



# Flutter Básico - Aula 1

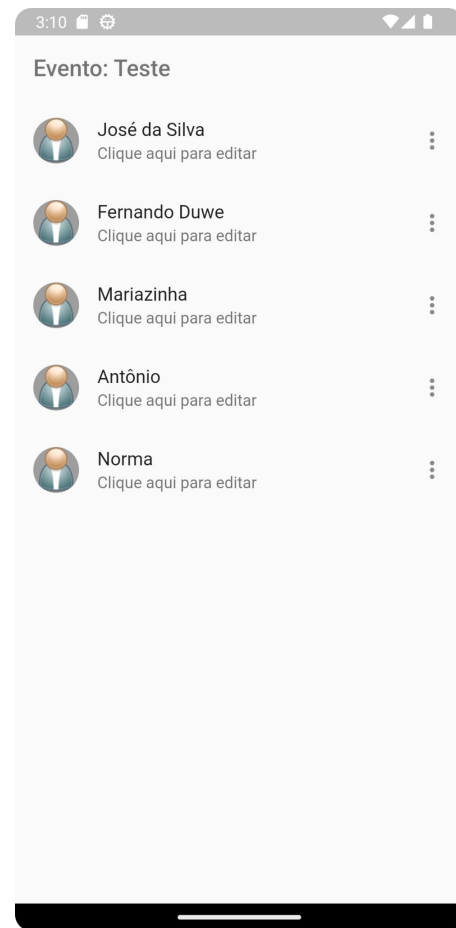
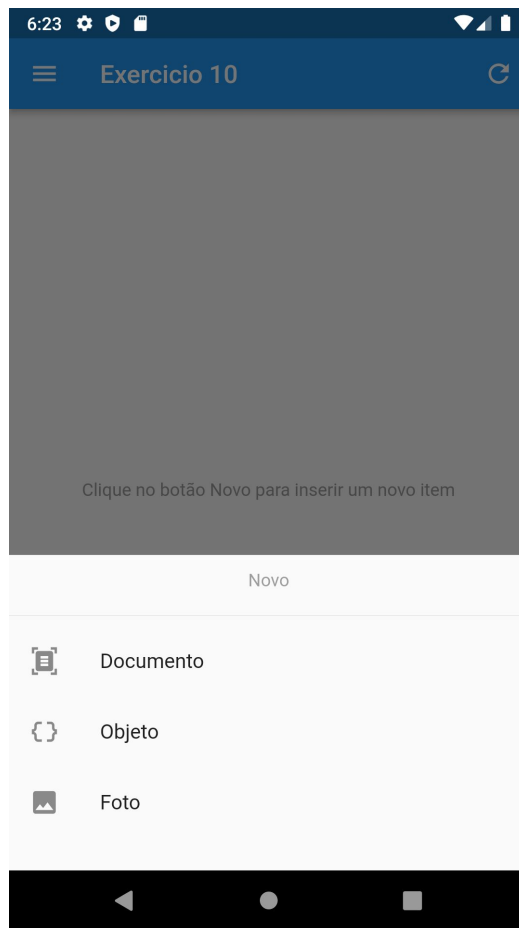
Exercícios 1 a 4



- Flutterando; [Ver mais](#)
- Flutter - Youtube; [Ver mais](#)

# Personalizando Widgets

- A partir de agora, criaremos os nossos próprios widgets
- Widgets que podem ser reutilizados
- Como criar seu próprio widget?





- Para criarmos os nossos próprios widgets, basta criarmos classes, herdando de `StatelessWidget` ou `StatefulWidget`
- Após criarmos nossos widgets, podemos reaproveitá-los em vários locais da nossa aplicação



- Como podemos enviar informação para o nosso widget?

```
class SimpleWidget extends StatelessWidget {  
  final int? containerCount;  
  final VoidCallback? onClick;  
  
  SimpleWidget({ this.containerCount, this.onClick, Key? key}) : super(key: key);  
  
  @override  
  Widget build(BuildContext context) {  
    return GestureDetector(  
      onTap: this.onClick,  
      child: Container(  
        padding: EdgeInsets.all(8),  
        child: Text("Sou um Container ${this.containerCount}"),  
      ), // Container  
    ); // GestureDetector  
  }  
}
```

# Personalizando Widgets: Enviando eventos para o nosso widget




- Adicionando eventos ao nosso widget

```
class SimpleWidget extends StatelessWidget {  
  final int? containerCount;  
  final VoidCallback? onClick;  
  
  SimpleWidget({ this.containerCount, this.onClick, Key? key}) : super(key: key);  
  
  @override  
  Widget build(BuildContext context) {  
    return GestureDetector(  
      onTap: this.onClick,  
      child: Container(  
        padding: EdgeInsets.all(8),  
        child: Text("Sou um Container ${this.containerCount}"),  
      ), // Container  
    ); // GestureDetector  
  }  
}
```



## Personalizando Widgets: Enviando eventos para o nosso widget - Eventos



```
typedef VoidCallback = void Function();
```

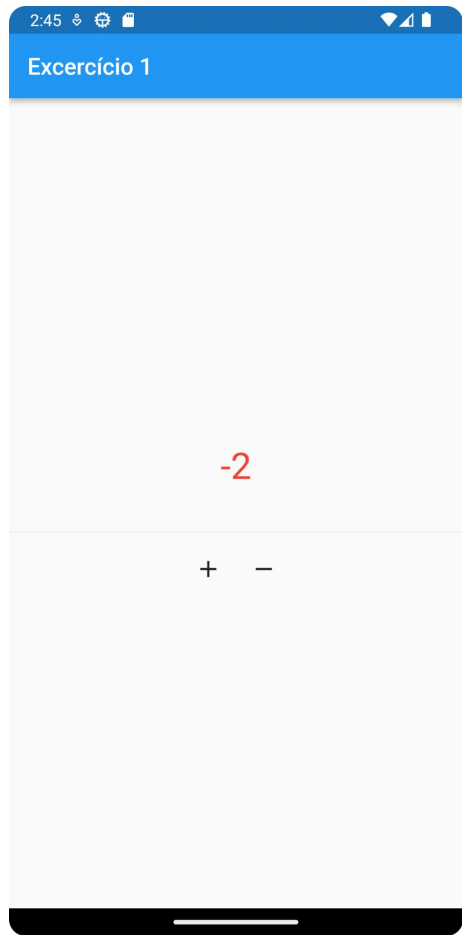
- Podemos criar eventos para enviar e receber informação ao nosso widget

```
class SimpleWidget extends StatelessWidget {  
  final int? containerCount;  
  final VoidCallback? onClick;  
  
  SimpleWidget({this.containerCount, this.onClick, Key? key}) : super(key: key);  
  
  @override  
  Widget build(BuildContext context) {  
    return Container(  
      padding: EdgeInsets.all(8),  
      child: Row(  
        children: [  
          Expanded(child: Text("Sou um Container ${this.containerCount}")),  
          ElevatedButton(onPressed: this.onClick, child: Text("Clicar"))  
        ],  
      ));  
  }  
}
```

- Vamos recriar o app contador (Exercício 3, do módulo anterior)
- Devem ser exibidos dois botões em tela, para incrementar ou decrementar o contador
- A exibição do contador deve ser feita em um widget separado
- Caso o contador seja maior ou igual a zero, o contador deve ficar em verde
- Caso seja menor que zero, deve ficar em vermelho

-2

7







- Adicionar restrições de tamanho para objetos como Containers
- Possui quatro propriedades: minHeight, maxHeight, minWidth e maxWidth
- Controla o tamanho máximo e mínimo que o widget filho pode ter
- Você pode usar também o widget ConstrainedBox

Height = Altura  
Width = Largura

```
child: Text( 'Olá, eu sou um texto que pode ficar bem grande'),  
constraints: BoxConstraints(  
  minHeight: 0,  
  minWidth: 0,  
  maxHeight: 500,  
  maxWidth: 500,  
), // BoxConstraints
```



- Se expande para o maior espaço disponibilizado pelo seu widget pai
- Seta as propriedades minHeight, maxHeight, minWidth e maxWidth para infinito (double.infinity)
- Pode receber por parâmetro um height e/ou width, nesse caso setando valor para as propriedades min e max

Height = Altura  
Width = Largura

```
...  
constraints: BoxConstraints.expand(),  
  
constraints: BoxConstraints.expand(  
    height: 350  
),
```



- Define o minHeight e minWidth igual a 0
- Os tamanhos máximos de altura e largura são definidos pelo parâmetro passado, que é do tipo Size

```
// Size(double width, double height)  
  
constraints: BoxConstraints.loose(Size(300, 300))
```

```
Height = Altura  
Width  = Largura
```



- Também recebe os parâmetros de altura e largura a partir de um widget do tipo Size
- Define o minHeight e maxHeight iguais aos parâmetros passados para size.height
- Define o minWidth e maxWidth iguais aos parâmetros passados para size.width

```
// Size(double width, double height)  
  
constraints: BoxConstraints.tight(Size(300, 300))
```

```
Height = Altura  
Width  = Largura
```



- Pode receber um tamanho de width ou height
- Ambos são opcionais
- Caso preenchido, preenche o valor mínimo e máximo de acordo com o que foi passado por parâmetro

```
constraints: BoxConstraints.tightFor(width: 300)
```

```
Height = Altura  
Width  = Largura
```



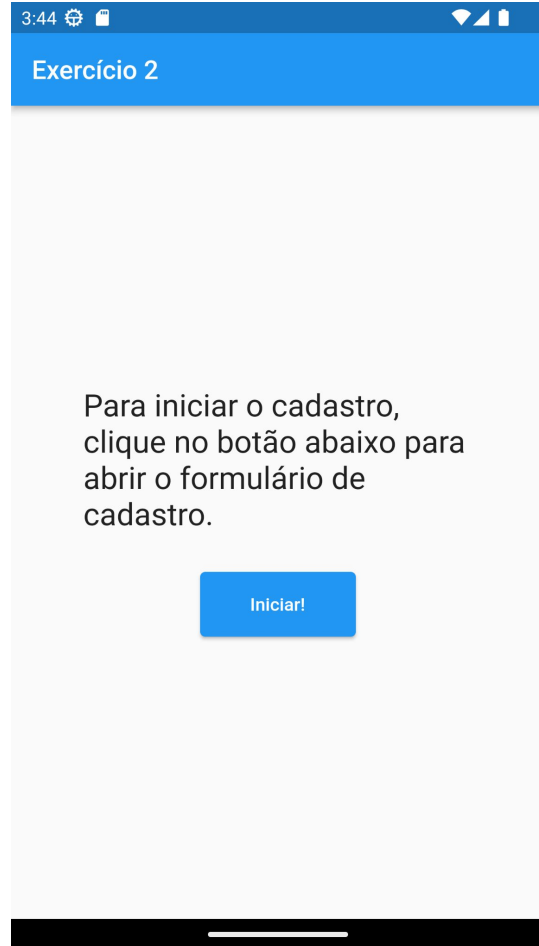
- Pode receber um tamanho de width ou height
- Ambos são opcionais
- Somente levará em consideração valores informados diferentes de infinitos

```
constraints: BoxConstraints.tightForFinite(height: 300)
```

```
Height = Altura  
Width  = Largura
```



- Utilizando as constraints que vimos anteriormente, desenhe o seguinte layout
- O texto exibido deve ter sua largura limitada para que fique semelhante ao exemplo
- O botão de Iniciar deve ter um tamanho maior ao botão padrão





- Para detectar qualquer gesto que desejar, utilize o GestureDetector
- Gestos como clique, duplo clique, pressionar com força, swipe, zoom, entre outros

```
GestureDetector(  
  onTap: () => print("onTap")  
  child: Container(  
    padding: EdgeInsets.all(30),  
    child: Text("Me Detecte"),  
    color: Colors.lightGreen,  
  ),  
)
```





```
onTap: () => print("Ao pressionar - clique")
```

```
onDoubleTap: () => print("Duplo Clique")
```

```
onForcePressEnd: (ForcePressDetails details) {  
    print("Pressionou com força. Força usada: ${details.pressure}")  
}
```



```
onPanUpdate: (details) => print("Efetuou Swipe")
```

```
onScaleStart: (details) => print("Zoom começou"),  
onScaleUpdate: (details) {  
  print("Zoom está acontecendo")  
}
```



- Vamos criar um app de lâmpada
- Ao dar um clique na lâmpada, ela deve acender
- Abaixo dessa lâmpada, ao dar um duplo clique, a cor da luz da lâmpada deve ser trocada
- Cores disponíveis: Amarelo, Azul e Verde



```
onPanUpdate: (details) {  
    if (details.delta.dx > 0) // Swipe para a direita  
  
    if (details.delta.dx < 0) // Swipe para a esquerda  
},
```

```
onPanUpdate: (details) {  
    this.sentidoSwipe = (details.delta.dx > 0 ? "direita" : "esquerda");  
},  
  
onPanEnd: (details) {  
    if (this.sentidoSwipe == "direita") {  
        // Levantou o dedo, apontando para a direita  
    } else {  
        // Levantou o dedo, apontando para a esquerda  
    }  
},
```



- Vamos criar um aplicativo tipo Tinder
- No Container ?, ao fazer o Swipe para a direita incrementa um contador de votos positivos
- No Container ?, ao fazer o Swipe para a esquerda incrementa um contador de votos negativos
- No topo da tela, exibe os contadores para testes

