



# FLUTTER

## FLUTTER #6

Copyright © 2024 Accenture. All rights reserved.

FLUTTER



**Flutter Documentação**

**<https://docs.flutter.dev>**

**Flutter IDE online**

**<https://flutlab.io>**

**Repositório**  
**<https://pub.dev/>**

**Navigation Drawer**  
**<https://m3.material.io/>**

**Widget of the week**  
**<https://docs.flutter.dev/ui/widgets>**



FLUTTER



# FLUTTER WIDGETS

## PARTE 1



FLUTTER





## Tudo é Widgets!

Os Widgets são componentes que carregam informações para o Flutter entender como desenhá-los na tela.

1. Widgets de informação.
2. Widgets de coleção.
3. Widgets de controle.
4. Widgets híbridos.

São os componentes visuais ou não, que iremos utilizar no nosso projeto Flutter.

Ex.: Botão, texto, barra de menus, barra de rolagem, Caixa de diálogo, etc...

FLUTTER



# Método Dispose





## O método **DISPOSE** é usado para liberar a memória alocada para variáveis.

Ele é chamado automaticamente pelo framework uma vez quando da destruição do widget.

```
@override
void dispose() {
  super.dispose();
  _searchCepController.clear();
}
```

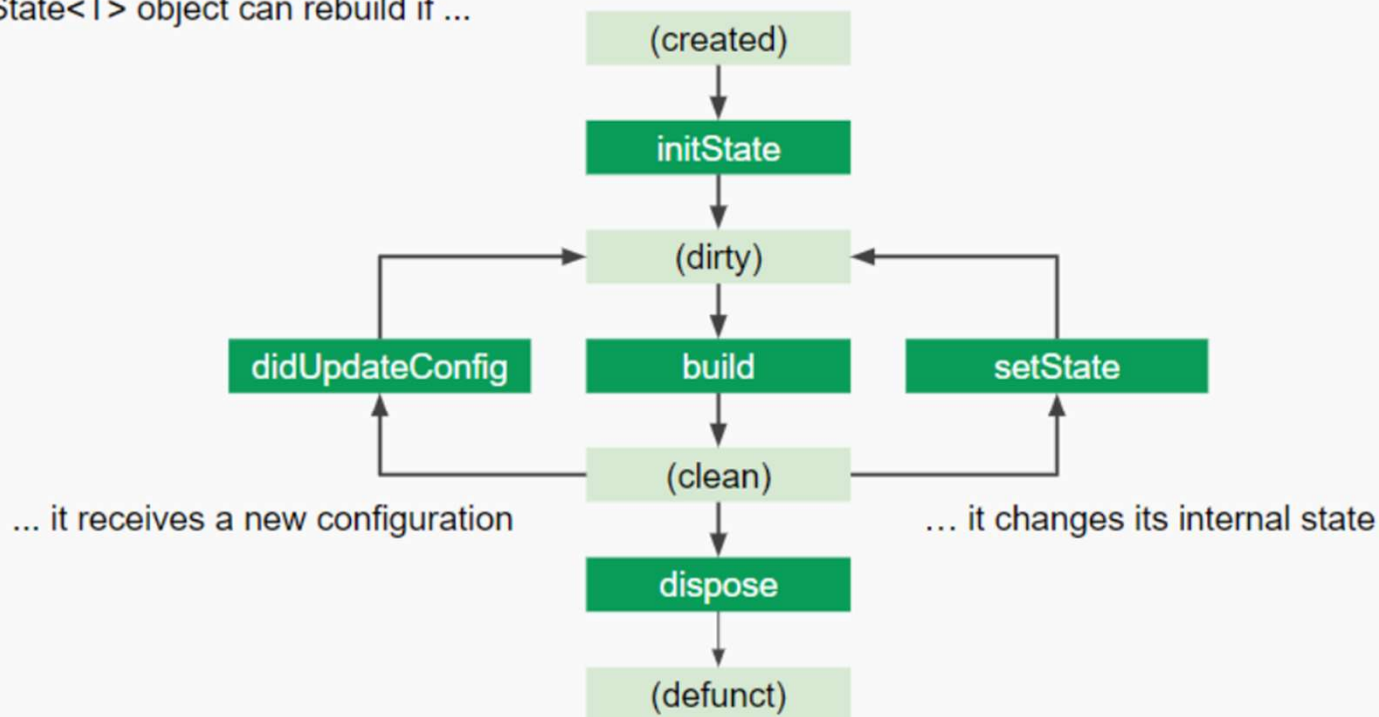




# State

Vamos analisar com calma o ciclo de vida de um objeto `State`.

A `State<T>` object can rebuild if ...





FLUTTER



# Método Build





## Todo widget possui um método **build**.

É o método responsável por renderizar o widget.

Ele é chamado automaticamente no início da execução.

Existem dois tipos de Widgets:

os **Stateless** e

os **Stateful**.

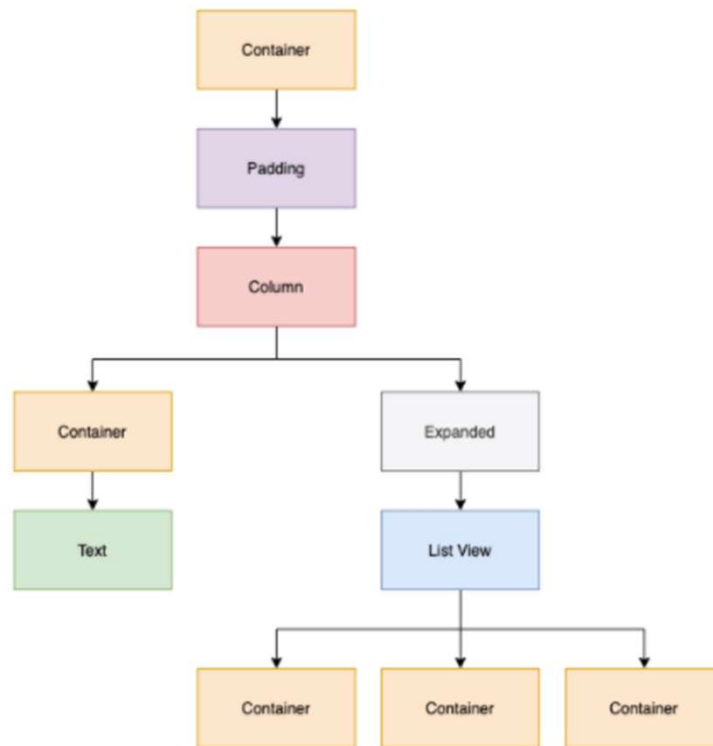
Cada um serve para um tipo de situação.

# FLUTTER



Um Widget pode receber um Child ou Childrens, que por sua vez podem também receber um Child ou Childrens.

E assim por diante, criando então uma hierarquia de elementos como na imagem abaixo



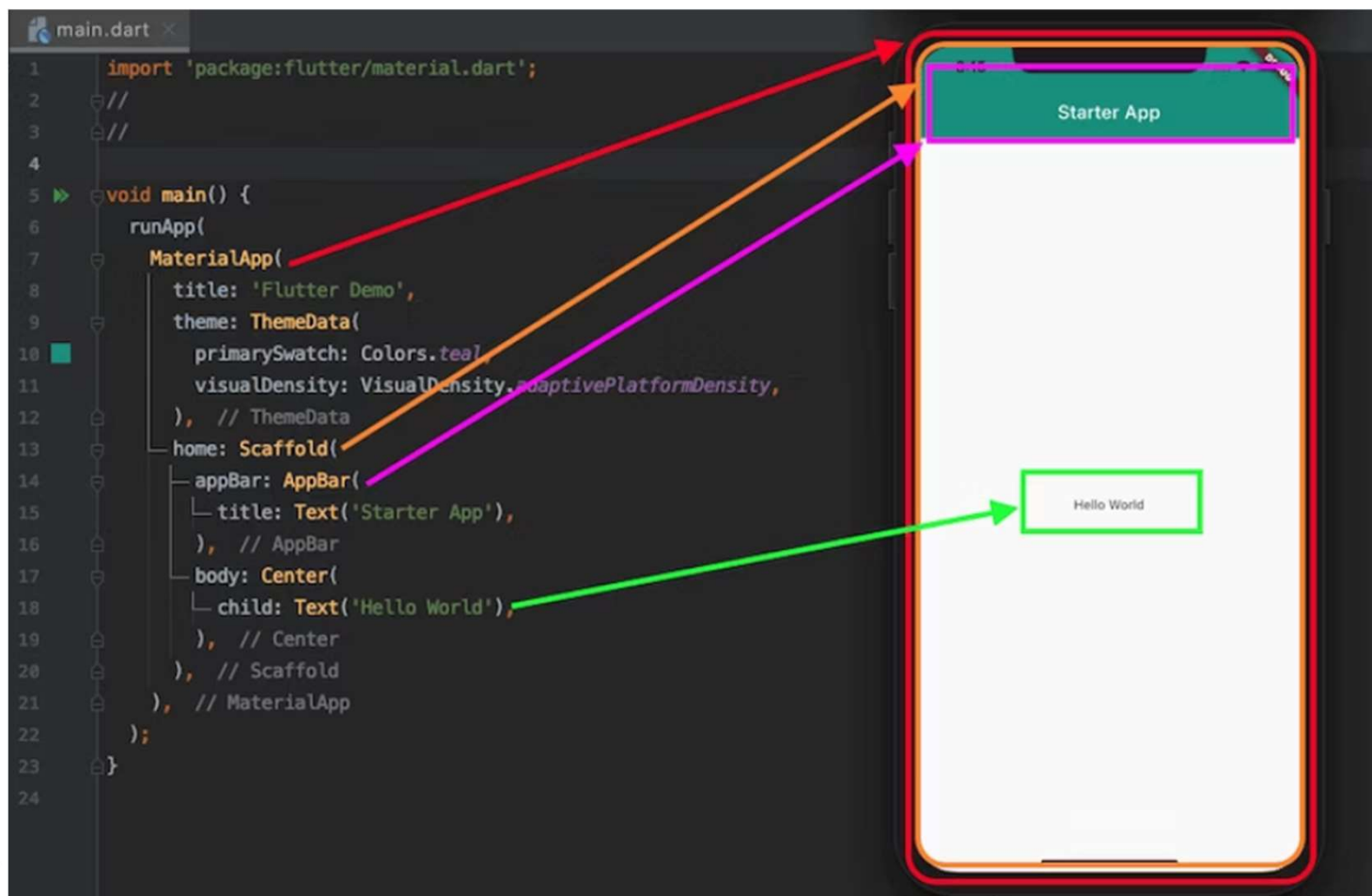
Copy

FLUTTER



# Material App

O MaterialApp é a raiz da árvore de widgets do seu aplicativo. Ele é responsável por vários aspectos importantes: Tema, homePage, navegação, localização e widgets



# FLUTTER



## Material App

Para usá-lo, precisamos criar uma nova instância dentro do método **runApp**.

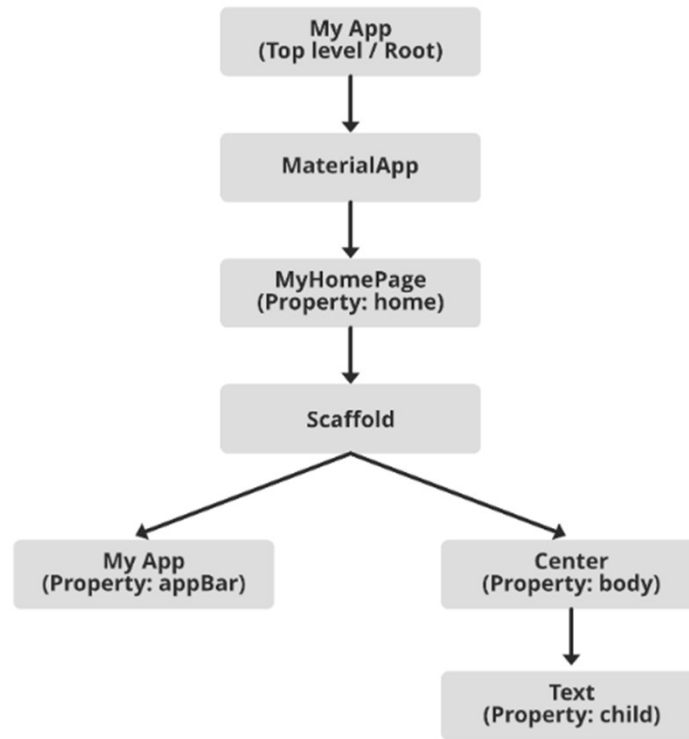
```
// o método main() é o ponto de entrada da sua aplicação
void main() {
  // chamando este método você executa sua aplicação
  runApp(
    // runApp() usa qualquer widget como um argumento.
    MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue
      ),
      home: ...
    )
  );
}
```



FLUTTER



# Hierarquia de um App



# FLUTTER



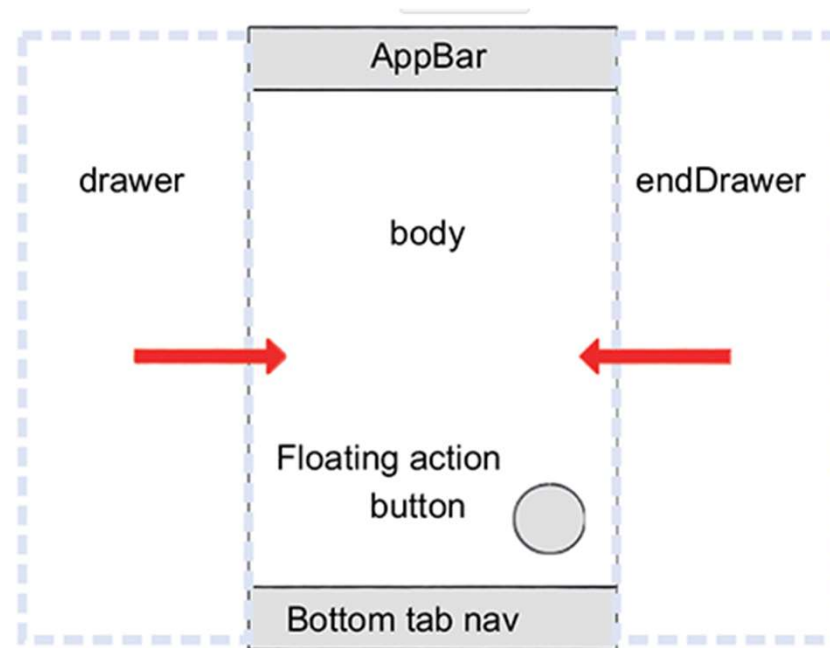
## Scaffold

O widget **Scaffold** consiste nos componentes visíveis (widgets) do aplicativo.

O Scaffold tem quatro propriedades principais:

1. AppBar
2. body.
3. Floating Action Button.
4. Bottom tab nav

Ele contém todos os **widgets** filhos e é aqui que a página inicial do app e as suas propriedades são definidas.





# FLUTTER

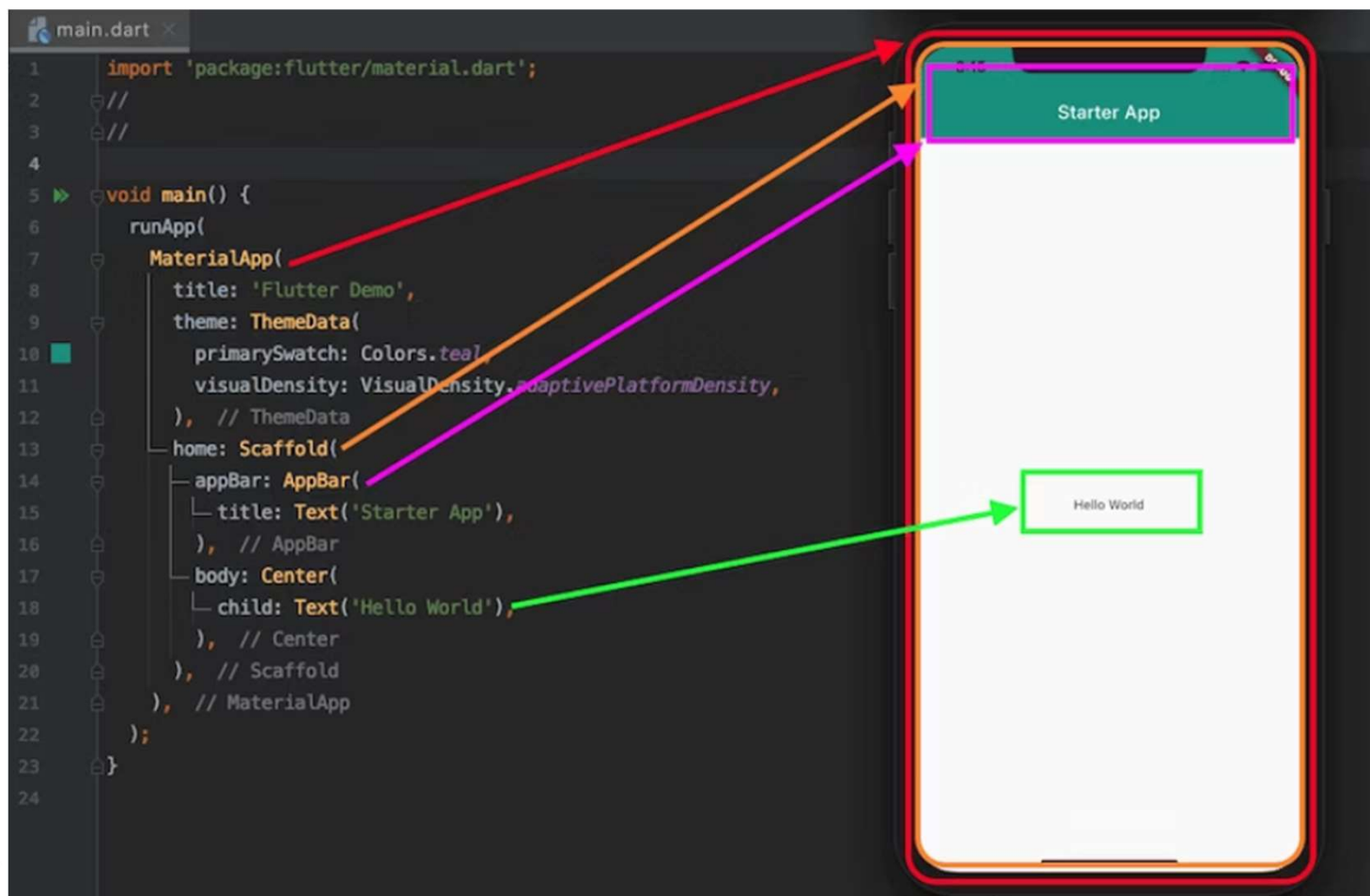


## Scaffold

O MaterialApp é um widget de conveniência que envolve vários widgets que normalmente são necessários para aplicativos.

O widget MaterialApp fornece um layout.

Para usá-lo, precisamos criar uma nova instância dentro do método `runApp`.



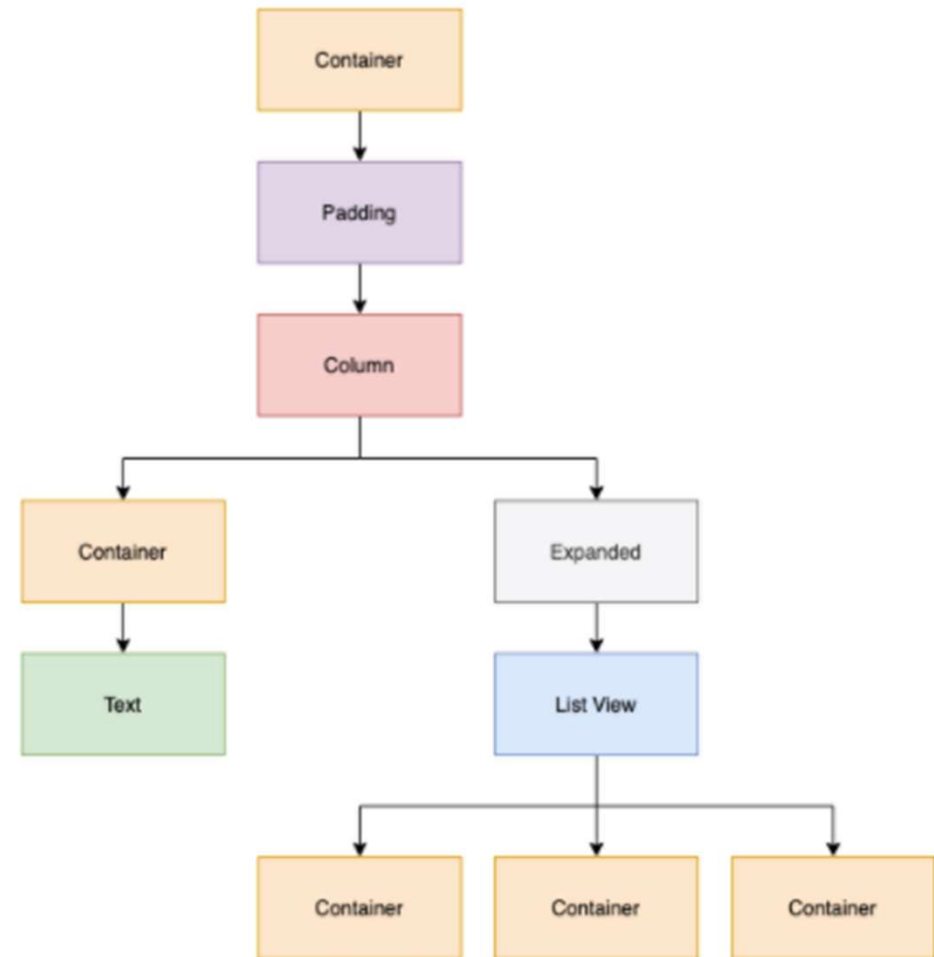


## Containers

O primeiro Widget que é interessante conhecermos é o Container.

Ele basicamente é um Widget multiuso, com o qual se pode definir tamanho, cor, borda, padding, margin e entre outras diversas características.

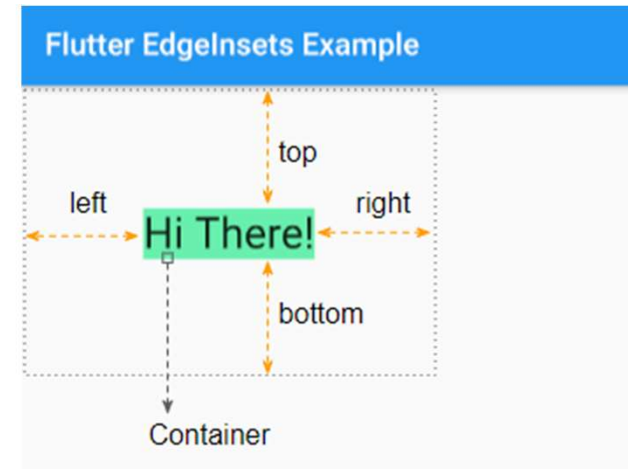
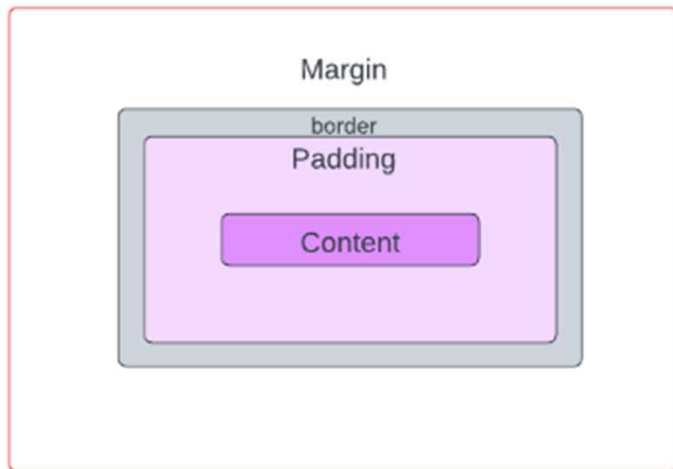
Um elemento base que precisa ser estilizado.





## Containers

### The Box Model



- `EdgeInsets.zero` : Sets a zero offset in all directions
- `EdgeInsets.all` : Sets one common offset in all directions
- `EdgeInsets.only` : Sets an offset only in the specified directions. The start, top, end, and bottom directions can be specified in any order
- `EdgeInsets.fromSTEB` : Sets an offset in all directions based on the value passed for each. The directions have to be specified in the start, top, end, and bottom order




# FLUTTER

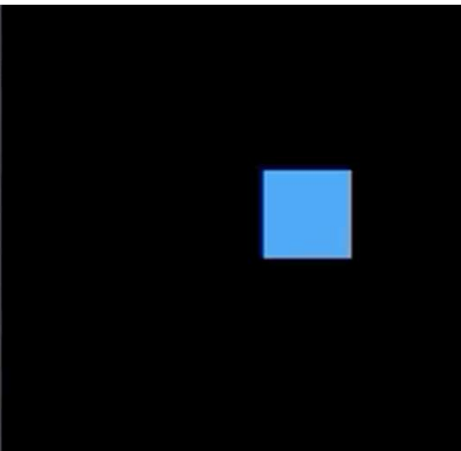


## Containers

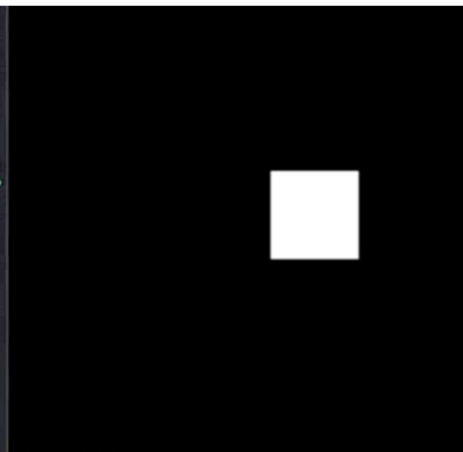
```
return Center(  
  child: Container(  
    color: Colors.white,  
    height: 60,  
    width: 50,  
  ), // Container  
); // Center  
}
```

The image shows the Flutter IDE interface. On the left, there is a code editor with the following Dart code: `return Center( child: Container( color: Colors.white, height: 60, width: 50, ), // Container ); // Center`. The code is syntax-highlighted. On the right, there is a preview window showing a white square centered on a black background. The square has a height of 60 and a width of 50. The IDE interface includes a 'Flutter Outline' pane on the left and a 'Flutter Performance' pane on the right.

```
return Center(  
  child: Container(  
    color: Colors.blue,  
    height: 50,  
    width: 50,  
  ), // Container  
); // Center  
}
```

The image shows the Flutter IDE interface. On the left, there is a code editor with the following Dart code: `return Center( child: Container( color: Colors.blue, height: 50, width: 50, ), // Container ); // Center`. The code is syntax-highlighted. On the right, there is a preview window showing a blue square centered on a black background. The square has a height of 50 and a width of 50. The IDE interface includes a 'Flutter Outline' pane on the left and a 'Flutter Performance' pane on the right.

```
return Center(  
  child: Container(  
    color: Colors.white,  
    height: 50,  
    width: 50,  
  ), // Container  
); // Center  
}
```

The image shows the Flutter IDE interface. On the left, there is a code editor with the following Dart code: `return Center( child: Container( color: Colors.white, height: 50, width: 50, ), // Container ); // Center`. The code is syntax-highlighted. On the right, there is a preview window showing a white square centered on a black background. The square has a height of 50 and a width of 50. The IDE interface includes a 'Flutter Outline' pane on the left and a 'Flutter Performance' pane on the right.



# Columns e Rows

Esses dois Widgets servem para alinharmos os elementos dentro do nosso aplicativo.

```
@override
Widget build(BuildContext context) {
  return Column(
    children: <Widget>[
      Container(width: 50, height: 50, color: Colors.red),
      Container(width: 50, height: 50, color: Colors.blue),
      Container(width: 50, height: 50, color: Colors.green),
    ], // <Widget>[]
  ); // Column
}
```



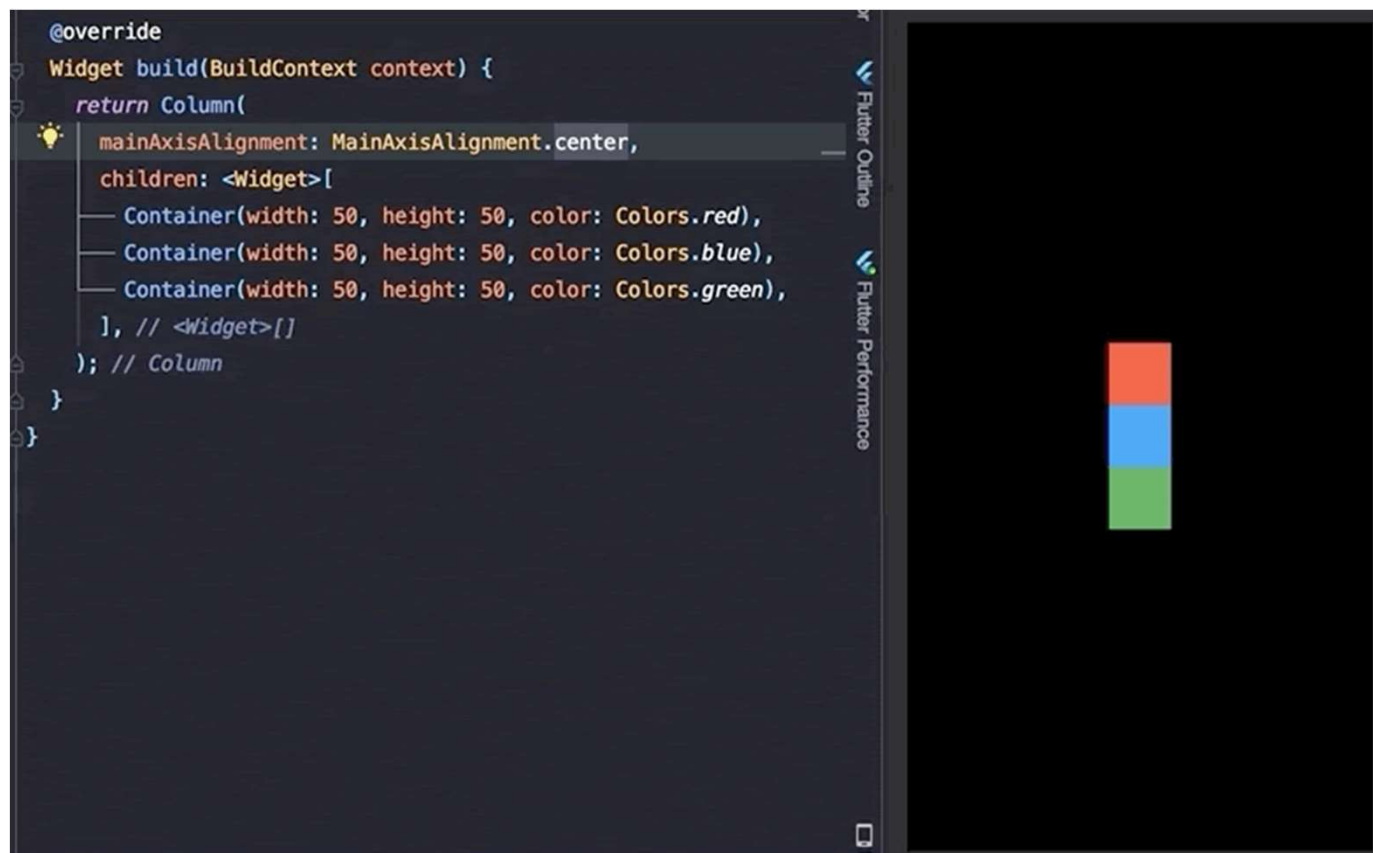
FLUTTER



# Columns e Rows

Columns e Rows são bem parecidos, porém com diferentes direções.

Eles também possuem alinhamentos como Center, Space Between, Space Around e Space Evenly.



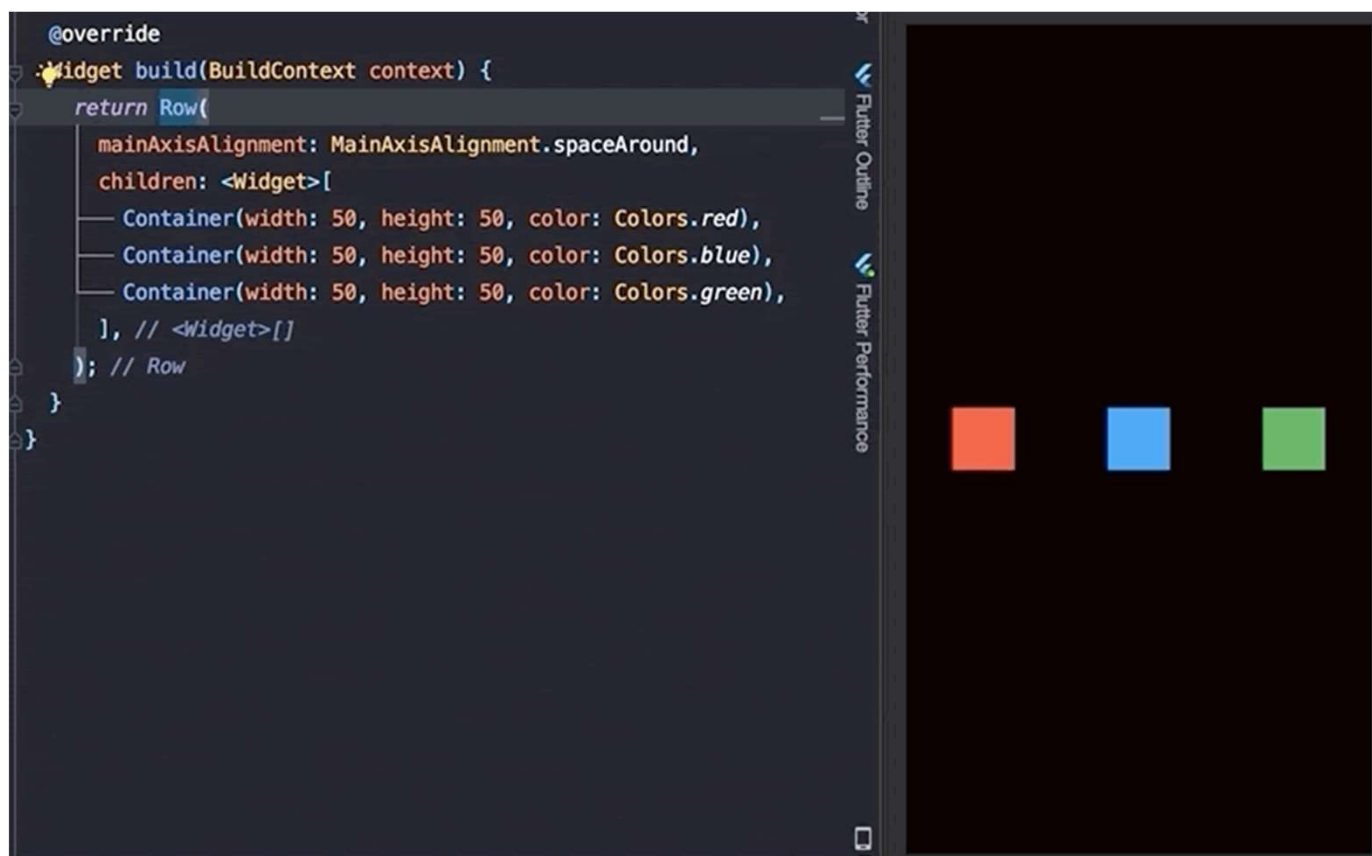


# Columns e Rows

<https://docs.flutter.dev/ui/layout>

Esses alinhamentos servem tanto para a Column quanto para o Row, o que facilita na hora de utilizar e lembrar dos nomes. Eles também possuem dois tipos alinhamentos, o Main Axis Alignment e o Cross Axis Alignment.

O **Main Axis Alignment** serve para alinhar a principal função do Widget. No caso do Column alinhamento vertical e do Row alinhamento horizontal





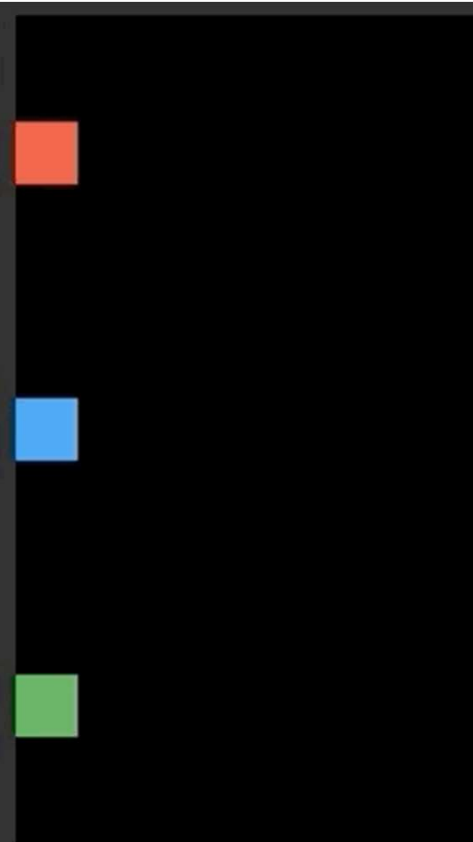
# FLUTTER



## Columns e Rows

Já o **Cross Axis Alignment** serve para alinhar a função inversa do Widget. No caso do Column alinhamento horizontal e do Row alinhamento vertical.

```
@override
Widget build(BuildContext context) {
  return Column(
    mainAxisAlignment: MainAxisAlignment.spaceAround,
    crossAxisAlignment: CrossAxisAlignment.start,
    children: <Widget>[
      Container(width: 50, height: 50, color: Colors.red),
      Container(width: 50, height: 50, color: Colors.blue),
      Container(width: 50, height: 50, color: Colors.green),
    ], // <Widget>[]
  ); // Column
}
```





# FLUTTER

## Atividade 2

Blocos de containers.

Utilize containers, rows e  
columns.

Utilize scaffold e drawer.

