

Desenvolvimento de Aplicativos Móveis

Professor Maurício Buess

mbuess@up.edu.br

<https://github.com/mauriciobuess>

Introdução ao Desenvolvimento Android:

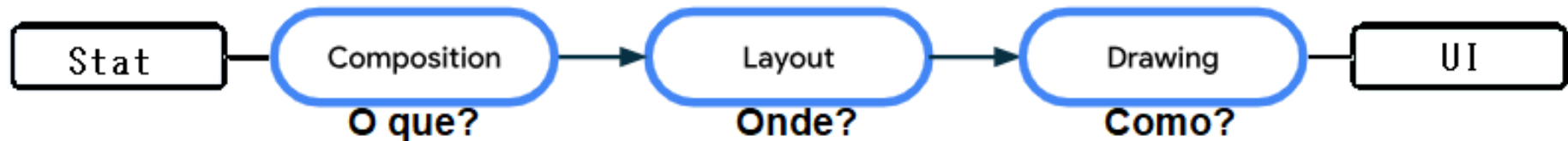
- Compreender o funcionamento do framework Jetpack Compose
- Compreender o funcionamento dos elementos básicos do framework

Jetpack Compose

- Kit de ferramentas para criação de *interface* nativa no Android.
- Simplifica e acelera o desenvolvimento de *interfaces* no Android com menos código.
- A interface do usuário (UI) de um app é o que você tem na tela.
- Funções combináveis são o elemento básico fundamental do Compose. E ela é uma função que descreve alguma parte da interface.
- A função combinável recebe a anotação `@Composable`. Essa anotação informa ao compilador do Compose que essa função se destina a converter dados em interface.

Jetpack Compose

- Os três elementos básicos de layout padrão do Compose são **Column**, **Row**, e **Box**. Essas são funções combináveis, ou seja, podemos colocar itens nelas.
- O Compose transforma o estado em elementos da IU usando:
 - Composição de elementos
 - Layout dos elementos
 - Desenho de elementos



Column, Row e Box

- Possuem o parâmetro Modifier
 - Parâmetro que permite modificar o comportamento e a aparência dos componentes no Jetpack Compose.
 - Ele é utilizado para aplicar espaçamentos, tamanhos, cores, alinhamentos, gestos e outras propriedades aos componentes.
- **Modifier** - é uma classe que permite a aplicação de modificações e estilos aos componentes de UI. Ele é imutável e usado de forma funcional, o que significa que você pode combinar vários modificadores aplicando-os em sequência.

Funções combináveis

- São o elemento básico fundamental do Compose;
- Uma função `@Composable` é uma função que emite uma Unit de descrição de alguma parte da IU;
- A função recebe alguma entrada e gera o que será exibido na tela;
- Pode emitir vários elementos da interface;
- Deve-se fornecer orientações sobre como os objetos devem ser organizados, caso contrário o Compose colocará os elementos de texto um sobre o outro;

Elementos de layout padrão Compose

- Use **Column** para colocar itens na tela verticalmente.

```
@Composable
fun ArtistCardColumn() {
    Column {
        Text("Alfred Sisley")
        Text("3 minutes ago")
    }
}
```

Alfred Sisley
3 minutes ago

Elementos de layout padrão Compose

- Use **Row** para colocar itens na tela verticalmente.

@Composable

```
fun ArtistCardRow(artist: Artist) {  
    Row(verticalAlignment = Alignment.CenterVertically) {  
        Image(bitmap = artist.image, contentDescription = "Artist image")  
        Column {  
            Text(artist.name)  
            Text(artist.lastSeenOnline)  
        }  
    }  
}
```



Alfred Sisley

3 minutes ago

Elementos de layout padrão Compose

- Use **Box** para colocar elementos uns sobre os outros.

@Composable

```
fun ArtistAvatar(artist: Artist) {  
    Box {  
        Image(bitmap = artist.image, contentDescription = "Artist image")  
        Icon(Icons.Filled.Check, contentDescription = "Check mark")  
    }  
}
```



Elementos de layout padrão Compose



Column



Row



Box

- o Compose processa layouts aninhados de forma eficiente, o que faz deles uma ótima maneira de projetar uma IU complexa.
- Evite layouts aninhados por motivos de desempenho.

Elementos de layout padrão Compose

- Para definir a posição dos filhos em uma **Row**, defina os argumentos **horizontalArrangement** e **verticalAlignment**.
- Para uma **Column**, defina os argumentos **verticalArrangement** e **horizontalAlignment**:

@Composable

```
fun ArtistCardArrangement(artist: Artist) {  
    Row(  
        verticalAlignment = Alignment.CenterVertically,  
        horizontalArrangement = Arrangement.End  
    ) {  
        Image(bitmap = artist.image, contentDescription = "Artist image")  
        Column { /*...*/ }  
    }  
}
```



Alfred Sisley

3 minutes ago

Usando modificadores

- A classe Modifier no Jetpack Compose fornece uma variedade de parâmetros (ou recursos) que podem ser usados para modificar e estilizar componentes de UI. Tais recursos permitem a realização de ajustes de aspectos como o layout, a aparência e o comportamento dos componentes.
- Modifier.padding - Adiciona espaçamento interno ao redor do conteúdo do componente.

Modifier.padding(16.dp)

Modifier.padding(start = 8.dp, end = 8.dp)

Usando modificadores

- Modifier.background - Define a cor ou o desenho de fundo do componente.

```
Modifier.background(Color.Blue)
```

```
Modifier.background(Brush.linearGradient(listOf(Color.Red, Color.Blue)))
```

- Modifier.size - Define o tamanho do componente..

```
Modifier.size(100.dp)
```

```
Modifier.size(width = 100.dp, height = 50.dp)
```

Usando modificadores

- `Modifier.fillMaxWidth` e `fillMaxHeight` - Faz com que o componente ocupe toda a largura ou altura disponível.

`Modifier.fillMaxWidth()`

`Modifier.fillMaxHeight()`

- `Modifier.width` e `height` - Define a largura e a altura específicas do componente.

`Modifier.width(100.dp)`

`Modifier.height(50.dp)`

Usando modificadores

- `Modifier.border` - Adiciona uma borda ao redor do componente.

```
Modifier.border(2.dp, Color.Black)
```

```
Modifier.border(2.dp, Brush.linearGradient(listOf(Color.Red, Color.Green)))
```

- `Modifier.align` - Define o alinhamento do componente dentro de um contêiner pai.

```
Modifier.align(Alignment.Center)
```

```
Modifier.align(Alignment.BottomEnd)
```

Usando modificadores

- Usados para decorar ou aumentar seus elementos;
- Os modificadores são essenciais para personalizar seu layout.

@Composable

```
fun ArtistCardModifiers(artist: Artist, onClick: () -> Unit) {  
    val padding = 16.dp  
    Column(Modifier.clickable(onClick = onClick)  
        .padding(padding)  
        .fillMaxWidth() ) {  
        Row(verticalAlignment = Alignment.CenterVertically) { /*...*/ }  
        Spacer(Modifier.size(padding))  
        Card(elevation=CardDefaults.cardElevation(defaultElevation=4.dp),  
            ) { /*...*/ }  
    }  
}
```




Alfred Sisley

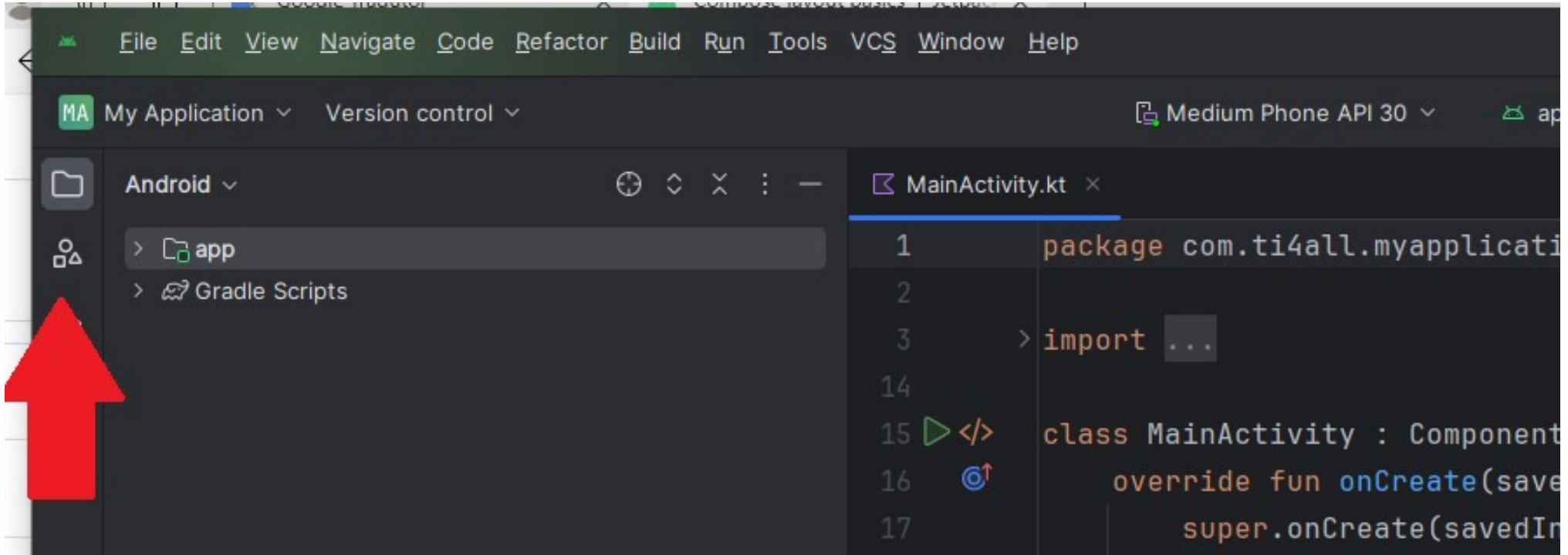
3 minutes ago



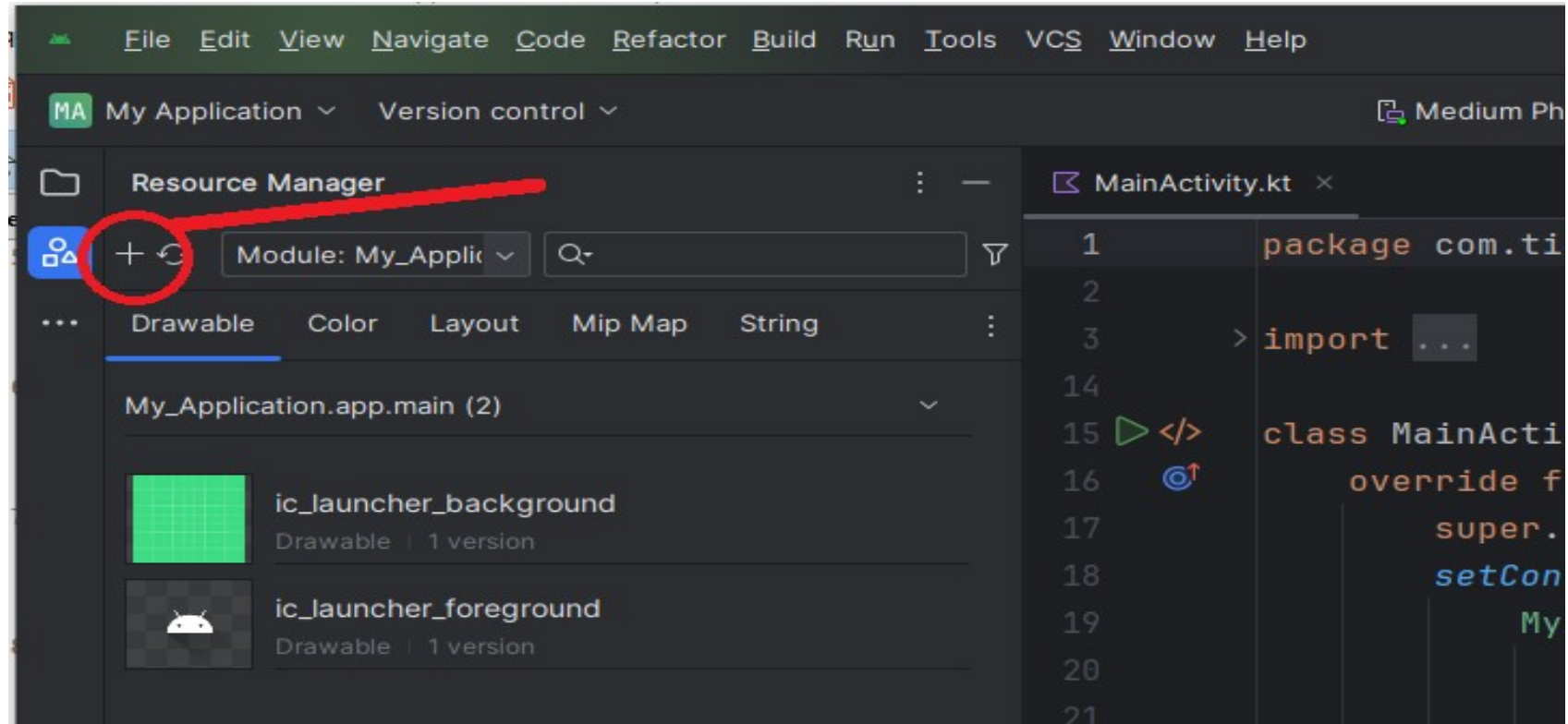
Usando modificadores

- Observe: diferentes funções modificadoras usadas juntas.
- **Clickable** faz um elemento que pode ser **@Compose** reagir à entrada do usuário.
- **padding** coloca espaço ao redor de um elemento.
- **fillMaxWidth** faz com que o elemento que pode ser composto preencha a largura máxima fornecida a ele por seu pai.
- **size()** especifica a largura e a altura preferidas de um elemento.
- os modificadores desempenham uma função semelhante à dos parâmetros de *layout* em *layouts* baseados em visualização. No entanto, como os modificadores às vezes são específicos do escopo, eles oferecem segurança de tipo e também ajudam você a descobrir e compreender o que está disponível e é aplicável a um determinado *layout*

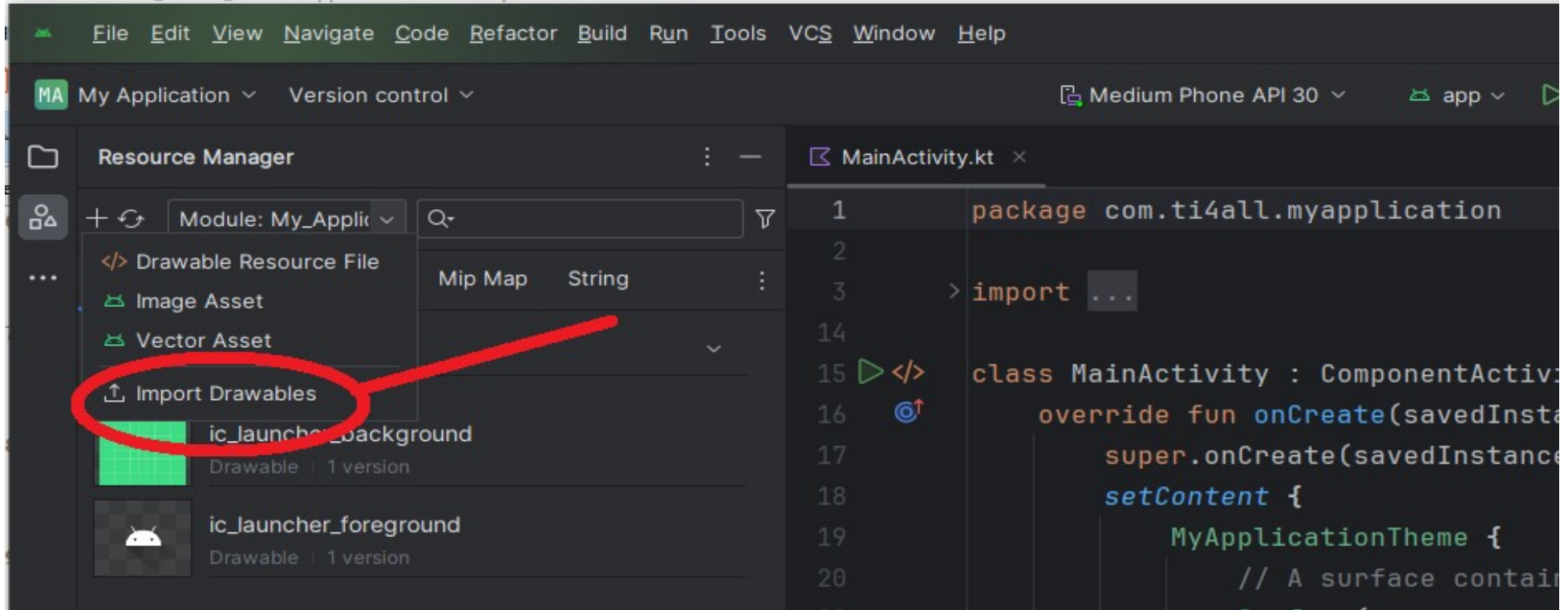
Inserindo Imagem



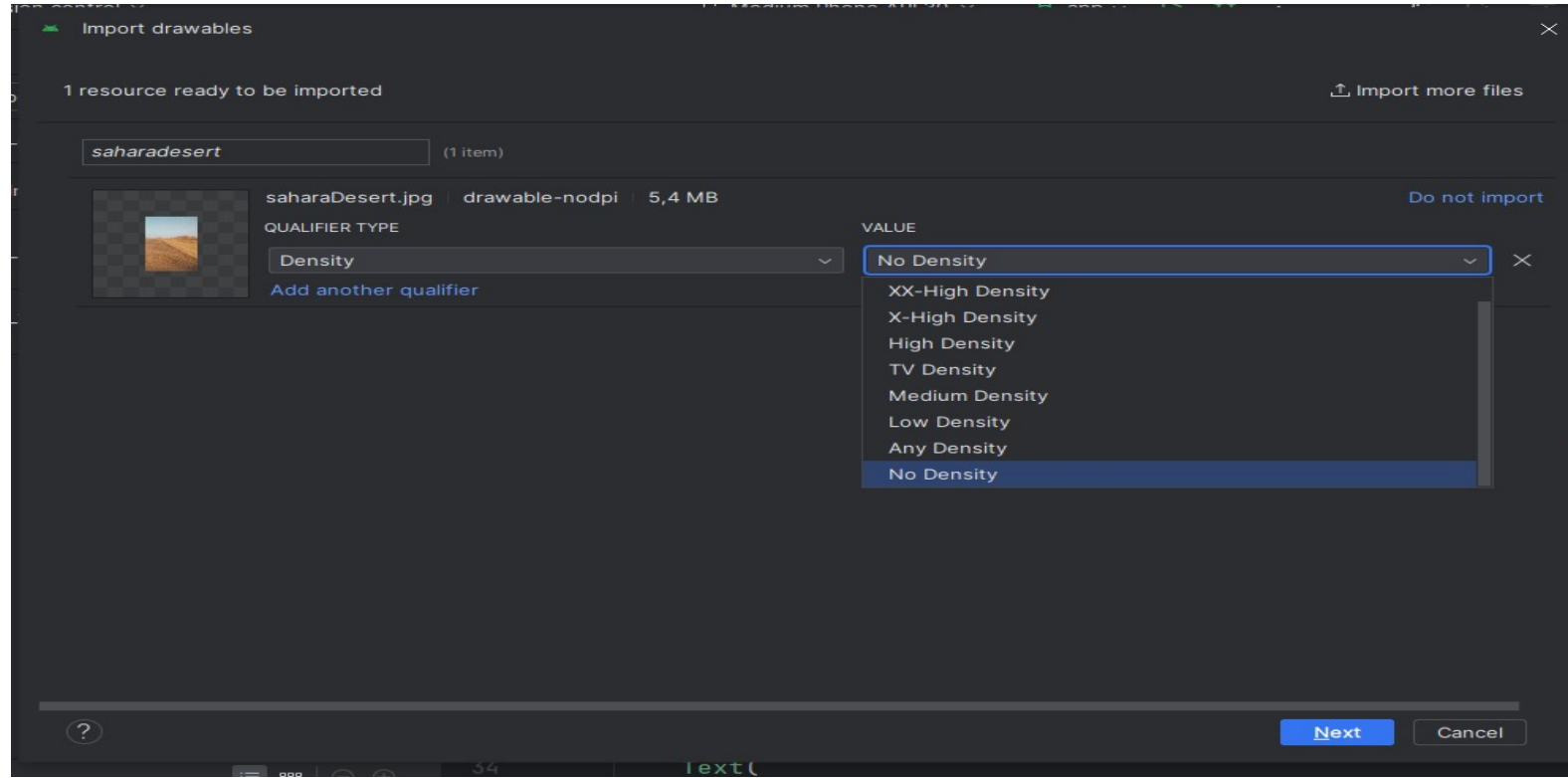
Inserindo Imagem



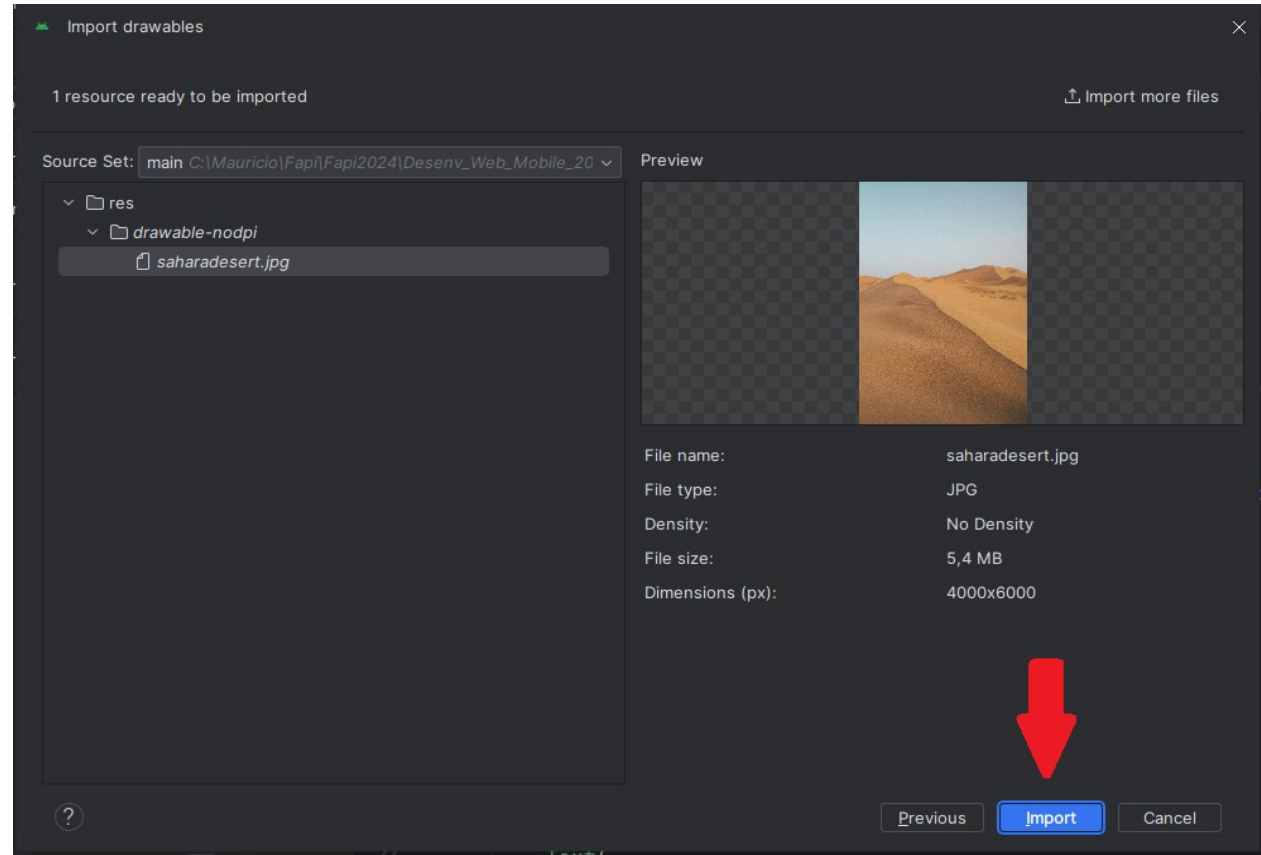
Inserindo Imagem



Inserindo Imagem



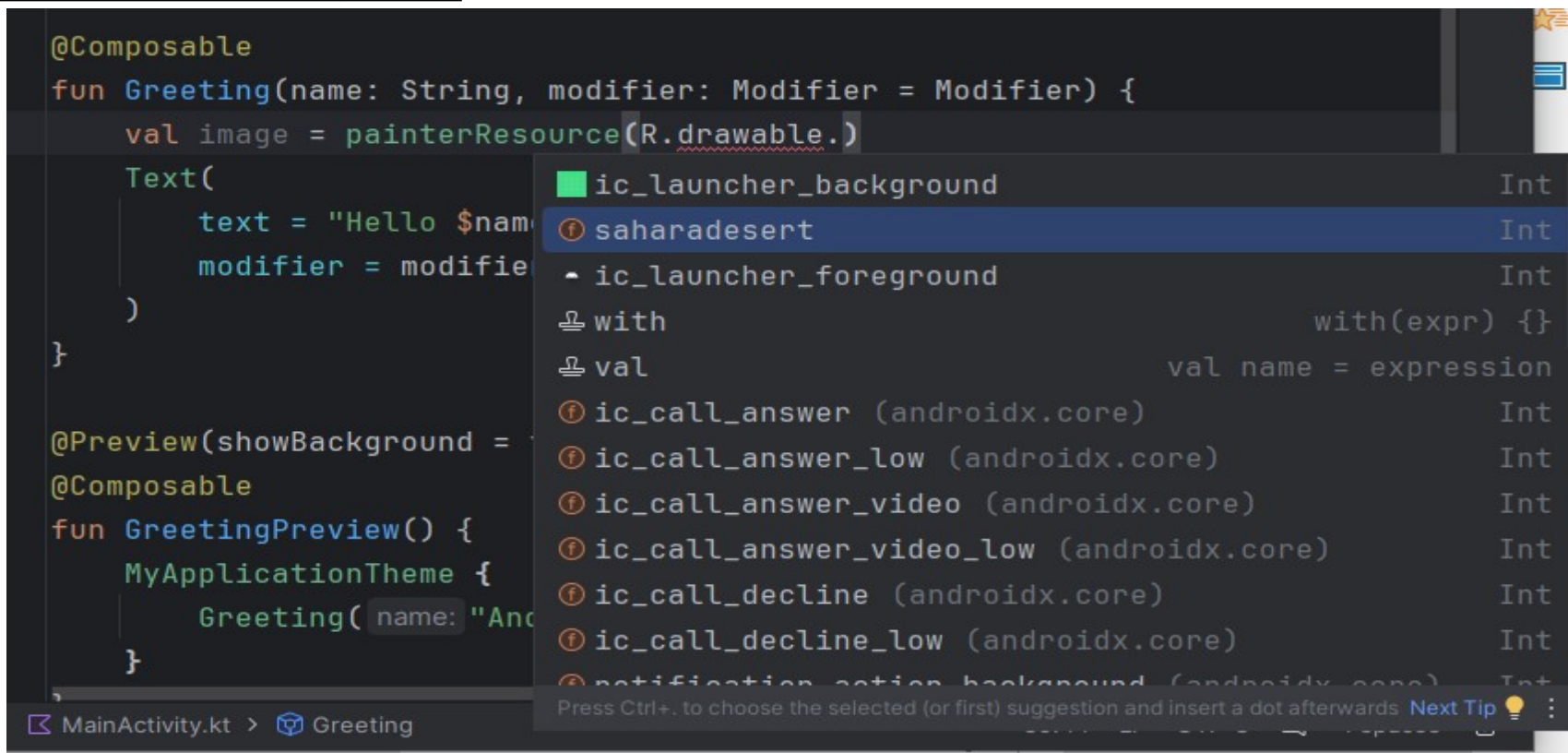
Inserindo Imagem



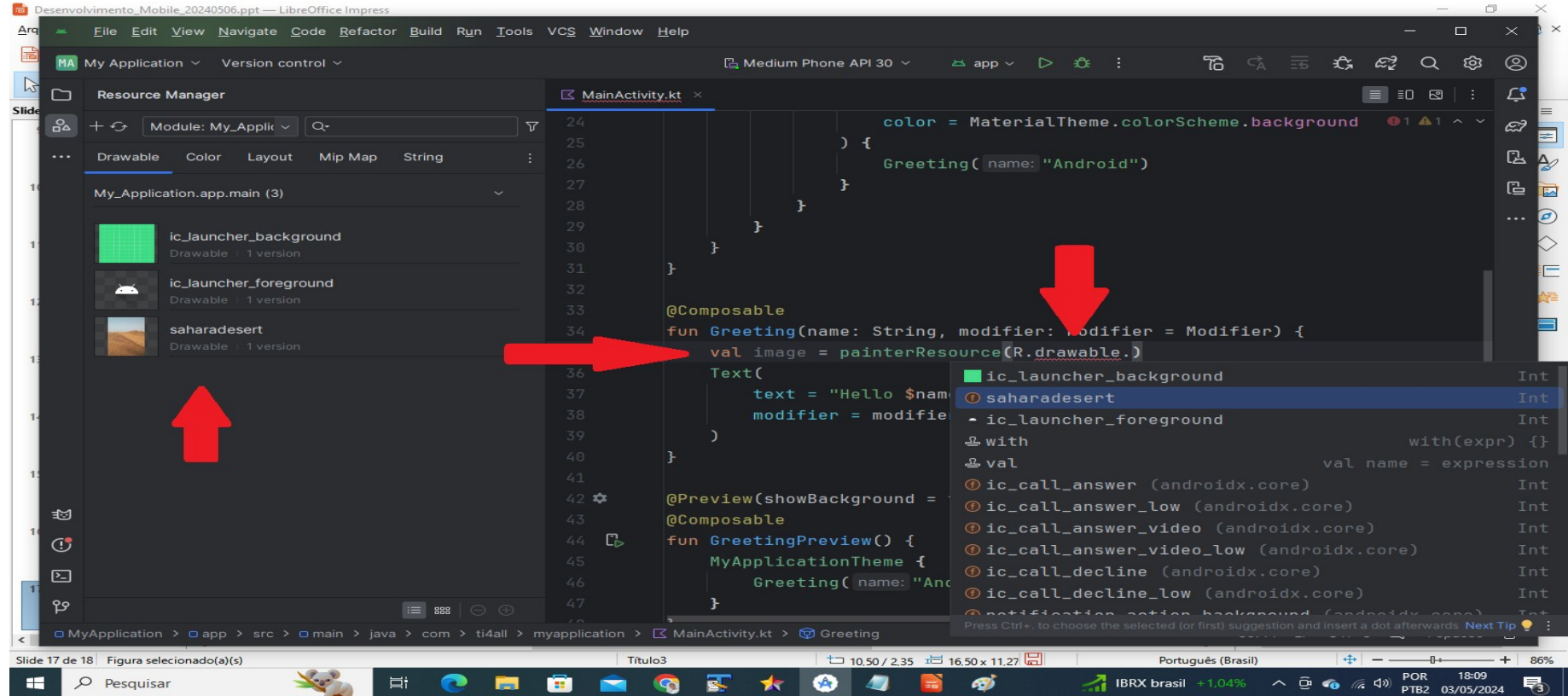
Inserindo Imagem

```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    val image = painterResource(R.drawable.)
    Text(
        text = "Hello $name",
        modifier = modifier
    )
}

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



Inserindo Imagem



Atividade (aula)

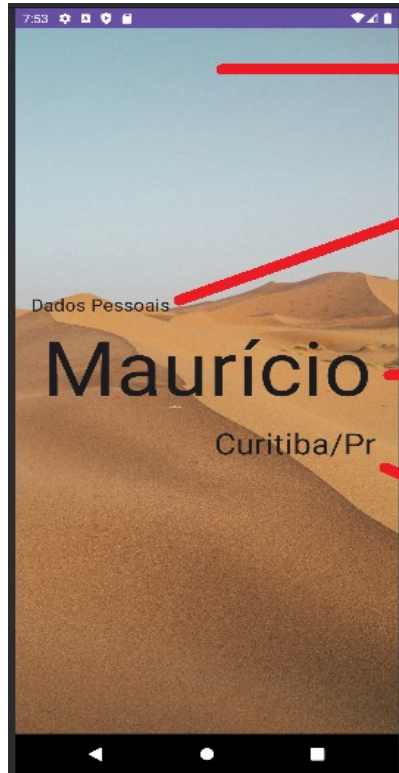


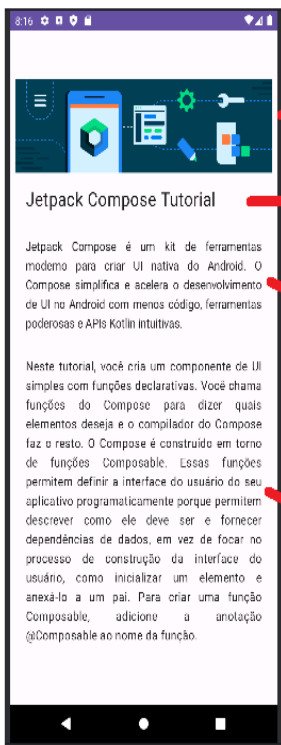
Imagem de fundo
(saharaDesert.jpg)

"Dados Pessoais"; tamanho da fonte: 20.sp;
altura da linha: 25.sp; texto alinhado à direita;
referência de alinhamento Inicial

Nome: tamanho da fonte: 90.sp; altura da
linha: 120.sp; Alinhamento central;
referência de alinhamento centro-horizontal

Cidade/Uf: tamanho da fonte: 35.sp;
padding: 8.dp; referência de alinhamento
Fim-da-linha.

Atividade (aula)



bg_compose_background.png - Defina a imagem para preencher toda a largura da tela.

Tamanho de fonte de 24 sp e preenchimento de 16 dp (início, fim, parte inferior e superior).

Tamanho de fonte padrão, preenchimento de 16 dp (início e fim) e alinhamento de texto justificado. (1)

Tamanho de fonte padrão, preenchimento de 16 dp (início, fim, parte inferior e superior) e alinhamento de texto justificado. (2)

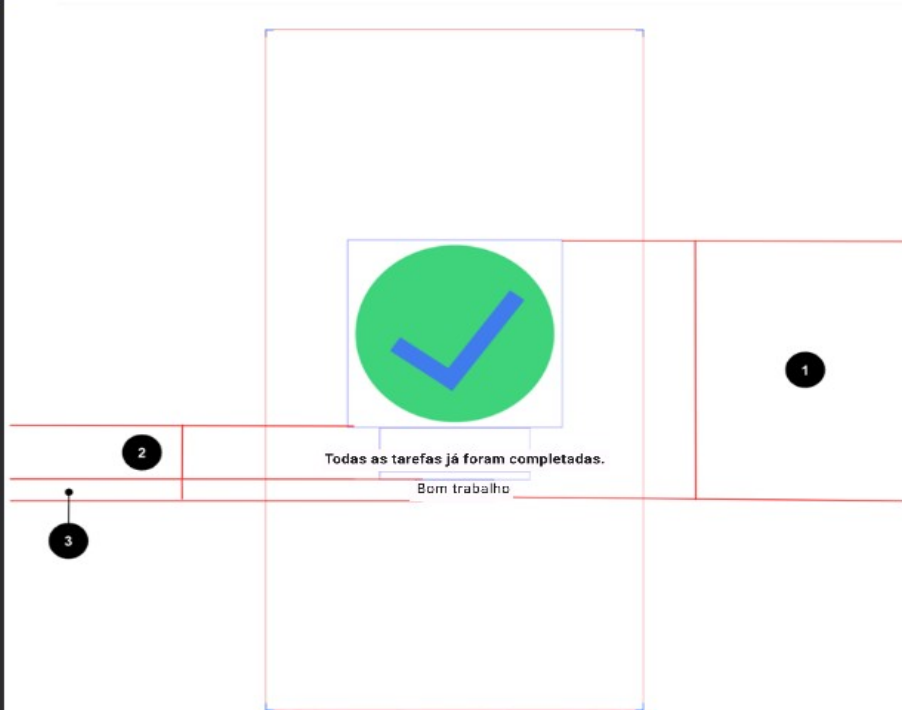
(1) Jetpack Compose é um kit de ferramentas para criar UI nativa do Android. O Compose simplifica e acelera o desenvolvimento de UI no Android com menos código, ferramentas poderosas e APIs Kotlin intuitivas.

(2) Neste tutorial, você cria um componente de UI simples com funções declarativas. Você chama funções do Compose para dizer quais elementos deseja e o compilador do Compose faz o resto.

O Compose é construído em torno de funções Composable. Essas funções permitem definir a interface do usuário do seu aplicativo programaticamente porque permitem descrever como ele deve ser e fornecer dependências de dados, em vez de focar no processo de construção da interface do usuário, como inicializar um elemento e anexá-lo a um pai.

Para criar uma função Composable, adicione a anotação `@Composable` ao nome da função.

Atividade 1



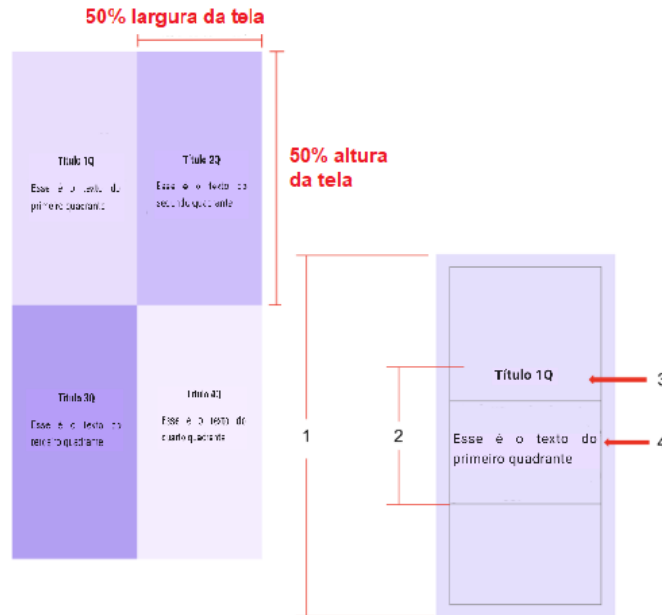
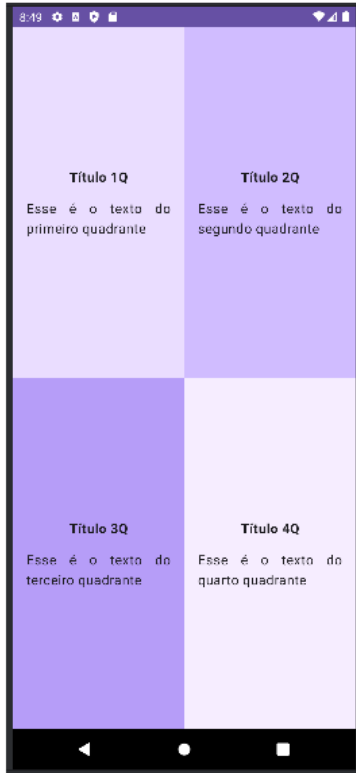
(1) ic_task_completed.png

- Centralize todo o conteúdo vertical e horizontalmente na tela.

(2) Texto de fonte Negrito, preenchimento de 24 dp na parte superior e preenchimento de 8 dp na parte inferior ("Todas as tarefas já foram completadas").

(3) Texto tamanho de fonte 16sp ("Bom trabalho").

Atividade 2



(1) Defina todo o quadrante (início, fim, superior e inferior) para um preenchimento de 16 dp.

(2) Centralize todo o conteúdo vertical e horizontalmente em cada quadrante.

(3) Formate o primeiro Texto em negrito e defina-o com um preenchimento inferior de 16 dp (“Título 1Q”, “Título 2Q”; “Título 3Q” e “Título 4Q”).

(4) Defina o segundo texto com tamanho de fonte padrão (“Esse é o texto do 1º. Quadrante”, “... do 2º. ...”, “...do 3º. ...”, “...do 4º....”).