

Componentes Visuais Básicos

Parte 1



Container

- É uma das classes mais importantes do Flutter. É parecido com o JPainel do Java Swing e os Fragments do Android Java/Kotlin.
- Consiste num “espaço de tela” onde é possível colocar outro widget. Nesse espaço é possível definir coisas como preenchimento, bordas, margens, paddings, etc.

```
import 'package:flutter/material.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Usando Container',
      theme: ThemeData(
        primarySwatch: Colors.blue,
        visualDensity: VisualDensity.adaptivePlatformDensity,
      ), // ThemeData
      home: MyHomePage(),
    ); // MaterialApp
  }
}

class MyHomePage extends StatefulWidget {
  @override
  _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Usando Container"),
      ), // AppBar
      body: _body(),
    ); // Scaffold
  }

  _body() {
    return Container(color: Colors.yellow);
  }
}
```

Container

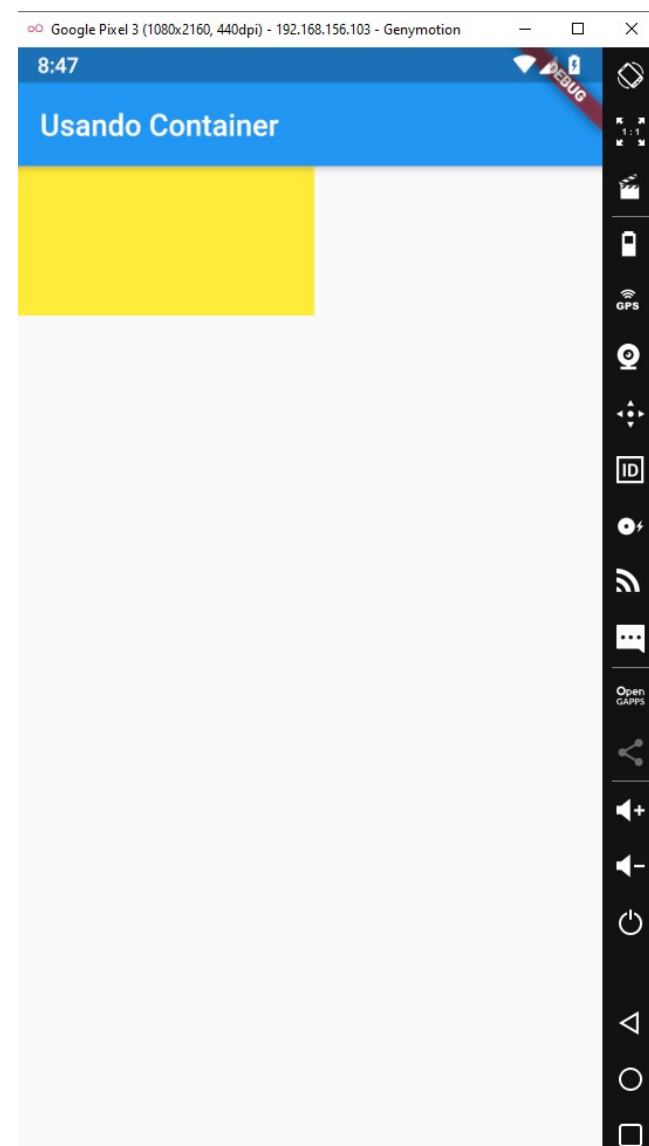
- A partir desse slide trabalharemos apenas o método `_body()`, o resto do código apresentado no slide anterior não será alterado.
- Como resultado do código da tela anterior temos o aplicativo da tela ao lado:



Container

- height: determina a altura do Container.
- width: determina a largura do Container.
- color: determinar a cor do Container.

```
_body() {  
  return Container(  
    color: Colors.yellow,  
    height: 100,  
    width: 200,  
  );  
}
```



Container – margin e padding

Note que quando retiramos height e width
o Container buscou obter todo o espaço disponível

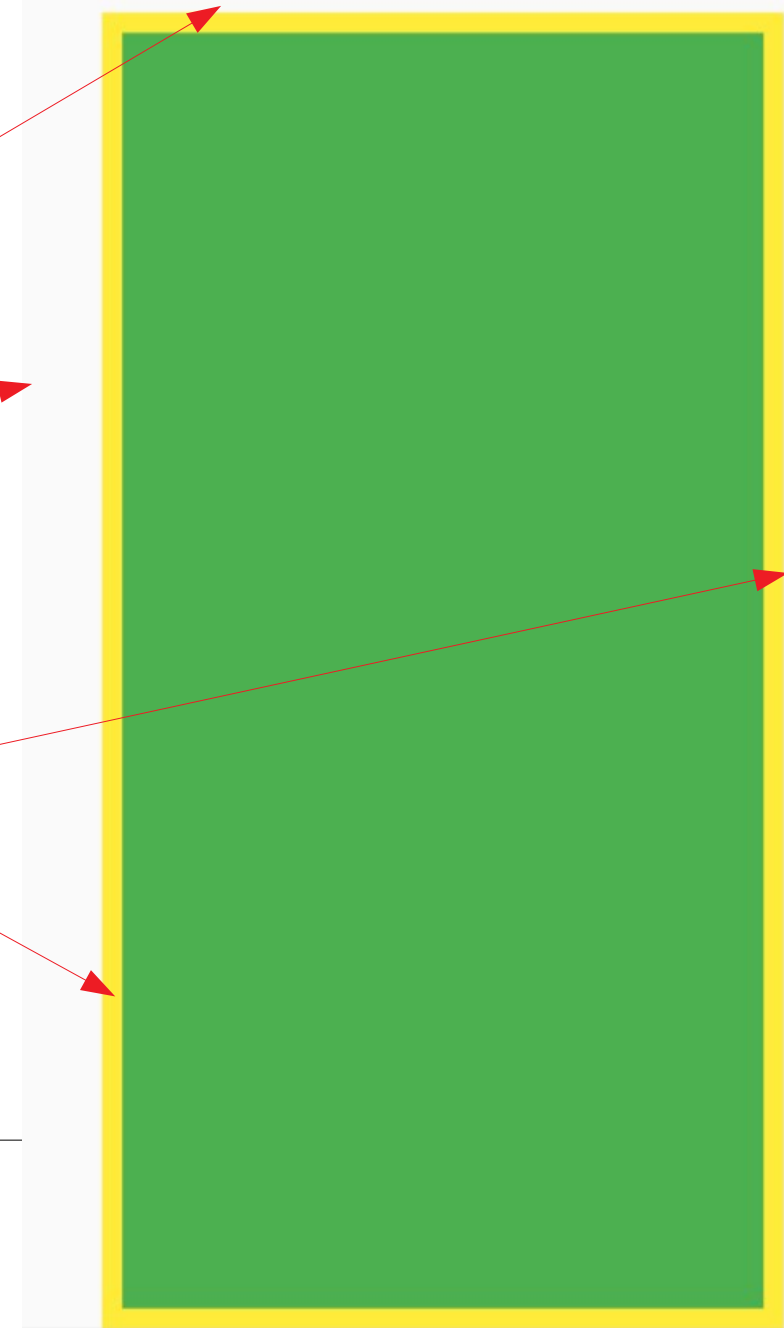
```
_body() {  
  return Container(  
    margin: EdgeInsets.only(top: 16, left: 40, right: 20),  
    padding: EdgeInsets.all(10),  
    color: Colors.yellow,  
    child: Container(  
      color: Colors.green,  
    ), // Container  
  ); // Container  
}
```

margin é uma margem externa ao widget

padding é uma margem interna ao widget
(para colocação de objetos dentro dele)

8:55

Usando Container



Container - BoxDecoration

- O BoxDecoration possibilita uma borda “estilizada” no widget. Seu uso não permite o uso do parâmetro color.

```
_body() {  
  return Container(  
    margin: EdgeInsets.only(top: 16, left: 40, right: 20),  
    padding: EdgeInsets.all(10),  
    color: Colors.yellow,  
    child: Container(  
      color: Colors.green,  
      decoration: BoxDecoration(  
        borderRadius: BorderRadius.circular(10.0),  
        border: Border.all(  
          color: Colors.red, style: BorderStyle.solid, width: 1.50), // Border.all  
        ), // BoxDecoration  
      ), // Container  
    ); // Container  
}
```

===== Exception caught by widgets library =====

The following assertion was thrown building MyHomePage(dirty, state: _MyHomePageState#28950):

Cannot provide both a color and a decoration

To provide both, use "decoration: BoxDecoration(color: color)".

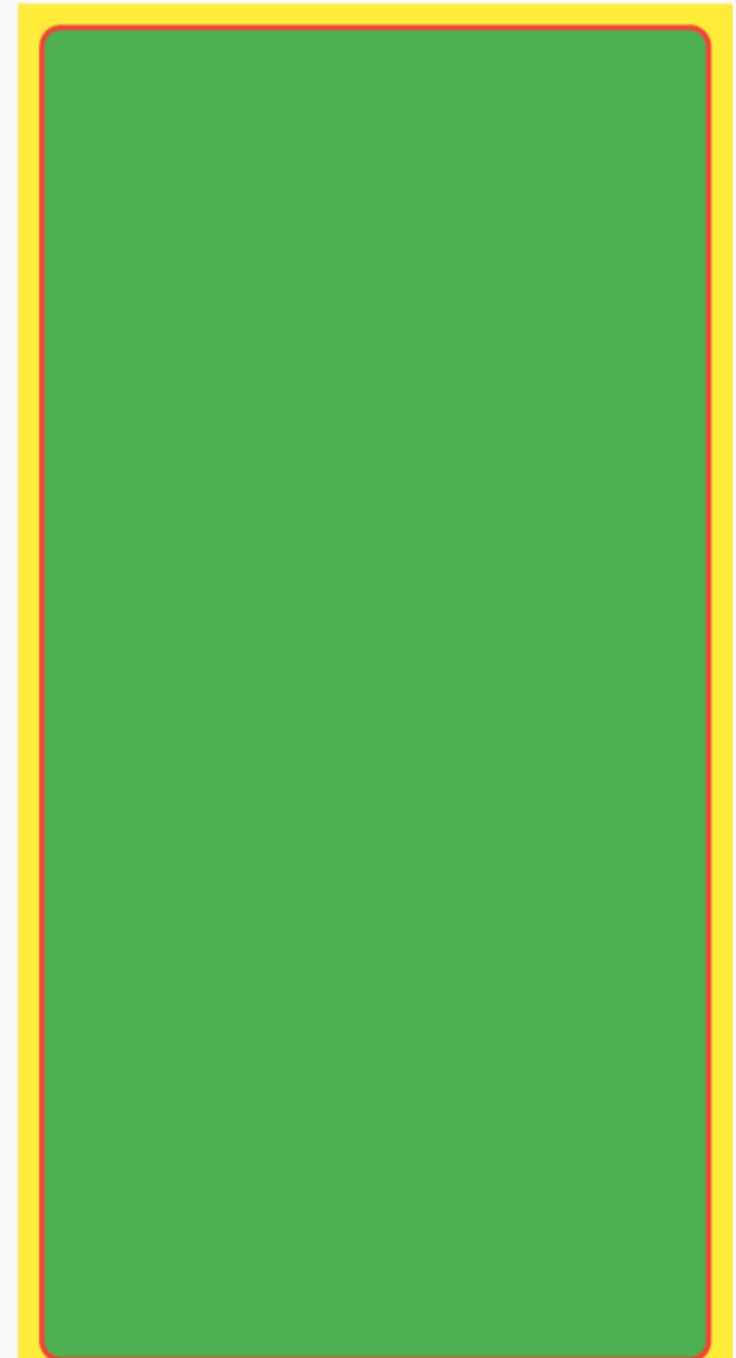
'package:flutter/src/widgets/container.dart':

Failed assertion: line 320 pos 15: 'color == null || decoration == null'



Container - BoxDecoration

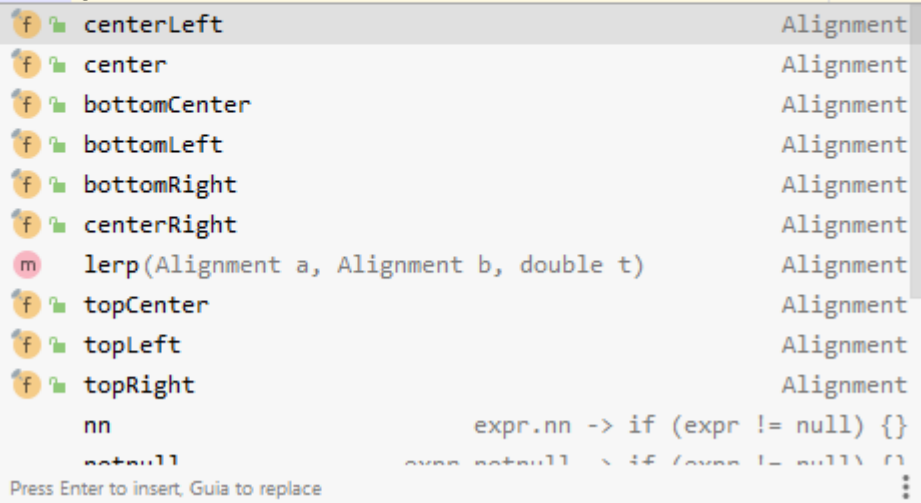
```
_body() {  
  return Container(  
    margin: EdgeInsets.only(top: 16, left: 40, right: 20),  
    padding: EdgeInsets.all(10),  
    color: Colors.yellow,  
    child: Container(  
      decoration: BoxDecoration(  
        color: Colors.green,  
        borderRadius: BorderRadius.circular(10.0),  
        border: Border.all(  
          color: Colors.red, style: BorderStyle.solid, width: 2.50), // Border.all  
        ), // BoxDecoration  
      ), // Container  
    ); // Container  
}
```



Container - alignment

- O alignment é a propriedade que alinha um widget interno dentro de seu widget externo.

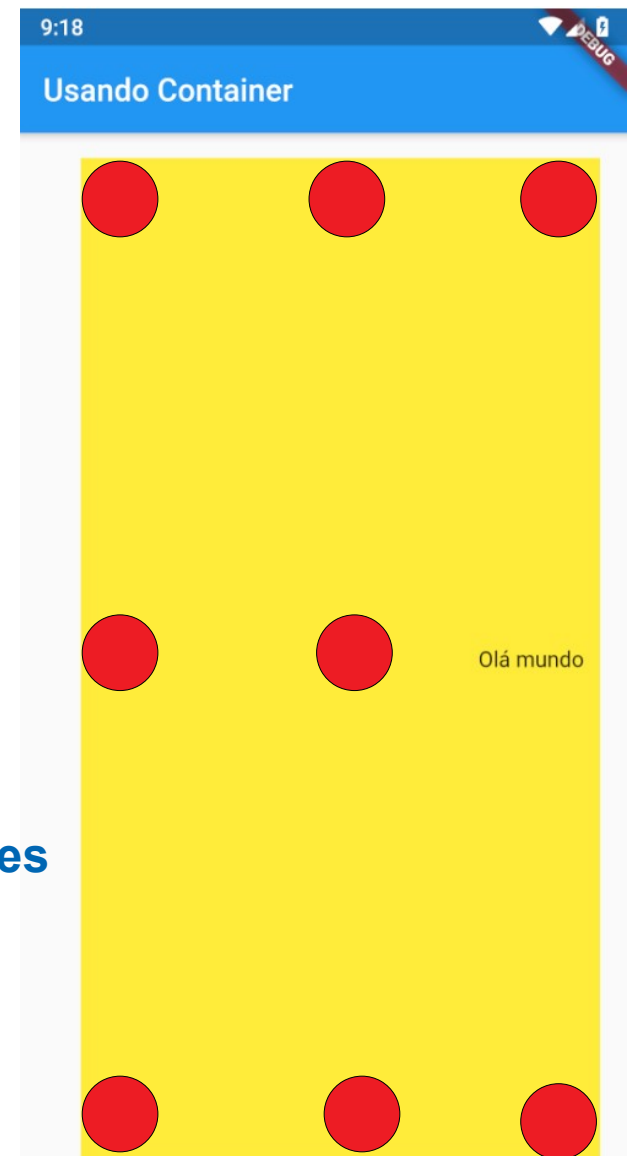
```
_body() {  
  return Container(  
    margin: EdgeInsets.only(top: 16, left: 40, right: 20),  
    padding: EdgeInsets.all(10),  
    color: Colors.yellow,  
    alignment: Alignment.,  
    child: Text("Olá")  
  );  
}
```



Container - alignment

```
_body() {  
  return Container(  
    margin: EdgeInsets.only(top: 16, left: 40, right: 20),  
    padding: EdgeInsets.all(10),  
    color: Colors.yellow,  
    alignment: Alignment.centerRight,  
    child: Text("Olá mundo")  
  ); // Container  
}
```

As bolas vermelhas indicam as outras possibilidades de alinhamento



Text

- Esse widget é responsável por exibir uma sequência de texto com um estilo único.
- O primeiro parâmetro (obrigatório) é a String que será apresentada no Text. Os demais parâmetros são opcionais e nomeados.

```
_body() {  
  return Container(  
    margin: EdgeInsets.only(top: 16, left: 40, right: 20),  
    padding: EdgeInsets.all(10),  
    color: Colors.yellow,  
    alignment: Alignment.centerRight,  
    child: Text(  
      "Olá mundo, Olá mundo, Olá mundo, Olá mundo, Olá mundo, Olá mundo, Olá mundo, Olá mundo",  
      textAlign: TextAlign.justify,  
      overflow: TextOverflow.ellipsis,  
      maxLines: 2,  
      style: TextStyle(  
        fontWeight: FontWeight.bold,  
        fontSize: 30  
      ), // TextStyle  
    ) // Text  
  ); // Container  
}
```

TextAlign.justify - alinha o texto de forma justificada

TextOverflow.ellipsis - coloca os "..." quando não há espaço suficiente para todo o texto.

maxLines: 2 - determina o número de linhas máximo para exibir o texto.

FontWeight.bold - deixa o texto em negrito

fontSize: 30 - Determina o tamanho da fonte utilizada



onTap - recebe a função que irá tratar o clique sobre o Text

TextDecoration.underline - coloca uma linha abaixo do texto

```
_body() {  
  return Container(  
    margin: EdgeInsets.only(top: 16, left: 40, right: 20),  
    padding: EdgeInsets.all(10),  
    color: Colors.lightGreenAccent,  
    alignment: Alignment.center,  
    child: InkWell(  
      onTap: () {  
        print("Ola mundo!!!");  
      },  
      child: Text(  
        "Olá mundo",  
        textAlign: TextAlign.center,  
        style: TextStyle(  
          fontSize: 22,  
          color: Colors.blue,  
          decoration: TextDecoration.underline,  
        ), // TextStyle  
      ), // Text  
    ), // InkWell  
  ); // Container  
}
```



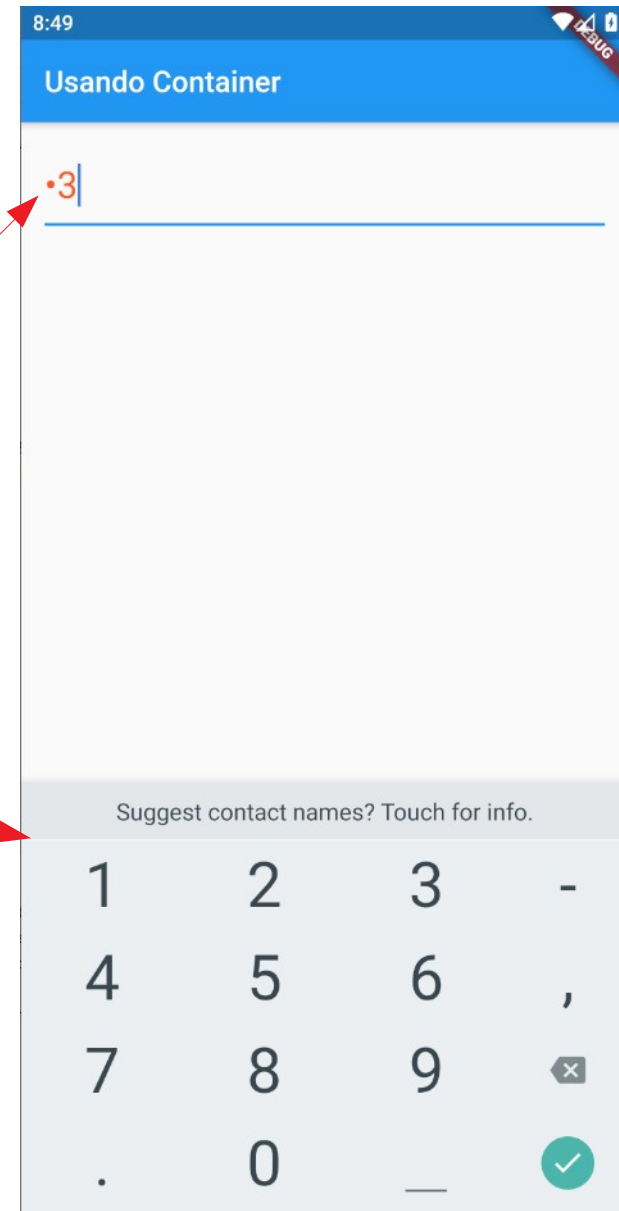
InkWell

- O InkWell possibilita deixar um Text “clicável” e executar uma ação quando clicado.

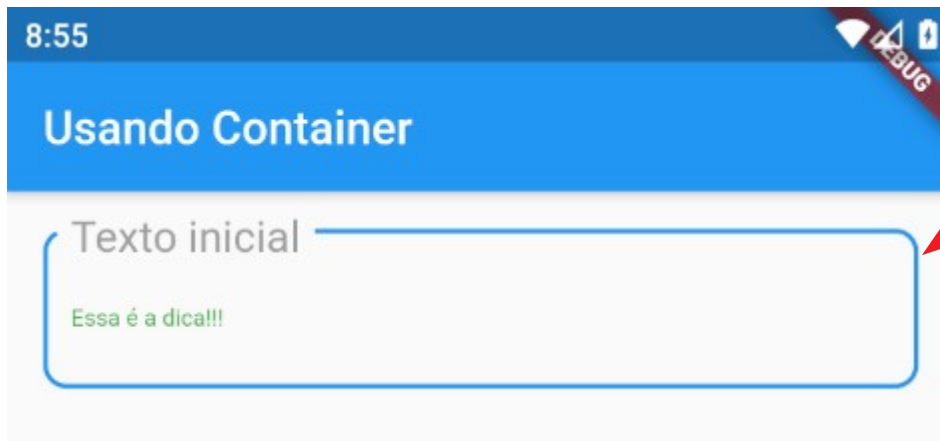
TextFormField

- É um campo de edição de texto usado em formulários.

```
_body() {  
  return Container(  
    margin: EdgeInsets.all(16),  
    child: TextFormField(  
      // Password  
      obscureText: true,  
      // Tipo do Teclado  
      keyboardType: TextInputType.number,  
      // Estilo da fonte  
      style: TextStyle(  
        fontSize: 25,  
        color: Colors.deepOrange,  
      ), // TextStyle  
    ), // TextFormField  
  ); // Container  
}
```



TextFormField



```
_body() {  
  return Container(  
    margin: EdgeInsets.all(16),  
    child: TextFormField(  
      // Estilo da fonte  
      style: TextStyle(  
        fontSize: 25,  
        color: Colors.deepOrange,  
      ), // TextStyle  
      decoration: InputDecoration(  
        border: OutlineInputBorder(  
          borderRadius: BorderRadius.circular(10),  
        ), // OutlineInputBorder  
        labelText: "Texto inicial",  
        // Estilo de labelText  
        labelStyle: TextStyle(  
          fontSize: 25,  
          color: Colors.grey,  
        ), // TextStyle  
        hintText: "Essa é a dica!!!",  
        // Estilo do hintText  
        hintStyle: TextStyle(  
          fontSize: 10,  
          color: Colors.green,  
        ), // TextStyle  
      ), // InputDecoration  
    ), // TextFormField  
  ); // Container  
}
```



RaisedButton

- A classe RaisedButton é uma das classes de botões do Flutter.

```
_body() {  
  return Container(  
    margin: EdgeInsets.all(16),  
    child: RaisedButton(  
      onPressed: () {  
        print("Clicou no botão");  
      },  
      color: Colors.yellowAccent,  
      child: Text("Clique aqui!!!"),  
    ) // RaisedButton  
  ); // Container  
}
```

```
Performing hot reload...  
Syncing files to device Samsung Galaxy S10 preview...  
Reloaded 0 of 495 libraries in 559ms.  
I/flutter ( 2512): Clicou no botão  
I/flutter ( 2512): Clicou no botão
```

9:16

Usando Container

Clique aqui!!!





RaisedButton

- Nesse exemplo usamos o widget Center (forma mais simples de centralizar outro widget na tela) e ao invés de texto colocamos um ícone no botão.

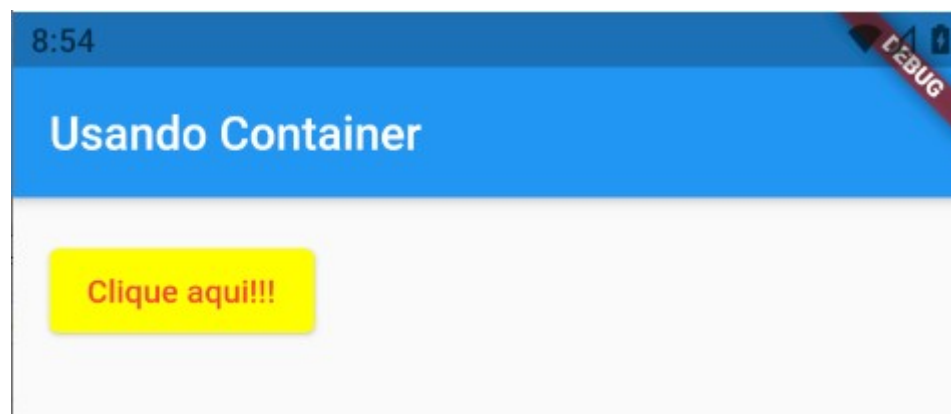
```
_body() {  
  return Center(  
    child: RaisedButton(  
      onPressed: () {  
        print("Clicou no botão");  
      },  
      color: Colors.yellowAccent,  
      child: Icon(Icons.all_inclusive),  
    ), // RaisedButton  
  ); // Center  
}
```



ElevatedButton

- Com as mudanças ainda frequentes no Flutter, a classe RaisedButton passou a ser considerada obsoleta. Em seu lugar precisaremos utilizar a classe ElevatedButton.

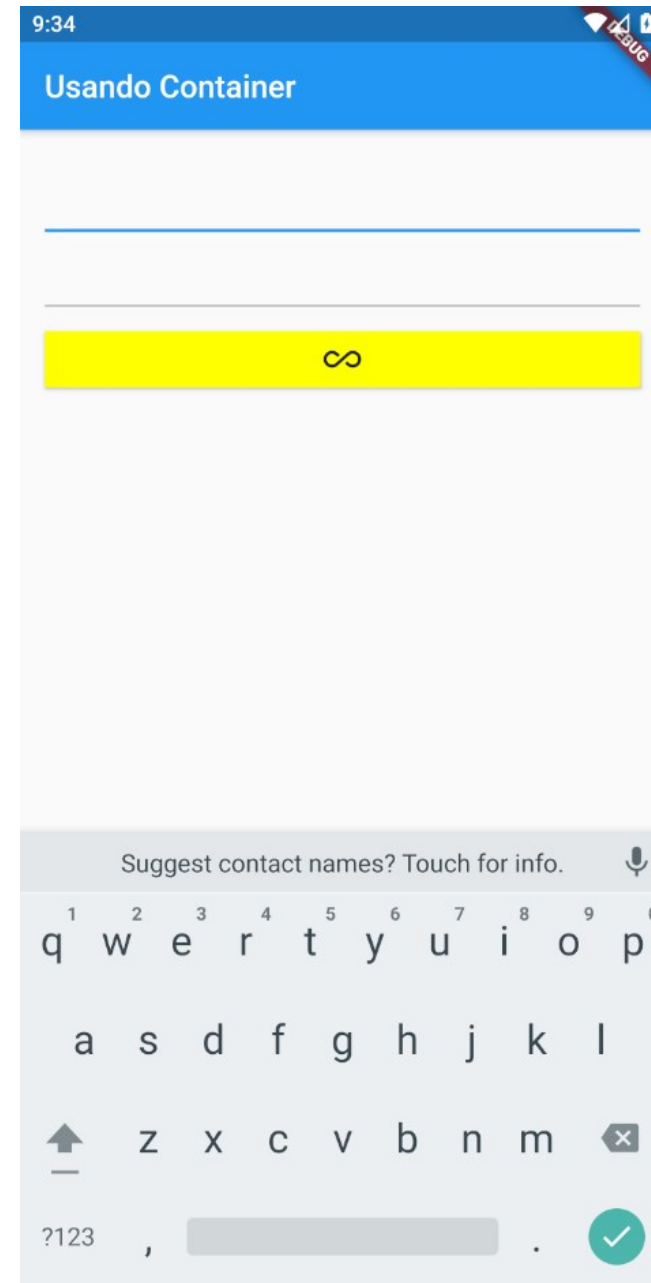
```
_body() {  
  return Container(  
    margin: EdgeInsets.all(16),  
    child: ElevatedButton(  
      onPressed: () {  
        print("Clicou no botão");  
      },  
      style: ElevatedButton.styleFrom(  
        primary: Colors.yellowAccent, // background  
        onPrimary: Colors.red, // foreground  
      ),  
      child: Text("Clique aqui!!!"),  
    ) // ElevatedButton  
  ); // Container  
}
```



ListView – uso básico

- O ListView é um widget que pode receber uma lista de outros widgets e os exibe em formato de listagem.

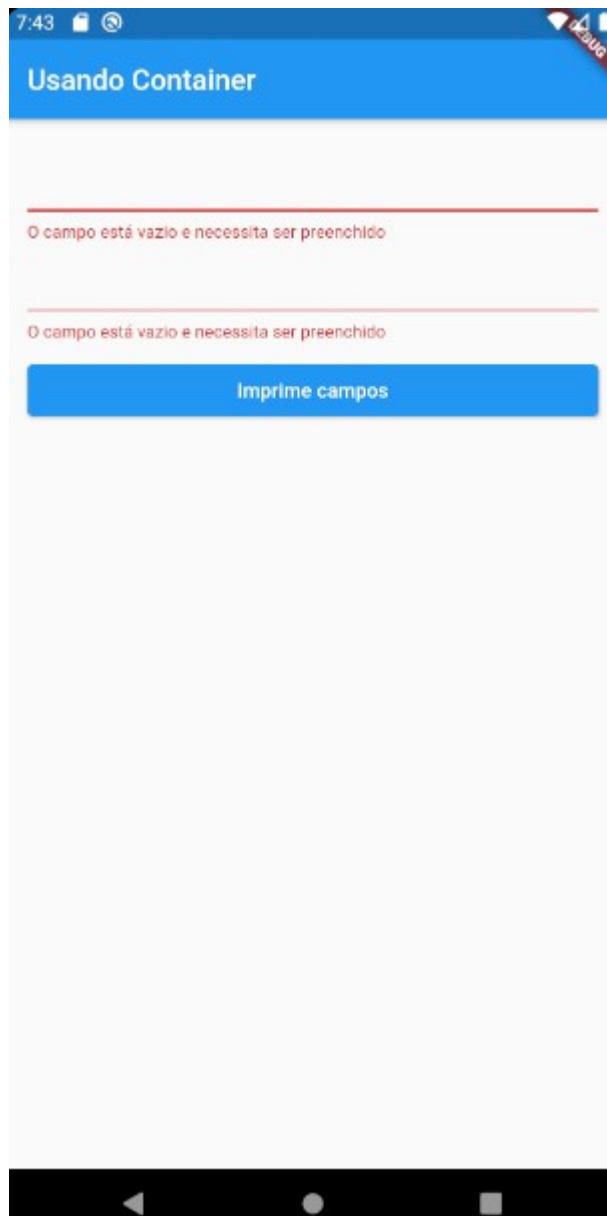
```
_body() {  
  return Container(  
    margin: EdgeInsets.all(16),  
    child: ListView(  
      children: [  
        TextFormField(),  
        TextFormField(),  
        SizedBox(height: 10),  
        ElevatedButton(  
          onPressed: () {  
            print("Clicou no botão");  
          },  
          style: ElevatedButton.styleFrom(  
            primary: Colors.yellow,  
            onPressed: Colors.black,  
          ),  
          child: Icon(Icons.all_inclusive)  
        ), // ElevatedButton  
      ],  
    ), // ListView  
  ); // Container  
}
```



Form

- Widget usado para trabalharmos com formulários.
- É possível validar os TextFormFields que estão no formulário através do método **validate()**.
- Cada TextFormField precisará definir uma função de validação para o atributo **validator**. O **validate()** varrerá pelos TextFormField do formulário acionando a função definida no atributo **validator** de cada um deles.
- Outra questão importante é que precisaremos recuperar os valores digitados nos TextFormFields. Isso pode ser feito através dos controladores colocados no atributo **controller** de cada um dos TextFormField.





```
final formkey = GlobalKey<FormState>();
final controlador1 = TextEditingController();
final controlador2 = TextEditingController();

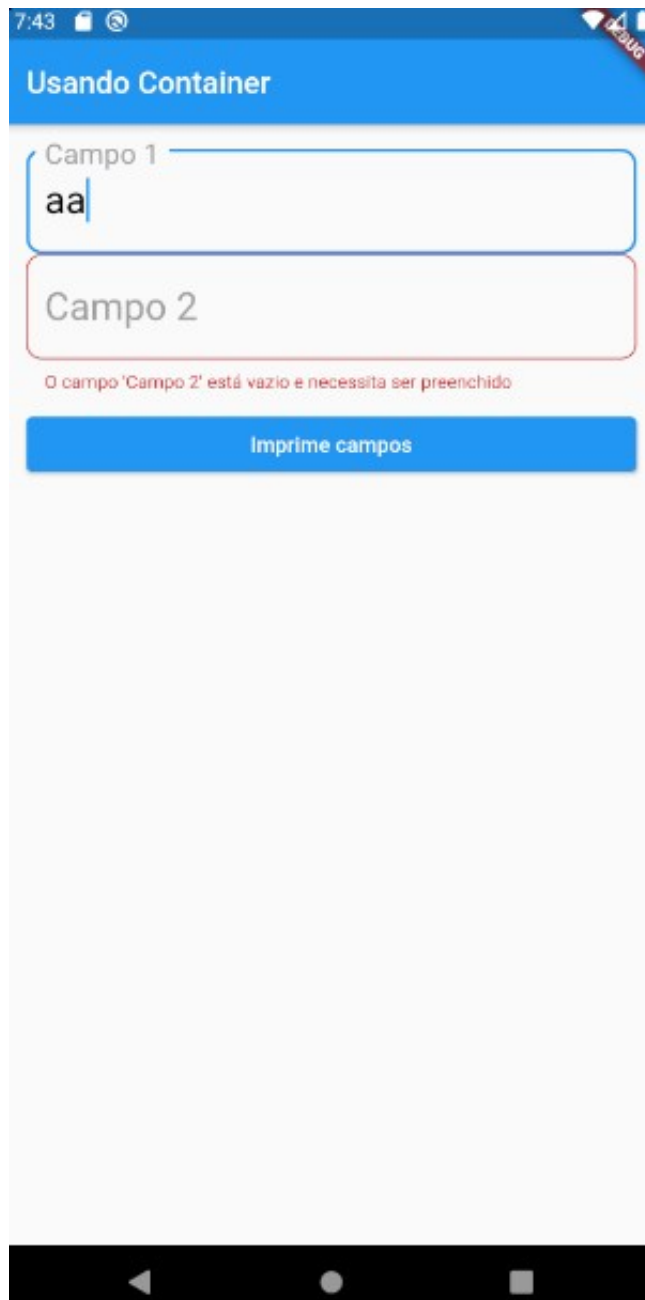
body() {
  return Form(
    key: formkey,
    child: Container(
      margin: EdgeInsets.all(16),
      child: ListView(
        children: [
          TextFormField(
            validator: (String texto){
              if(texto.isEmpty)
                return "O campo está vazio e necessita ser preenchido";
              return null;
            },
            controller: controlador1,
          ), // TextFormField
          TextFormField(
            validator: (String texto){
              if(texto.isEmpty)
                return "O campo está vazio e necessita ser preenchido";
              return null;
            },
            controller: controlador2,
          ), // TextFormField
          SizedBox(
            height: 10
          ), // SizedBox
          ElevatedButton(
            onPressed: (){
              if(formkey.currentState.validate()){
                print("O valor do campo1 é ${controlador1.text} e "
                  "o valor do campo 2 é ${controlador2.text}");
              }
            },
            child: Text("Imprime campos")
          ) // ElevatedButton
        ],
      ), // ListView
    ), // Container
  ); // Form
}
```



Criando nosso próprio TextField

- No exemplo anterior foi possível ver que utilizamos a mesma função de validação para os campos.
- É possível e provável que desejemos usar o mesmo comportamento de validação em diversos contextos.
- A criação do nosso próprio campo de edição pode simplificar essa questão.
- Vejamos o mesmo exemplo usando um campo de edição personalizado (posteriormente o código desse campo será mostrado).





```
final formkey = GlobalKey<FormState>();
final controlador1 = TextEditingController();
final controlador2 = TextEditingController();

_body() {
  return Form(
    key: formkey,
    child: Container(
      margin: EdgeInsets.all(16),
      child: ListView(
        children: [
          CampoEdicao(
            "Campo 1",
            controlador: controlador1,
          ), // CampoEdicao
          CampoEdicao(
            "Campo 2",
            controlador: controlador2,
          ), // CampoEdicao
          SizedBox(
            height: 10
          ), // SizedBox
          ElevatedButton(
            onPressed: (){
              if(formkey.currentState.validate()){
                print("O valor do campo1 é ${controlador1.text} e "
                  " o valor do campo 2 é ${controlador2.text}");
              }
            },
            child: Text("Imprime campos")
          ) // ElevatedButton
        ],
      ), // ListView
    ), // Container
  ); // Form
}
```



Widget customizado

- Além de não termos que implementar a validação para cada TextFormField aproveitamos para deixar o design do widget já customizado.
- A criação de widgets customizados acelera muito o processo de desenvolvimento pois o reaproveitamento pode ser bem maior.
- No widget customizado aproveitou-se para acrescentar a questão da mudança de foco (o exemplo será modificado para explicá-la).



```

import 'package:flutter/material.dart';

class CampoEdicao extends StatelessWidget {
  String texto_label;
  String texto_dica;
  bool password;
  TextEditingController controlador;
  FormFieldValidator<String> validador;
  TextInputType teclado;
  FocusNode marcador_foco;
  FocusNode recebedor_foco;

  CampoEdicao(
    this.texto_label,
    {this.texto_dica = "",
      this.password = false,
      this.controlador = null,
      this.validador = null,
      this.teclado = TextInputType.text,
      this.marcador_foco = null,
      this.recebedor_foco = null}){
    if(this.validador == null){
      this.validador = (String text){
        if(text.isEmpty)
          return "O campo '$texto_label' está vazio e necessita ser preenchido";
        return null;
      };
    }
  }
}

```

```

@override
Widget build(BuildContext context) {
  return TextFormField(
    validator: validador,
    obscureText: password,
    controller: controlador,
    keyboardType: teclado,
    textInputAction: TextInputAction.next,
    focusNode: marcador_foco,
    onFieldSubmitted:(String text){
      FocusScope.of(context).requestFocus(recebedor_foco);
    },
    // Estilo da fonte
    style: TextStyle(
      fontSize: 25,
      color: Colors.black,
    ), // TextStyle
    decoration: InputDecoration(
      border: OutlineInputBorder(
        borderRadius: BorderRadius.circular(10),
      ), // OutlineInputBorder
      labelText: texto_label,
      // Estilo de labelText
      labelStyle: TextStyle(
        fontSize: 25,
        color: Colors.grey,
      ), // TextStyle
      hintText: texto_dica,
      // Estilo do hintText
      hintStyle: TextStyle(
        fontSize: 10,
        color: Colors.green,
      ), // TextStyle
    ), // InputDecoration
  ); // TextFormField
}

```

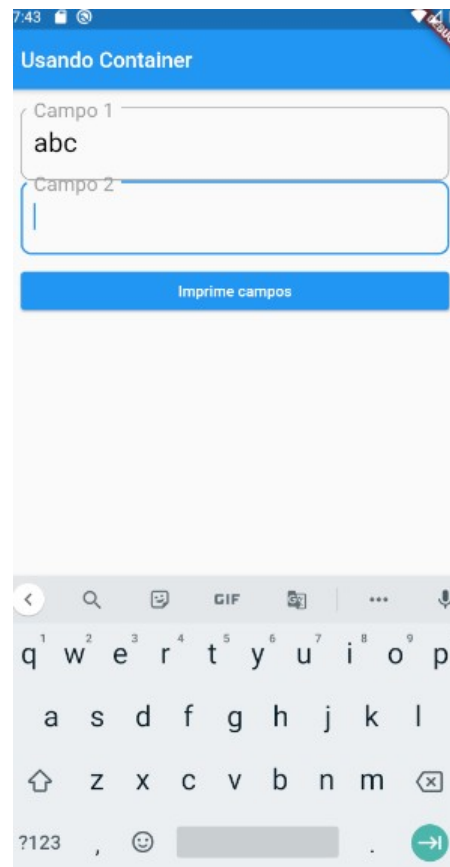
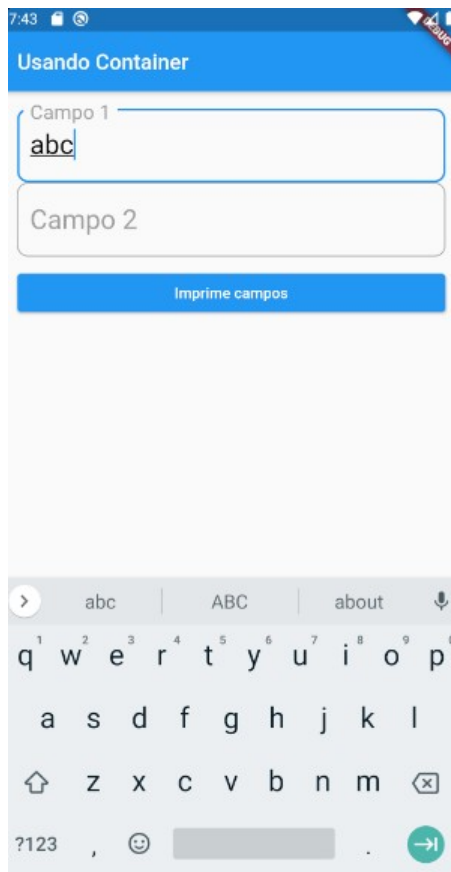


Mudança de foco

- No CampoEdicao quando a tecla “Tab” é clicada ele pode avançar para o próximo widget (a ideia é facilitar a interação via teclado virtual).
- **TextInputAction.next** define que o “Tab” avançará para o próximo widget.
- **focusNode** define o foco desse widget.
- **onFieldSubmitted** terá o objeto que receberá o foco.

```
textInputAction: TextInputAction.next,  
focusNode: marcador_foco,  
onFieldSubmitted:(String text){  
  | FocusScope.of(context).requestFocus(recebedor_foco);  
},
```





```
final formkey = GlobalKey<FormState>();
final controlador1 = TextEditingController();
final controlador2 = TextEditingController();
FocusNode foco = FocusNode();
_body() {
  return Form(
    key: formkey,
    child: Container(
      margin: EdgeInsets.all(16),
      child: ListView(
        children: [
          CampoEdicao(
            "Campo 1",
            controlador: controlador1,
            recebedor_foco: foco,
          ), // CampoEdicao
          CampoEdicao(
            "Campo 2",
            controlador: controlador2,
            marcador_foco: foco,
          ), // CampoEdicao
          SizedBox(
            height: 10
          ), // SizedBox
          ElevatedButton(
            onPressed: (){
              if(formkey.currentState.validate()){
                print("O valor do campo1 é ${controlador1.text} e "
                  "o valor do campo 2 é ${controlador2.text}");
              }
            },
            child: Text("Imprime campos")
          ) // ElevatedButton
        ],
      ), // ListView
    ), // Container
  ); // Form
}
```



Vamos praticar ...

- Não será disponibilizado o projeto desse exemplo. Você deverá implementá-lo no Android Studio (aproveite para modificá-lo e experimentar os atributos dos widgets apresentados).
- Em aulas posteriores os projetos serão disponibilizados pois a quantidade de código será muito maior.
- Faça também seu próprio widget de Botão.



Dúvidas?

