacional Android inicia a MainActivity primeiro. A partir daí, você pode configurar a interface do usuário e as ações iniciais do aplicativo.**

Nesta classe, possuímos duas funções muito importantes: onCreate e setContent. A função **onCreate**, invoca as outras funções que serão responsáveis pela criação da interface do usuário (UI), ou seja, estabelece as ações iniciais do aplicativo. Portanto, é um dos métodos mais importantes e fundamentais da classe MainActivity em um aplicativo Android

desenvolvido usando Kotlin. **Ele é chamado quando a atividade é criada e é onde você configura grande parte da inicialização do seu aplicativo.**

A função `setContent` é usada na classe `MainActivity` do Android para definir o layout da interface do usuário que será exibido na tela quando a atividade for iniciada, ou seja, configura a visualização da atividade. Essa função é usada no método `onCreate` da `MainActivity` para associar um arquivo de layout XML (que define a estrutura visual de uma tela ou componente) ao conteúdo da atividade.

```

15 class MainActivity : ComponentActivity() {
16     override fun onCreate(savedInstanceState: Bundle?) {
17         super.onCreate(savedInstanceState)
18         setContent {
19             MeuAppTheme {
20                 // A surface container using the 'background' color from the theme
21                 Surface(
22                     modifier = Modifier.fillMaxSize(),
23                     color = MaterialTheme.colorScheme.background
24                 ) {
25                     Greeting( name: "Android")
26                 }
27             }
28         }
29     }
30 }

```

Fonte da imagem: de autoria própria

3.2 Funções com anotações

No contexto do Kotlin para o desenvolvimento Android, as anotações são metadados especiais que podem ser adicionados ao código-fonte para fornecer informações adicionais ao compilador, à máquina virtual Java ou a outras ferramentas que processam o código. As anotações são precedidas pelo símbolo "@".

As anotações podem ser aplicadas a várias entidades no Kotlin, incluindo classes, propriedades, funções e parâmetros de função. **No contexto específico do Android, as anotações são frequentemente usadas para configurar comportamentos específicos do sistema Android, otimizar código ou fornecer informações para ferramentas de desenvolvimento.**

No arquivo MainActivity encontramos algumas funções com anotações que são extremamente importantes na execução e build (construção) da aplicação e estas funções podem ser chamadas a partir da função setContent.

3.3 @Composable

A anotação @Composable é uma parte central do toolkit de UI declarativa do Jetpack Compose, uma biblioteca moderna para construir interfaces do usuário no Android usando a linguagem de programação Kotlin.

Com o Jetpack Compose, podemos construir a interface do usuário de maneira declarativa, o que significa que a descrição de como a interface deve ser exibida em um determinado estado, em vez de manipular diretamente as visualizações individuais.

Ao usar a anotação **@Composable em Kotlin**, podemos sinalizar que estamos definindo uma função que descreve uma parte da interface do usuário. **Essa função é chamada de função composável (função combinável), e ela pode receber parâmetros que representam o estado e outras informações da interface do usuário.**

Sempre que o estado ou as entradas mudam, o Compose se encarrega de atualizar automaticamente a interface do usuário com base nas novas informações.

Observe o exemplo abaixo: A função Greeting (saudação) recebe um nome e exibe **“Hello \$name!”**.

```

32  @Composable
33  fun Greeting(name: String, modifier: Modifier = Modifier) {
34      Text(
35          text = "Hello $name!",
36          modifier = modifier
37      )
38  }

```

Fonte da imagem: de autoria própria

É possível fazer a alteração deste valor e ser exibida na interface da aplicação.

```

32  @Composable
33  fun Greeting(name: String, modifier: Modifier = Modifier) {
34      Text(
35          text = "Olá, meu nome é $name!",
36          modifier = modifier
37      )
38  }

```

GreetingPreview

Olá, meu nome é Android!

Fonte da imagem: de autoria própria

Como o nosso nome não é Android, precisamos fazer algumas modificações no nosso código. Desta forma iremos conhecer agora funções que são específicas e voltadas para a visualização do aplicativo. A função `@Preview` e `Default Preview`.

3.3.1 DefaultPreview

É usada no Jetpack Compose, a biblioteca de UI declarativa do Android, para definir uma visualização padrão (preview) para um composable específico.

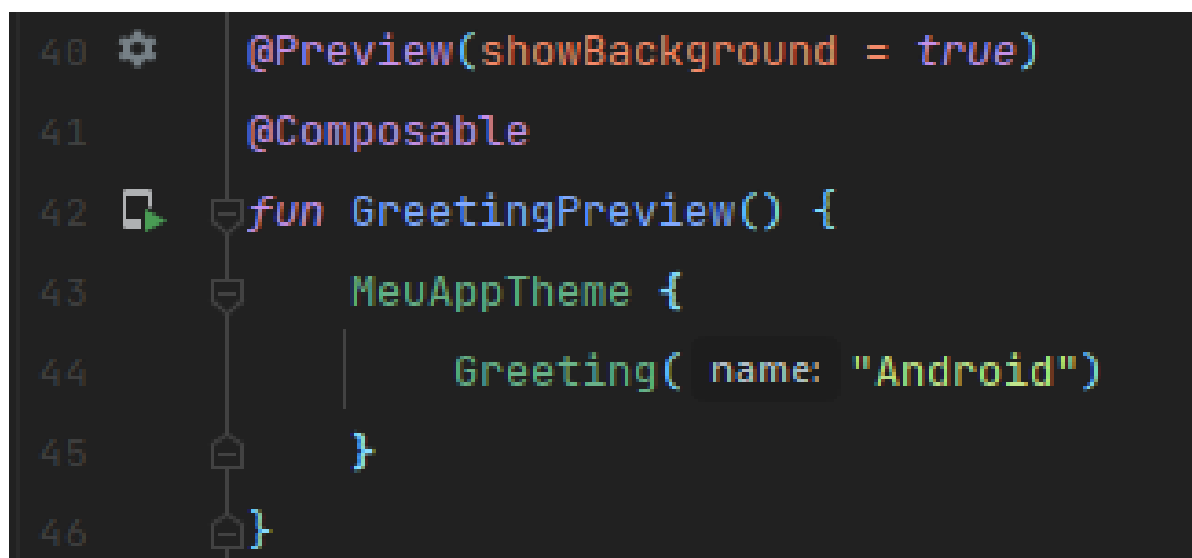
Essa anotação é usada para criar visualizações que permitem aos desenvolvedores ver como o composable será renderizado no Android Studio, durante o desenvolvimento, sem ter que executar o aplicativo no dispositivo ou emulador.

E por ser uma função de visualização precisamos adicionar uma notação de visualização → **@Preview**.

3.3.2 @Preview

A anotação **@Preview** é usada no **Jetpack Compose**, a biblioteca de UI declarativa do **Android**, para criar visualizações de elementos que podem ser compostos (componentes UI) diretamente no Android Studio. Isso permite que os desenvolvedores visualizem como os composables serão renderizados sem precisar executar o aplicativo completo no dispositivo ou emulador.

É uma ferramenta muito útil para iterar rapidamente sobre o design da interface do usuário. A imagem abaixo mostra que a notação de Preview recebe um parâmetro chamado `showBackground` e está definido como `true` (verdadeiro), desta forma, este parâmetro adiciona um plano de fundo no aplicativo.



```

40  @Preview(showBackground = true)
41  @Composable
42  fun GreetingPreview() {
43      MeuAppTheme {
44          Greeting( name: "Android")
45      }
46  }
    
```

Fonte da imagem: de autoria própria

Para que possamos deixar mais personalizado a visualização do nosso aplicativo vamos fazer as seguintes alterações no código:


```

40  @Preview(showBackground = true)
41  @Composable
42  fun GreetingPreview() {
43      MeuAppTheme {
44          Greeting( name: "Cristina")
45      }
46  }

```

Na chamada da função **Greeting** (linha 44 da imagem ao lado), mudamos o valor do name para um nome mais personalizado.

Na linha 36 do nosso código vamos envolver a propriedade Text com um container chamado Surface (superfície) e vamos escolher outra cor de fundo para a nossa aplicação. Neste caso, foi colocado Magenta.

```

34  @Composable
35  fun Greeting(name: String, modifier: Modifier = Modifier) {
36      Surface(color = Color.Magenta) {
37          Text(
38              text = "Olá, meu nome é $name!",
39              modifier = modifier
40          )
41      }
42  }

```




Vamos adicionar um espaço ao redor do texto. Estes recursos são importantes pois eles facilitam a vida do usuário.

```

36  @Composable
37  fun Greeting(name: String, modifier: Modifier = Modifier) {
38      Surface(color = Color.Magenta) {
39          Text(
40              text = "Olá, meu nome é $name!",
41              modifier = Modifier.padding(24.dp)
42          )
43      }
44  }

```




Fonte da imagem: de autoria própria

4. JETPACK COMPOSE

A interface do usuário (UI) é crucial não apenas para melhorar a experiência do usuário, mas também para garantir que todos possam navegar facilmente em um aplicativo. Se a interface não for clara, os usuários não conseguirão usar o aplicativo adequadamente, o que frequentemente afeta o sucesso do deste. Portanto, uma UI bem projetada não apenas torna as funções do aplicativo acessíveis, mas também desempenha um papel importante no êxito global da aplicação.

O **Jetpack Compose** é um **framework** moderno de criação de interfaces de usuário (UI) para o desenvolvimento de aplicativos Android altamente usado no mercado de trabalho, é oficial do Android e está integrado ao Android Studio. Ao desenvolver a interface usando Compose, você pode referenciar os recursos presentes na pasta "res" através da classe R. Por exemplo, ao utilizar uma imagem na interface, você pode acessar o ID da imagem na classe R para exibi-la usando um composável. Ele foi desenvolvido pela Google com o objetivo de simplificar e agilizar a criação de interfaces de usuário de maneira declarativa e reativa. Em vez de utilizar a abordagem tradicional de XML e código Java/Kotlin para criar a interface do usuário, o **Jetpack Compose permite que os desenvolvedores construam interfaces de forma mais intuitiva e eficiente, usando Kotlin como linguagem principal.**

Acompanhe algumas características:

- ✚ **Declarativo:** em vez de descrever a interface do usuário passo a passo em código procedural, o Compose permite que os desenvolvedores declarem como a interface deve ser exibida de maneira mais direta e declarativa. Isso facilita a compreensão e a manutenção do código.
- ✚ **Reativo:** o Compose utiliza um modelo reativo, o que significa que as interfaces de usuário são atualizadas automaticamente em resposta a alterações nos dados subjacentes. Isso elimina a necessidade de gerenciar manualmente as atualizações de interface.
- ✚ **Composição hierárquica:** o Compose permite a criação de componentes de interface menores e reutilizáveis, que podem ser combinados para formar interfaces mais complexas. Isso facilita a organização do código e a criação de interfaces consistentes.



Pré-visualização em tempo real: uma característica muito útil do Jetpack Compose é a capacidade de visualizar em tempo real como a interface está sendo construída enquanto você escreve o código. Isso agiliza o processo de desenvolvimento e ajuda a identificar problemas rapidamente.



Total integração com o ecossistema Android: o Jetpack Compose é projetado para funcionar perfeitamente com os componentes e bibliotecas existentes do Android, permitindo aos desenvolvedores combinar o Compose com outros recursos do ecossistema Android.



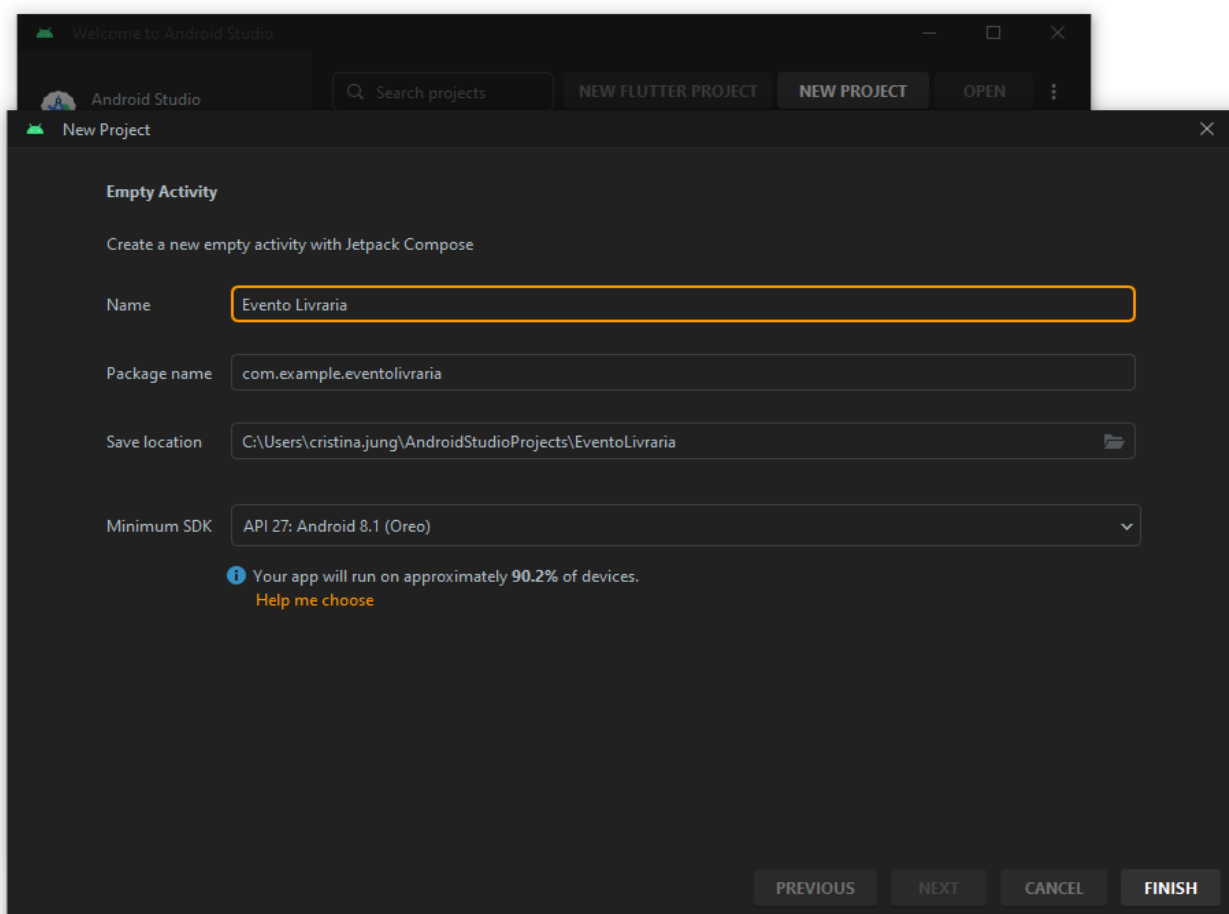
Sintaxe Kotlin: o Jetpack Compose é totalmente escrito em Kotlin e faz uso extensivo das características da linguagem, como extensões e funções de ordem superior, para simplificar a criação de interfaces.

4.1 Trabalhando com o Compose

Vamos começar um novo projeto em que precisamos criar um convite para a inauguração de uma livraria. Neste projeto iremos trabalhar com os seguintes conceitos:

- Criação de uma interface simples usando elementos de texto de composição;
- Como escrever funções combináveis, como Text, Column e Row;
- Classe R;
- Trabalhar com imagens em funções combináveis;
- Exibir o conteúdo no app em um layout.

Na imagem abaixo, encontramos o nome da nossa aplicação.



Fonte da imagem: de autoria própria

Após abrir o projeto, vamos esperar carregar todos os serviços e configurar a exibição da aplicação em modo Split. Clique no link de Refres e Building para que o Android Studio possa mostrar a visualização da aplicação.

Na próxima etapa, vamos renomear a função que possui a notação `@Composable` conforme a próxima imagem (`LibraryEventPreview`), sempre usamos o padrão CamelCase para nomes de funções.

```

40  @Preview(showBackground = true)
41  @Composable
42  fun LibraryEventPreview() {
43      EventoLivrariaTheme {
44          Greeting( name: "The Book House")
45      }
46  }
  
```

A função combinável precisa fornecer valores padrão para todos os parâmetros para que a prévia possa ser mostrada. **Por esse motivo, uma prévia direta da função Greeting() não é recomendada.** Em vez disso, é necessário adicionar outra função, a **LibraryEventPreview()**, que **chama a Greeting() com um parâmetro adequado** → **neste caso, já colocamos o nome da livraria.** Portanto, seu propósito é definir uma visualização padrão para um composable específico.

E nosso código deverá ficar como a próxima imagem:

```

1  package com.example.eventolivraria
2
3  import ...
4
14
15  class MainActivity : AppCompatActivity() {
16      override fun onCreate(savedInstanceState: Bundle?) {
17          super.onCreate(savedInstanceState)
18          setContent {
19              EventoLivrariaTheme {
20                  // A surface container using the 'background' color from the theme
21                  Surface(
22                      modifier = Modifier.fillMaxSize(),
23                      color = MaterialTheme.colorScheme.background
24                  ) {
25
26                  }
27              }
28          }
29      }
30  }
31  @Composable
32  fun GreetingText(message: String, modifier: Modifier = Modifier) {
33      Text(
34          text = message
35      )
36  }
37  @Preview(showBackground = true)
38  @Composable
39  fun LibraryEventPreview() {
40      EventoLivrariaTheme {
41          GreetingText( message = " Seja Bem Vindo a The Book House")
42      }
43  }

```

Fonte da imagem: de autoria própria

4.2 Mudando o tamanho, altura da linha e alinhamento do texto:

```

33  @Composable
34  fun GreetingText(message: String, modifier: Modifier = Modifier) {
35      Text(
36          text = message,
37          fontSize = 100.sp,
38          lineHeight = 116.sp,
39          textAlign = TextAlign.Center
40      )
41  }
42  @Preview(showBackground = true)
43  @Composable
44  fun LibraryEventPreview() {
45      EventOlivariaTheme {
46          GreetingText( message = " Seja Bem Vindo a The Book House")
47      }
48  }

```

É importante observarmos algumas propriedades de estilização e seus valores. Especificamente, nos casos de `fontSize` e `lineHeight`, estamos usando `sp` e quando declararmos esta unidade, o Android Studio irá destacar.

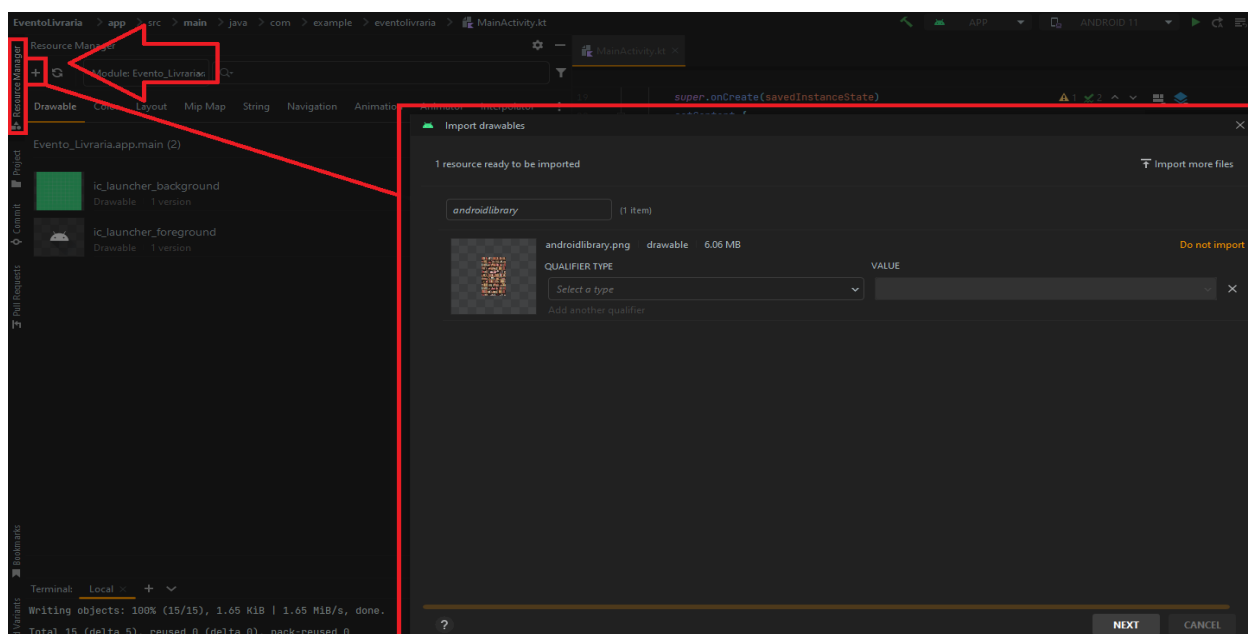
Para solucionar este problema, podemos pressionar as teclas **ALT ENTER** para que a extensão seja importada. Ela irá aparecer no local superior do arquivo, a parte dos imports:

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

4.3 Adicionando uma imagem no projeto

Para inserir uma imagem no projeto, precisamos configurá-lo para receber este recurso. Abra a imagem usada neste app usando este [link](#) (clique em download), ou procure uma mais ao seu gosto.

No Android Studio, clique em **View > Tool Windows > Resource Manager** ou na guia **Resource Manager** ao lado da janela **Project**.



Fonte da imagem: de autoria própria

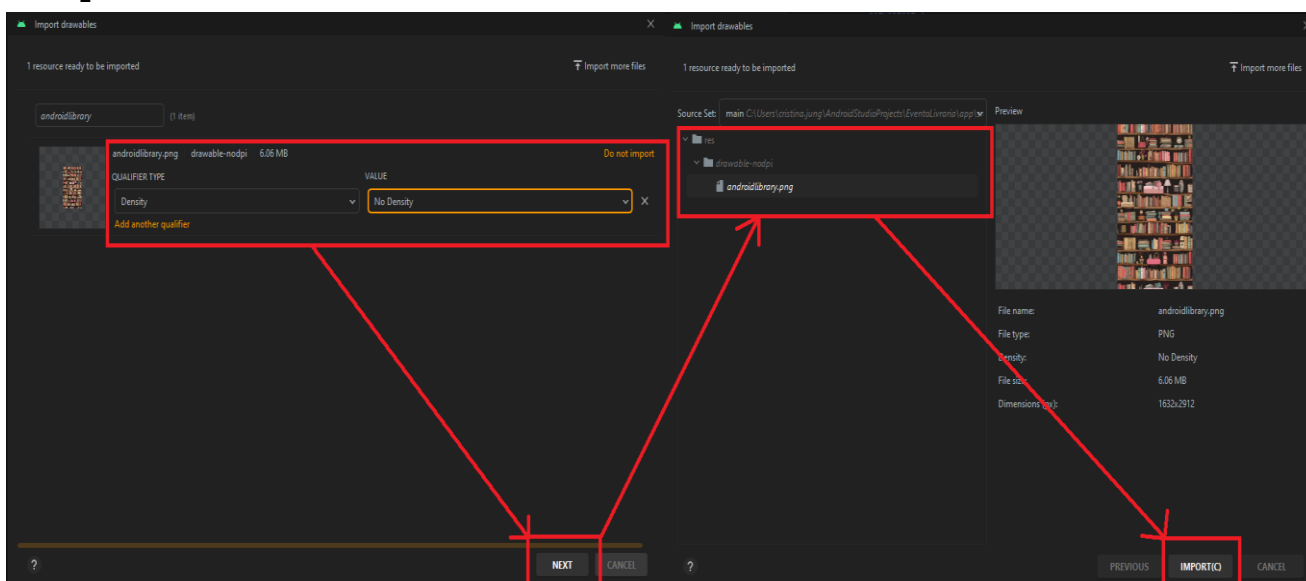
DICA IMPORTANTE!

Os dispositivos Android têm telas de tamanhos variados (por exemplo, smartphones, tablets, TVs etc.), e elas também têm diferentes tamanhos de pixel. Ou seja, enquanto um dispositivo tem 160 pixels por polegada quadrada, outro encaixa 480 pixels no mesmo espaço.

Caso não considerarmos essas variações em densidade de pixel, o sistema pode dimensionar as imagens, resultando em imagens desfocadas, imagens grandes que consomem muita memória ou imagens com tamanho incorreto.

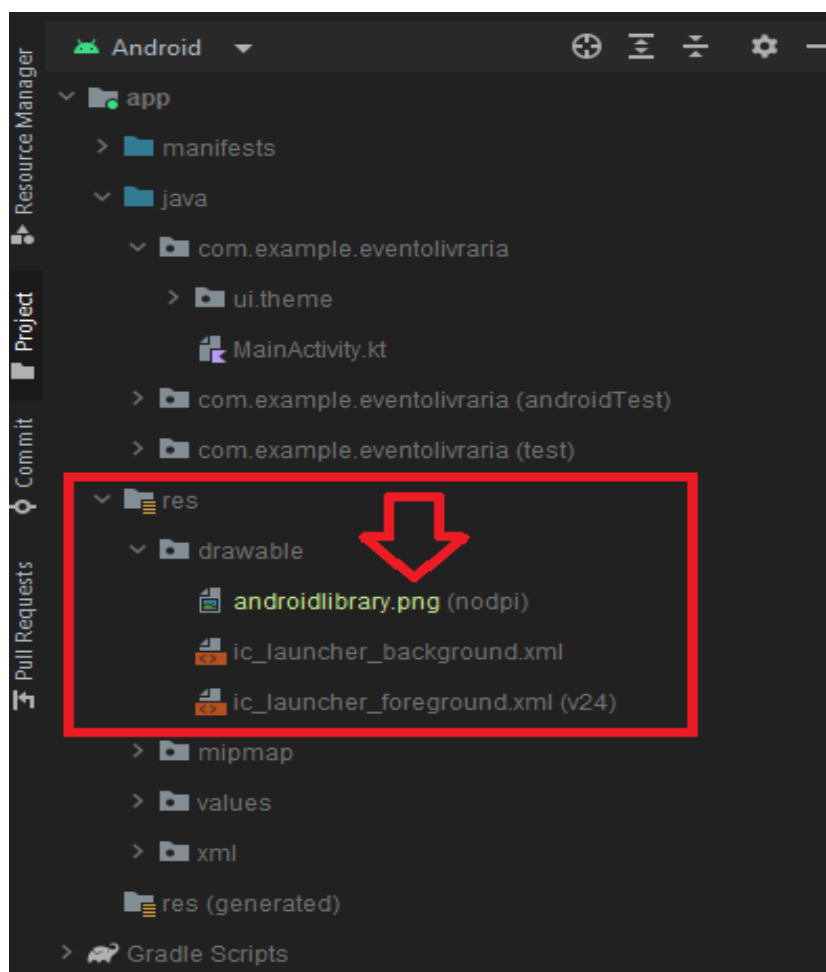
Quando redimensionamos imagens que ultrapassam o tamanho que o sistema Android consegue processar, é gerado um erro de falta de memória.

As fotografias e imagens de plano de fundo, como a atual (androidlibrary.png), precisam ser colocadas na pasta **drawable-nodpi**, o que interrompe o comportamento de redimensionamento.



Fonte da imagem: de autoria própria

Iremos encontrar a imagem neste local dentro do projeto:

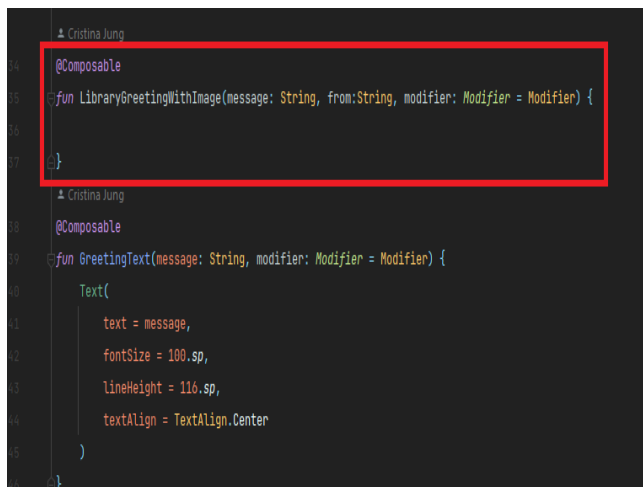


Fonte da imagem: de autoria própria

4.4 Adicionando uma imagem combinável - trabalhando com recursos (res/) no projeto

Para que uma imagem seja mostrada no app, é necessário informar o local de exibição. Do mesmo modo como usamos um elemento Text que pode ser composto para mostrar texto, é possível usar um **elemento Image** para mostrar uma imagem. Nesta etapa, iremos adicionar um elemento de composição Image ao app e vamos usar a imagem de fundo de livros que foi transferida por download, posicionar essa imagem e ajustar o tamanho dela para preencher a tela.

Vamos acessar novamente o arquivo **MainActivity** e iremos adicionar a função **LibraryGreetingWithImage(): um chamado message para a mensagem de convite e outro from**



```

34 @Composable
35 fun LibraryGreetingWithImage(message: String, from: String, modifier: Modifier = Modifier) {
36
37 }
38
39 @Composable
40 fun GreetingText(message: String, modifier: Modifier = Modifier) {
41     Text(
42         text = message,
43         fontSize = 100.sp,
44         lineHeight = 116.sp,
45         textAlign = TextAlign.Center
46     )
47 }
  
```

Passamos dois parâmetros String à função LibraryGreetingWithImage(): um chamado **message** para a mensagem do convite e outro chamado **from** para o nome de quem está convidando, como por exemplo, mensagem de saudação no início do aplicativo.

Todas as funções combináveis precisam aceitar um parâmetro Modifier opcional. Os modificadores informam para um elemento da UI como serão dispostos, exibidos ou se comportaram no layout pai.

É importante saber que estamos inserindo recursos no aplicativo, como no caso da imagem. Portanto recursos são os arquivos complementares e o conteúdo estático usado pelo seu código, como bitmaps, definições de layout, strings da interface do usuário, instruções de animação, entre outros.

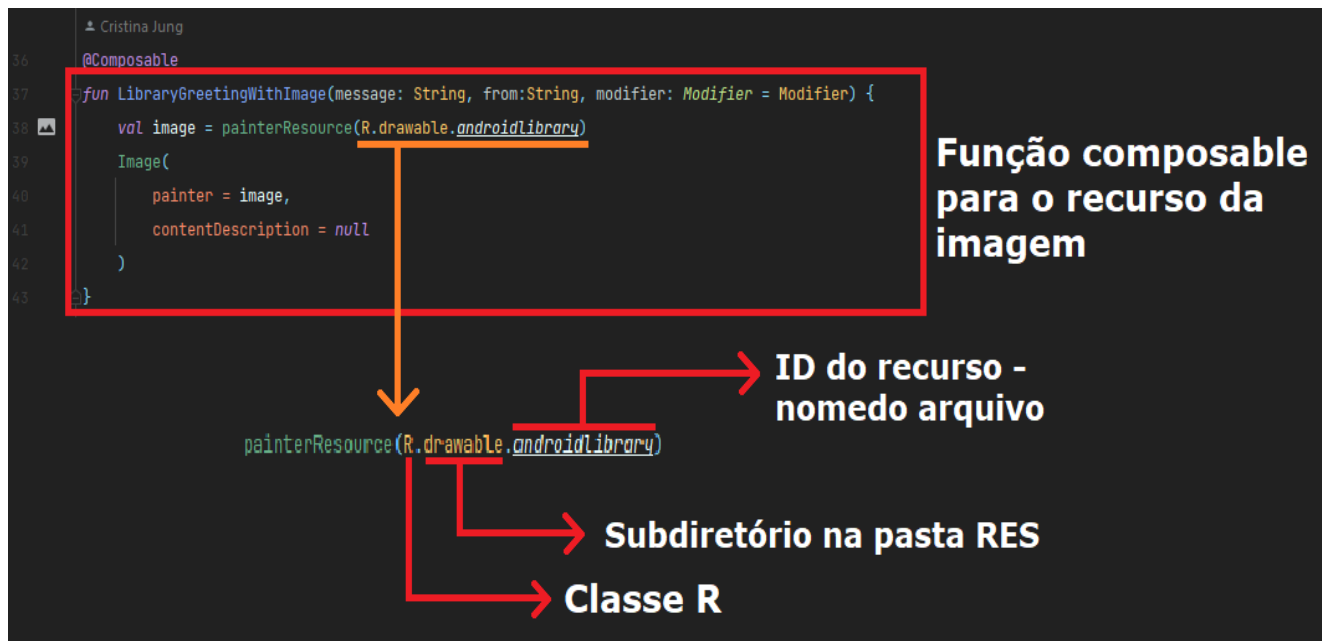
Existem vários tipos de recursos que possuem suporte dentro do diretório res/ no projeto do Android Studio, é possível ter acesso a todos eles neste link → [\(Visão geral dos recursos de aplicativo | Desenvolvedores Android\)](#)

5. A Classe R

O Jetpack Compose pode acessar os recursos definidos no seu projeto Android. O acesso pode ser feito usando IDs de recurso que são gerados na Classe R do projeto.

A classe R é gerada automaticamente pelo Android e contém os IDs (identificadores) de todos os recursos no projeto, imagens, strings e outros recursos que serão adicionados à pasta "res" do seu projeto, que contém todos os recursos do seu projeto, organizados em subpastas como "layout", "drawable" e "values". Ela é essencial para separar os diferentes tipos de recursos utilizados no aplicativo. Os IDs são utilizados para acessar os recursos de maneira eficiente durante a execução do aplicativo.

Na maioria dos casos, o ID do recurso é igual ao nome do arquivo.



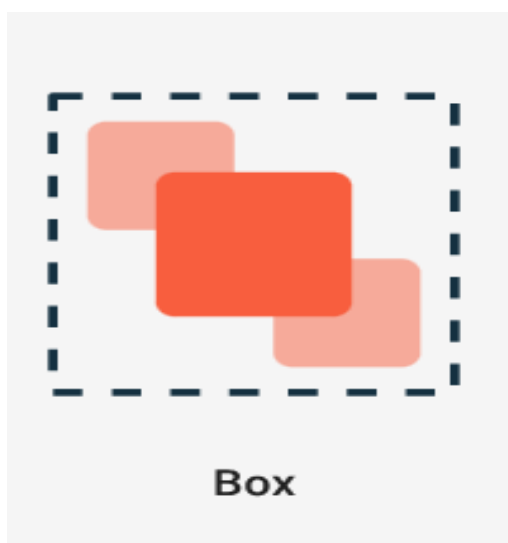
Fonte da imagem: de autoria própria

5.1 Adicionando um layout de caixa

Os três elementos básicos de layout padrão do Compose são **Column**, **Row** e **Box**. Vamos conhecer melhor o elemento combinável Box.

O layout Box é um dos elementos de layout padrão do Compose. Usamos o layout Box para empilhar elementos uns sobre os outros.

O layout Box também permite a configuração do alinhamento específico dos elementos que ele contém.



Fonte da imagem: <[Desenvolvedores Android](#)>

Na função **LibraryGreetingWithImage()**, adicione um elemento Box ao redor do elemento Image, conforme mostrado abaixo:

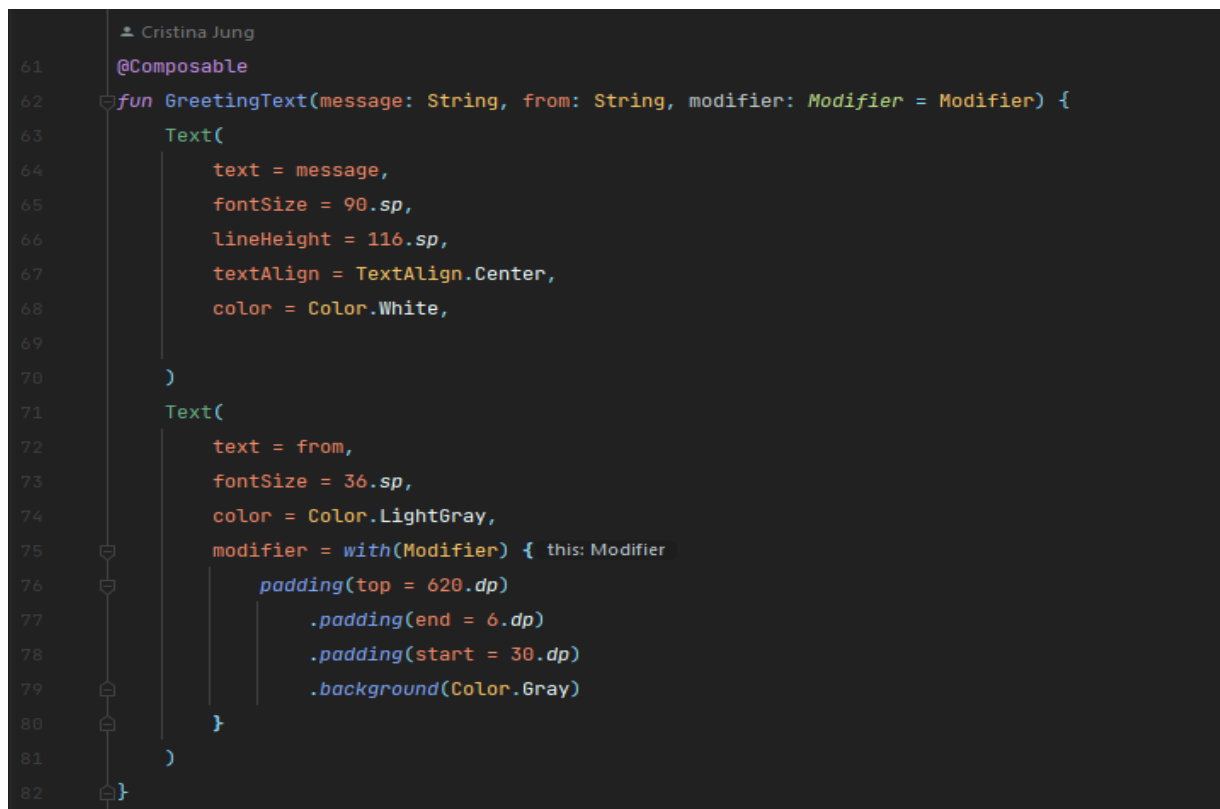


```

37 @Composable
38 fun LibraryGreetingWithImage(message: String, from: String, modifier: Modifier = Modifier) {
39     val image = painterResource(R.drawable.androidlibrary)
40     Box { this: BoxScope
41         Image(
42             painter = image,
43             contentDescription = null
44         )
45     }
46 }
    
```

Fonte da imagem: de autoria própria

Para deixar este elemento mais relacionado com a interface do usuário, podemos, ao final do elemento combinável Box, chamamos a função **GreetingText()** e passamos a ela a mensagem do convite, a assinatura e o modificador, conforme mostrado na imagem abaixo:



```

61 @Composable
62 fun GreetingText(message: String, from: String, modifier: Modifier = Modifier) {
63     Text(
64         text = message,
65         fontSize = 90.sp,
66         lineHeight = 116.sp,
67         textAlign = TextAlign.Center,
68         color = Color.White,
69     )
70
71     Text(
72         text = from,
73         fontSize = 36.sp,
74         color = Color.LightGray,
75         modifier = with(modifier) { this: Modifier
76             padding(top = 620.dp)
77             .padding(end = 6.dp)
78             .padding(start = 30.dp)
79             .background(Color.Gray)
80         }
81     )
82 }
    
```

Fonte da imagem: de autoria própria

5.2 Alinhar e organizar o texto usando Column

No Android Jetpack Compose, não usamos mais o XML para definir a interface do usuário, mas sim escreve código Kotlin para criar a UI de maneira mais declarativa.

Em vez de uma "Column", no Android Jetpack Compose, você usaria o composable Column para criar uma coluna vertical de elementos em sua interface do usuário. Aqui está um exemplo de sintaxe básica de Column no Android Compose:

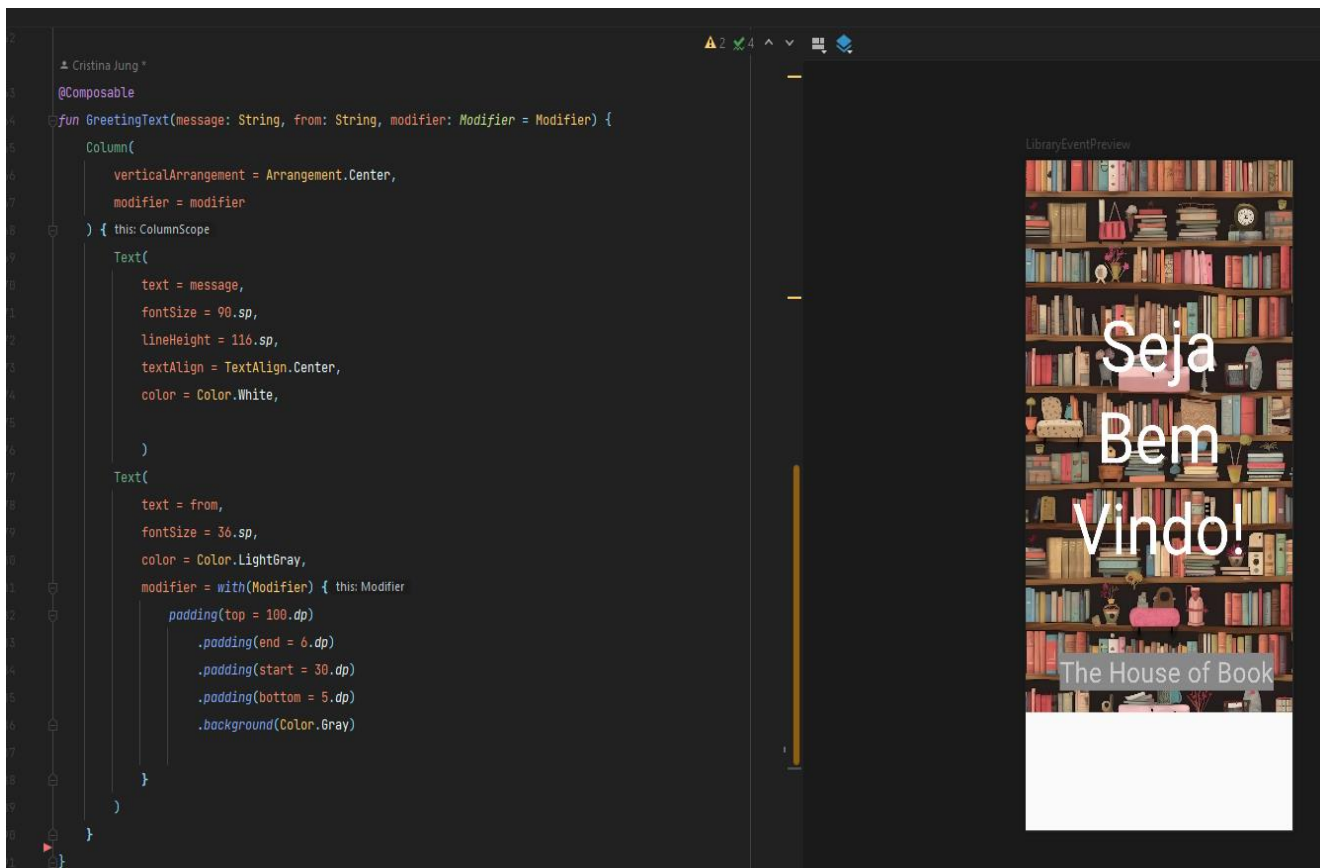
```
new *
104 @Composable
105 fun MyComposable() {
106     Column { this: ColumnScope
107         // Elemento 1
108         // Elemento 2
109         // Outros elementos
110     }
111 }
```

Fonte da imagem: de autoria própria

No exemplo acima, o Column agrupa os composable elements dentro dele **verticalmente**. Podemos adicionar vários elementos dentro do Column para que eles sejam dispostos verticalmente um abaixo do outro.

É importante observar que o Android Jetpack Compose é uma nova abordagem para criar interfaces de usuário no Android e oferece uma sintaxe e uma experiência de desenvolvimento diferentes em comparação com as abordagens tradicionais baseadas em XML.

O Compose é voltado para a criação de interfaces mais dinâmicas e reativas usando o Kotlin. Certifique-se de estar usando a versão adequada do Android Studio e os plugins necessários para desenvolver com o Android Jetpack Compose.



Fonte da imagem: de autoria própria

Para conferência e correção do código → [clique neste link para acessá-lo.](#)

6. Referências

Múltiplos autores: **KOTLIN DOCS**. 2023. Disponível em: <[Kotlin Docs | Kotlin Documentation](#)>. Acesso em: 17 de julho de 2023

Múltiplos autores: **DOCUMENTAÇÃO ANDROID STUDIO**. 2023. Disponível em: <[Download Android Studio & App Tools](#)>. Acesso em: 18 de julho de 2023

Gradle DOCS. 2023. Disponível em: <[Gradle](#)>. Acesso em 20 de julho de 2023.

Múltiplos autores. **ACTIVITY ANDROID - DOCUMENTAÇÃO ANDROID ANDROID STUDIO**. Disponível em: <[Activity | Android Developers](#)> Acesso em 20 de julho de 2023.

Múltiplos autores. **CLASSES E OBJETOS - DOCUMENTAÇÃO ANDROID STUDIO**. Disponível em: <[Usar classes e objetos no Kotlin](#)> Acesso em 05 de agosto de 2023.

Múltiplos autores. **COMPOSABLE - DOCUMENTAÇÃO ANDROID STUDIO**. Disponível em: <[Composable | Android Developers](#)>. Acesso em 06 de agosto de 2023.

Múltiplos autores. **CONCEITOS BÁSICOS DE LAYOUT DO COMPOSE -DOCUMENTAÇÃO ANDROID STUDIO**. Disponível em: <[Conceitos básicos de layout do Compose | Jetpack Compose | Android Developers](#)>. Acesso em 06 de agosto de 2023