Desenvolvimento para Dispositivos Móveis

Modificadores de Layout, Estilo e de Interação

Prof. Bruno Azevedo

UNIP – Universidade Paulista

Modifier no Jetpack Compose

- `Modifier` é um objeto que descreve como um componente deve ser exibido e comportar-se.
- Ele permite aplicar alterações visuais e de layout, como:
  - Tamanho e posição (`fillMaxSize`, `padding`, `weight`).
  - Aparência (`background`, `border`, `shadow`).
  - Interatividade (clicável, arrastável, etc).
- `Modifier` é imutável e pode ser encadeado (chamadas em sequência).
- Exemplo:

```
Text(
  text = "Algum texto!",
  modifier = Modifier
    .padding(16.dp)
    .background(Color.LightGray)
)
```

- O texto terá espaçamento interno de 16 dp e fundo cinza claro.

Modificadores de Layout

- Os modificadores de layout controlam o tamanho e o posicionamento dos `Composables`.
- São aplicados com `modifier = Modifier...` e podem ser encadeados.
- Permitem definir tamanhos, alinhamentos, espaçamentos e etc.

padding

- Adiciona espaço interno entre o conteúdo e as bordas do componente.
- Pode ser aplicado igualmente em todas as direções ou de forma específica (horizontal, vertical, start, end, etc.).

```
Text(
  text = "Exemplo com padding",
  modifier = Modifier
    .padding(16.dp) // Espaço interno de 16.dp em todos os lados
    .background(Color.Yellow)
)
```

- O texto ficará com um fundo amarelo e terá 16.dp de espaço entre o conteúdo e as bordas.

fillMaxSize()

- O modificador `fillMaxSize()` faz o composable ocupar todo o espaço disponível do pai.
- Utilizado quando queremos que o componente preencha a tela ou o contêiner onde está inserido.

```
Box(
  modifier = Modifier
    .fillMaxSize()
    .background(Color.LightGray)
) {
  Text(
    "Preenchendo a tela",
    modifier = Modifier.align(Alignment.Center)
  )
}
```

- A Box ocupa toda a tela.
- O texto é centralizado usando `align()`.

wrapContentSize()

- O modificador `wrapContentSize()` ajusta o tamanho do componente ao seu conteúdo.
- Pode ser usado com um alinhamento para posicionar o conteúdo dentro do espaço disponível.

```
Box(
  modifier = Modifier
    .fillMaxSize()
    .wrapContentSize(Alignment.BottomEnd)
    .background(Color.LightGray)
) {
  Text(
    "Texto no canto inferior direito",
    modifier = Modifier
      .background(Color.Yellow)
      .padding(8.dp)
  )
}
```

- O texto mantém seu tamanho natural.
- A `wrapContentSize` alinha o conteúdo no canto inferior direito.

size

- Define simultaneamente a `largura` e a `altura` de um componente.
- Pode receber um único valor (para largura e altura iguais) ou dois valores distintos.
- Garante que o componente tenha dimensões fixas, independentemente do conteúdo interno.

```
Box(
  modifier = Modifier
    .size(width = 200.dp, height = 100.dp)
    .background(Color.Magenta)
)
```

- Neste exemplo, a `Box` terá exatamente 200 dp de largura e 100 dp de altura.

width e height

- Permitem definir separadamente a largura e a altura do componente.
- width() fixa a largura; o conteúdo maior pode ser cortado ou adaptado.
- height() fixa a altura; comporta-se de forma semelhante quanto ao conteúdo.
- A unidade dp (density-independent pixels) garante tamanho consistente em diferentes telas.

```
Box(
  modifier = Modifier
    .width(200.dp)
    .height(100.dp)
    .background(Color.Gray)
)
```

- O efeito visual é o mesmo que usar size(200.dp, 100.dp).

fillMaxWidth e fillMaxHeight

- `fillMaxWidth()`: faz o componente ocupar toda a largura disponível no contêiner pai.
- `fillMaxHeight()`: faz o componente ocupar toda a altura disponível no contêiner pai.
- Utilizamos quando queremos que o componente se expanda completamente em uma das direções.

```
Box(
  modifier = Modifier
    .fillMaxWidth()
    .height(80.dp)
    .background(Color.Cyan)
)
```

- A `Box` acima terá altura fixa de 80.dp e ocupará toda a largura do pai.

offset

- Move o componente a partir da sua posição original, sem alterar seu tamanho nem o layout ao redor.
- Os valores de deslocamento podem ser positivos ou negativos.

```
Box(
  modifier = Modifier
    .size(100.dp)
    .background(Color.LightGray)
) {
  Text(
    "Deslocado",
    modifier = Modifier
      .offset(x = 20.dp, y = 12.dp)
      .background(Color.Green)
  )
}
```

- A Box cinza representa o espaço original.
- O Text aparece deslocado dentro dela, 20.dp para a direita e 12.dp para baixo.

align

- Posiciona o componente dentro de um `Box`, de acordo com o alinhamento especificado.
- Só pode ser usado diretamente dentro de um `Box`.
- Os alinhamentos mais comuns são: `Alignment.TopStart`, `Alignment.Center`, `Alignment.BottomEnd`, entre outros.

```
Box(
  modifier = Modifier
    .size(200.dp)
    .background(Color.LightGray)
) {
  Text(
    "Centralizado",
    modifier = Modifier
      .align(Alignment.Center)
      .background(Color.Yellow)
  )
}
```

- O `Box` serve como contêiner visual (cinza).
- O `Text` é centralizado dentro do `Box` com `Alignment.Center`.

Modifier.weight()

- Distribui o espaço disponível proporcionalmente entre elementos de uma `Row` ou `Column`.
- Exemplo:

```
Row {
    Button(
        onClick = {},
        modifier = Modifier.weight(1f)
    ) {
        Text("Botão 1")
    }

    Button(
        onClick = {},
        modifier = Modifier.weight(2f)
    ) {
        Text("Botão 2")
    }
}
```

- O espaço disponível será dividido em três partes iguais e o segundo botão ocupará dois terços do espaço.
- O número deve ser um `float`, portanto, o `f` ao lado do valor.

Alinhamento em Row

- Em Row, o eixo principal é horizontal.
- verticalAlignment alinha os filhos verticalmente dentro da linha.
- horizontalArrangement define espaçamento e alinhamento horizontal entre os filhos.
- Exemplo:

```
Row(
    modifier = Modifier.fillMaxWidth().background(Color.LightGray),
    verticalAlignment = Alignment.CenterVertically,
    horizontalArrangement = Arrangement.SpaceBetween
)
```

- verticalAlignment = Alignment.CenterVertically posiciona os filhos centralizados verticalmente dentro da Row.
- horizontalArrangement = Arrangement.SpaceBetween distribui os filhos ao longo da largura disponível, com espaços iguais entre eles.

Alinhamento em Column

- Em `Column`, o eixo principal é vertical.
- `horizontalAlignment` alinha os filhos horizontalmente dentro da coluna.
- `verticalArrangement` define espaçamento e alinhamento vertical entre os filhos.
- Exemplo:

```
Column(
    modifier = Modifier.fillMaxHeight().background(Color.LightGray),
    horizontalAlignment = Alignment.CenterHorizontally,
    verticalArrangement = Arrangement.spacedBy(16.dp)
)
```

- `horizontalAlignment = Alignment.CenterHorizontally` posiciona os filhos centralizados horizontalmente dentro da `Column`.
- `verticalArrangement = Arrangement.spacedBy(16.dp)` adiciona espaçamento fixo de 16 dp entre os filhos.

Opções de `verticalAlignment` e `horizontalArrangement` em `Row`

- `verticalAlignment` (alinhamento vertical dos filhos na linha):
  - `Alignment.Top`: Alinha ao topo.
  - `Alignment.CenterVertically`: Centraliza verticalmente.
  - `Alignment.Bottom`: Alinha à base.
- `horizontalArrangement` (arranjo horizontal dos filhos na linha):
  - `Arrangement.Start`: Agrupa os filhos à esquerda.
  - `Arrangement.Center`: Agrupa os filhos no centro horizontal.
  - `Arrangement.End`: Agrupa os filhos à direita.
  - `Arrangement.SpaceBetween`: Espaço máximo entre o primeiro e último filho, distribuindo os demais igualmente.
  - `Arrangement.SpaceAround`: Espaço igual ao redor dos filhos.
  - `Arrangement.SpaceEvenly`: Espaços iguais antes, entre e depois dos filhos.
  - `Arrangement.spacedBy(dp)`: Espaçamento fixo entre os filhos.

Opções de `horizontalAlignment` e `verticalArrangement` em `Column`

- `horizontalAlignment` (alinhamento horizontal dos filhos na coluna):
  - `Alignment.Start`: Alinha à esquerda.
  - `Alignment.CenterHorizontally`: Centraliza horizontalmente.
  - `Alignment.End`: Alinha à direita.
- `verticalArrangement` (arranjo vertical dos filhos na coluna):
  - `Arrangement.Top`: Agrupa os filhos no topo.
  - `Arrangement.Center`: Agrupa os filhos no centro vertical.
  - `Arrangement.Bottom`: Agrupa os filhos na base.
  - `Arrangement.SpaceBetween`: Espaço máximo entre o primeiro e último filho, distribuindo os demais igualmente.
  - `Arrangement.SpaceAround`: Espaço igual ao redor dos filhos.
  - `Arrangement.SpaceEvenly`: Espaços iguais antes, entre e depois dos filhos.
  - `Arrangement.spacedBy(dp)`: Espaçamento fixo entre os filhos.

Modificadores de Estilo

- Modificadores de estilo são usados para alterar a aparência visual dos componentes no Jetpack Compose.
- Permitem a personalização de cores, formas, bordas, recortes, transparência e sombras.
- Assim como os modificadores de layout, são aplicados via `Modifier`

background

- O modificador `background` define a cor de fundo de um componente.
- Ele preenche toda a área do componente, atrás de seu conteúdo.
- Exemplo:

```
Box(
  modifier = Modifier
    .background(Color.Blue)    // cor de fundo azul sólida
)
```

- Neste exemplo, uma caixa azul.

shape

- O modificador `shape` define o formato do fundo quando usado com `background`.
- Permite criar bordas arredondadas, círculos, formas personalizadas, etc.
- Exemplo:

```
Box(
  modifier = Modifier
    .size(100.dp)
    .background(
      color = Color.Red,                    // cor vermelha
      shape = RoundedCornerShape(16.dp) // cantos arredondados de 16dp
    )
)
```

- Acima, uma caixa com cantos arredondados.
- Quando há mais de um parâmetro, o Kotlin exige que você indique explicitamente o nome do parâmetro.

border

- O modificador `border` adiciona uma borda visível ao redor do componente.
- Você pode escolher a espessura, a cor e a forma da borda.
- A borda respeita o formato definido pelo `shape` caso usado.
- Exemplo:

```
Box(
  modifier = Modifier
    .size(100.dp)
    .border(
      width = 2.dp,                          // espessura da borda
      color = Color.Black,                   // cor preta
      shape = RoundedCornerShape(8.dp)     // cantos arredondados
    )
)
```

- Neste exemplo, uma caixa com borda preta e cantos arredondados.

clip

- O modificador `clip` recorta a área visível do componente conforme uma forma.
- Tudo que ultrapassar essa forma não será exibido.
- Pode ser usado com formas prontas como `CircleShape` ou personalizadas.
- Exemplo:

```
Box(
  modifier = Modifier
    .size(100.dp)
    .clip(CircleShape)        // recorta em círculo
    .background(Color.Green)
)
```

- Aqui, um componente quadrado cortado em círculo.

alpha

- O modificador `alpha` controla a opacidade do componente.
- Aceita valores entre 0f (transparente) e 1f (totalmente opaco).
- Pode ser aplicado a qualquer componente para ajustar sua visibilidade.
- Exemplo:

```
Box(
  modifier = Modifier
    .size(100.dp)
    .background(Color.Magenta)
    .alpha(0.5f)                    // 50% transparente
)
```

- Aqui, uma caixa rosa com 50% de transparência.

shadow

- O modificador `shadow` adiciona sombra ao redor do componente.
- Ajuda a criar sensação de profundidade e destacar elementos na tela.
- Permite definir o tamanho (elevation) e forma da sombra.
- Exemplo:

```
Box(
  modifier = Modifier
    .shadow(
      elevation = 8.dp,                    // intensidade da sombra
      shape = RoundedCornerShape(12.dp) // formato da sombra
    )
    .background(Color.Yellow)
)
```

- Acima, uma caixa amarela com sombra arredondada.

Modificadores de Interação

- `clickable`: torna o componente clicável para executar ações.
- `toggleable`: permite alternar entre estados ligados e desligados.
- `scrollable`: habilita rolagem manual em componentes específicos.

Modificador clickable

- Permite capturar toques e executar uma ação.
- Pode ser aplicado a textos, imagens, caixas e botões personalizados.
- Exemplo:

```
Text(
  "Clique aqui",
  modifier = Modifier
    .padding(16.dp)
    .clickable {
      println("Texto clicado!")
    }
)
```

- Mostra uma mensagem no console ao clicar.

Modificador toggleable

- Indica que um componente pode alternar entre dois estados: ligado e desligado.
- Usado para switches, checkboxes ou qualquer item que tenha estado binário.
- Exemplo:

```
var selecionado by remember { mutableStateOf(false) }
Box(
  modifier = Modifier
    .size(100.dp)
    .background(if (selecionado) Color.Green else Color.Red)
    .toggleable(
      value = selecionado,
      onValueChange = { selecionado = it }
    )
)
```

- Acima, uma caixa que muda de cor ao ser selecionada.
- O estado `selecionado` controla o comportamento e aparência do componente.

Modificador scrollable

- Permite que o usuário role o conteúdo manualmente.
- Utilizando quando temos conteúdo maior que o espaço disponível.
- Deve ser usado com um estado de rolagem para controlar a posição atual.
- Exemplo:

```
val scrollState = rememberScrollState()
Column(
  modifier = Modifier
    .height(200.dp)
    .verticalScroll(scrollState)
    .background(Color.LightGray)
    .padding(8.dp)
)
```

- Neste exemplo, uma coluna que pode ser rolada verticalmente.
- Use `verticalScroll` para rolagem vertical e `horizontalScroll` para rolagem horizontal.

OutlinedTextField

- O `OutlinedTextField` é um composable com borda utilizado para entrada de texto.
- Exibe um rótulo (`label`) que pode flutuar acima da caixa de texto.

```
var texto by remember { mutableStateOf("") }
OutlinedTextField(
    value = texto,
    onValueChange = { texto = it },
    label = { Text("Digite algo") }
)
```

Atividade 8

- Façam a atividade 8.