

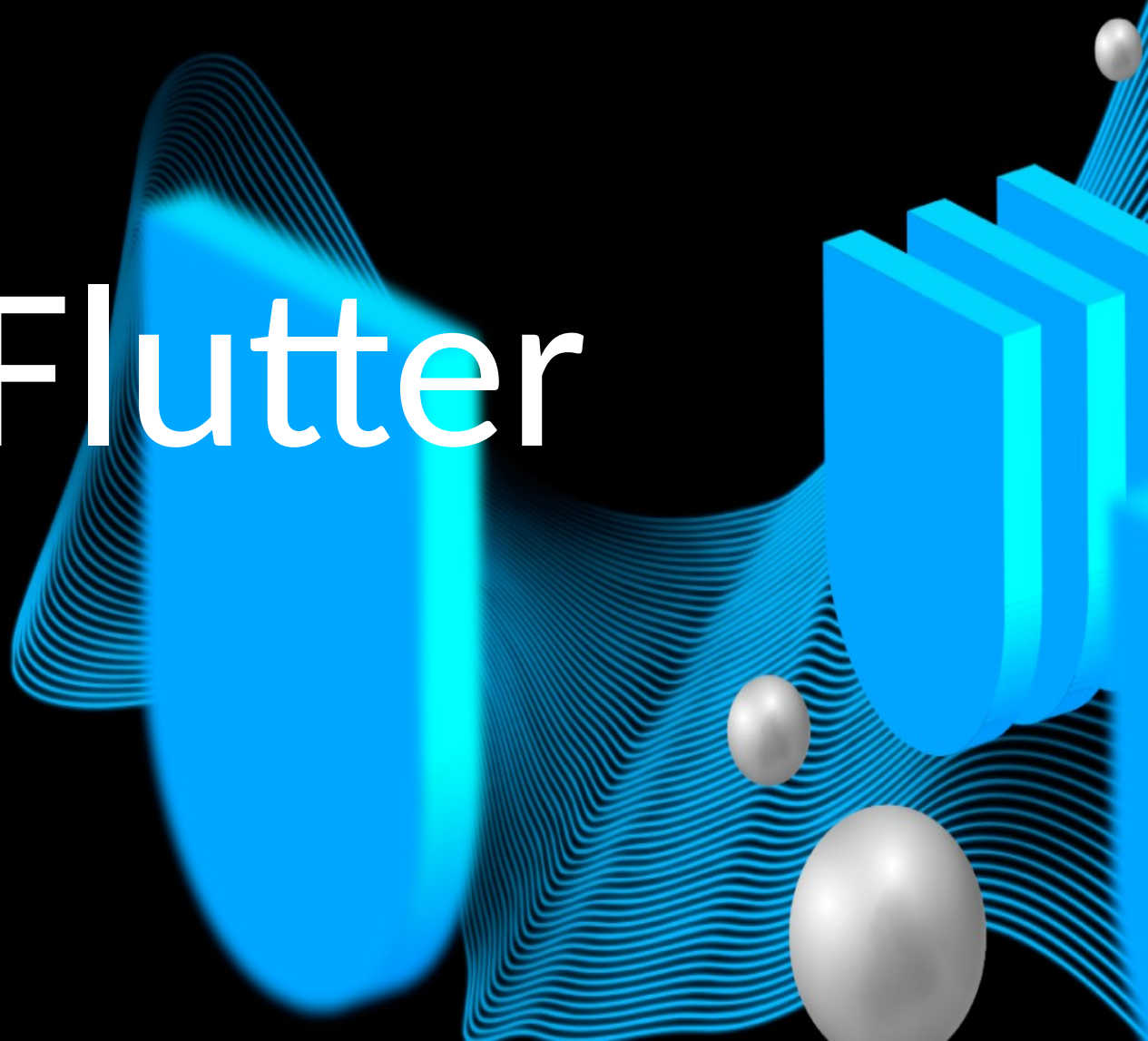


UNITAU
Universidade de Taubaté





Flutter



Hello World



Para criar uma aplicação, execute o comando:

```
flutter create <nome da aplicacao>
```

Por exemplo:

```
> flutter create hello_world
```

Isso irá criar um subdiretorio **hello_world** sob o diretório corrente com os arquivos do projeto.

Para executar o projeto, execute o emulador:

```
> flutter emulator --launch pixel
```

Mude para o diretório da aplicação:

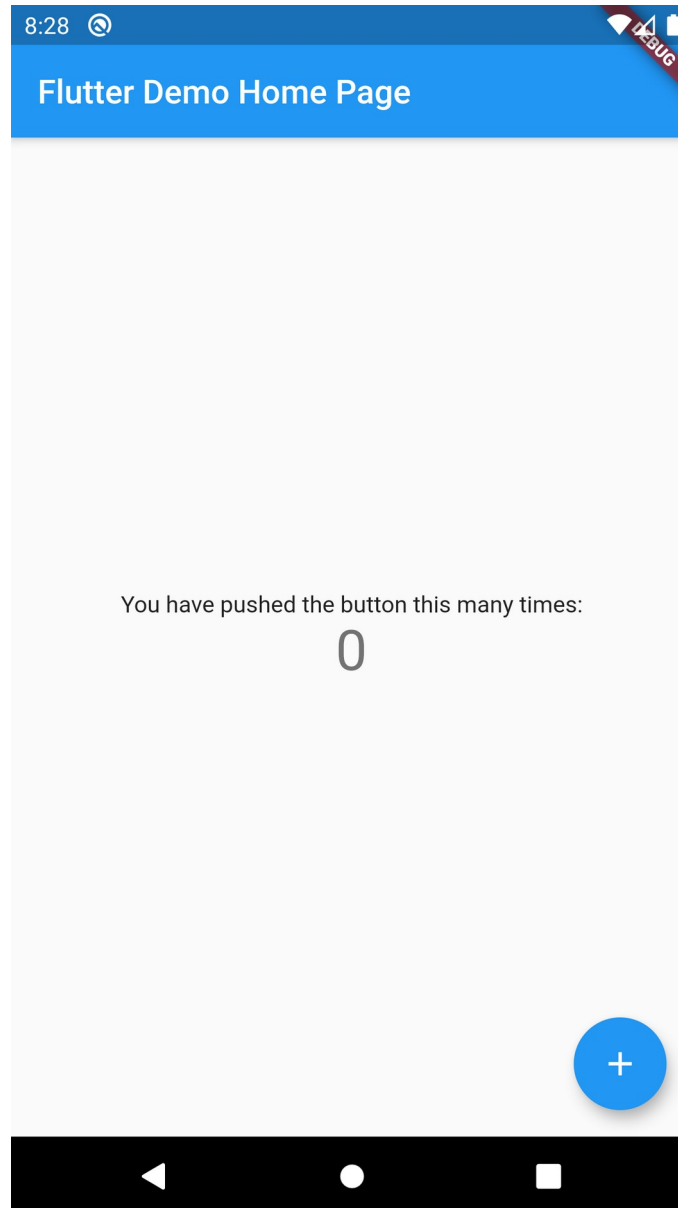
```
> cd hello_world
```

Execute a aplicação:

```
> flutter run
```

Para encerrar a aplicação, digite **q** no terminal onde está sendo executada a aplicação.

Hello World



Hello World



As aplicações do flutter iniciam a execução pela função **main** que deve iniciar o objeto da aplicação.

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

Pode ser usada a notação lambda para a função **main**.

```
void main() => runApp(MyApp());
```

Widgets



Widgets são os objetos da interface de uma aplicação.

Existem diversos widgets como **Scaffold**, **AppBar**, **RaiseButton**, **Container** e outros. O catálogo dos widgets disponíveis no flutter pode ser visto em:

[**https://flutter.dev/docs/development/ui/widgets**](https://flutter.dev/docs/development/ui/widgets)

Além do catálogo dos widgets, o site também contém vídeos instrutivos sobre Flutter, inclusive uma série de vídeos curtos com uma breve explicação sobre os widgets (**Flutter Widget of the Week**):

[**https://flutter.dev/docs/resources/videos**](https://flutter.dev/docs/resources/videos)

Os widgets do flutter seguem o design de interface do Google (Material Design) ou o design da Apple, chamado no flutter de Cupertino (**CupertinoButton**, **CupertinoDialog** e outros).

Widgets Tree



Os widgets de uma aplicação são organizados em uma árvore (**Widgets Tree**).

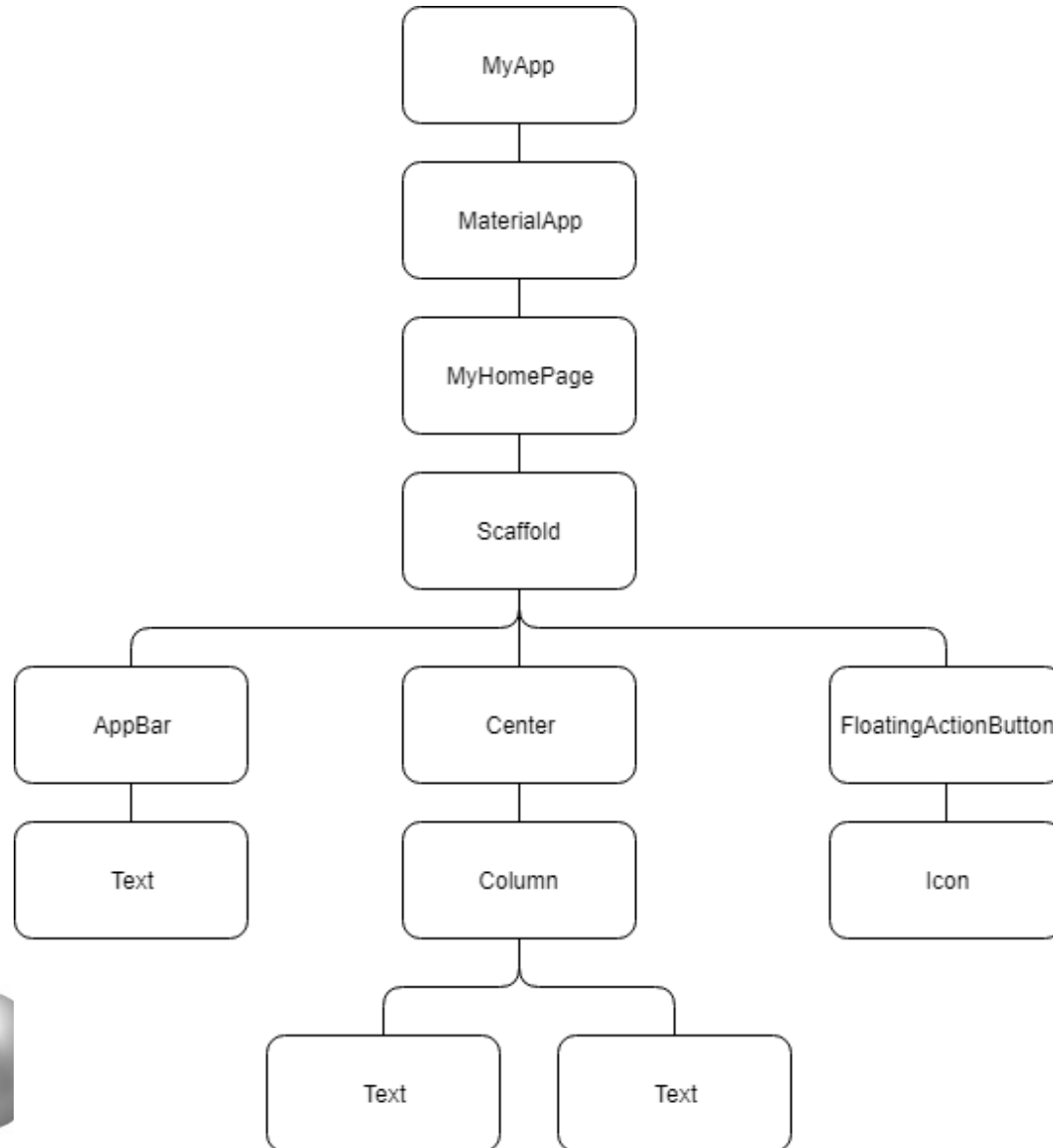
Quando um widget é criado, é chamado o método **build**. O Flutter irá recriar a subárvore de widgets abaixo desse widget com o retorno do método **build**.

Widgets Tree



```
Widget build(BuildContext context) {  
  return Scaffold(  
    appBar: AppBar( title: Text(widget.title), ),  
    body: Center(  
      child: Column(  
        mainAxisAlignment: MainAxisAlignment.center,  
        children: <Widget>[  
          Text(  
            'You have pushed the button this many times:',  
          ),  
          Text( '$_counter',  
            style: Theme.of(context).textTheme.display1, ),  
        ],  
      ),  
    ),  
    floatingActionButton: FloatingActionButton(  
      onPressed: _incrementCounter,  
      tooltip: 'Increment',  
      child: Icon(Icons.add),  
    ),  
  );  
}
```


Widgets Tree



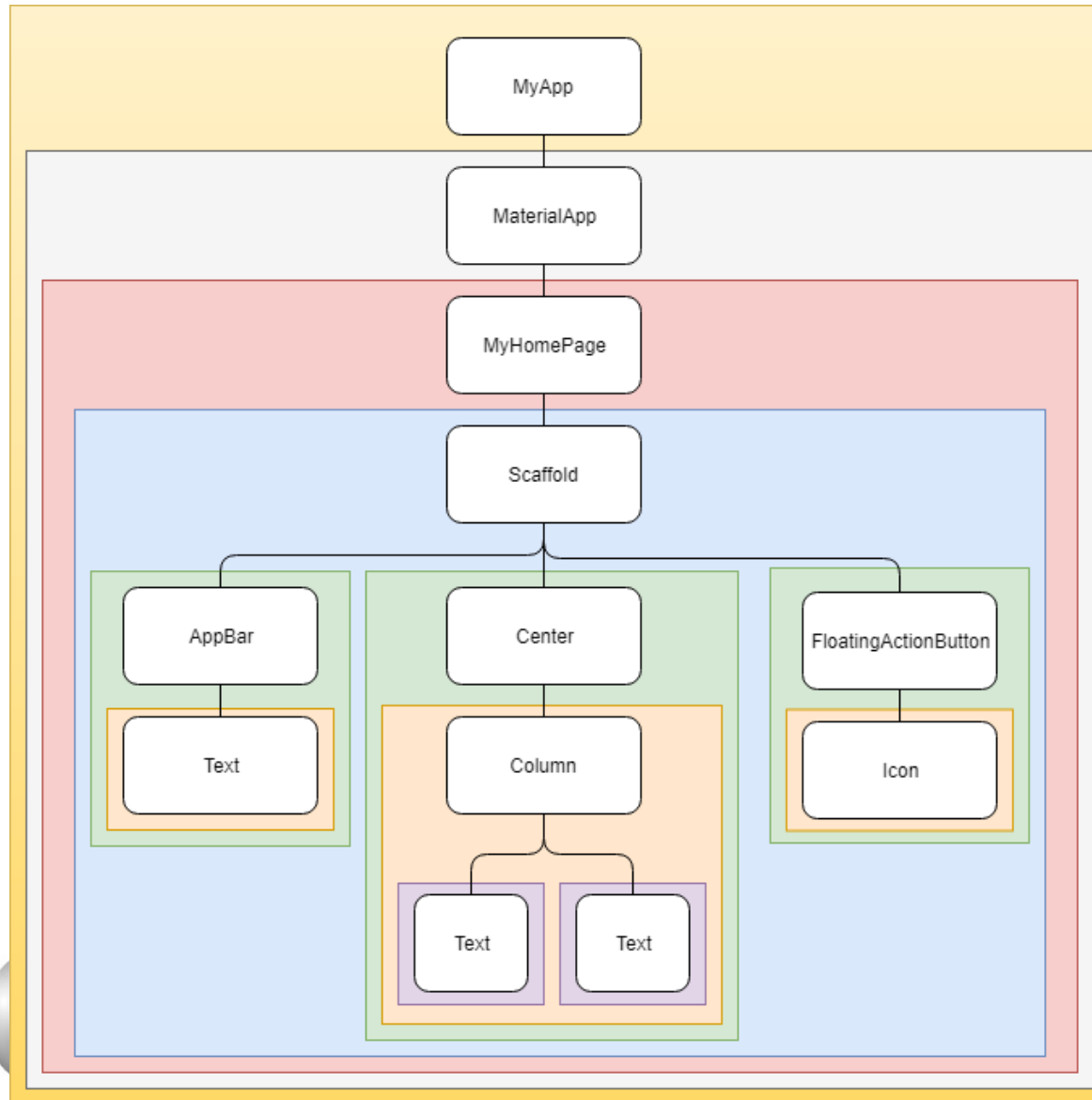
Contexto



Cada widget tem associado a ele um contexto. Esse contexto tem informações sobre a localização do widget na Widget Tree.

Contextos são encadeados compondo uma árvore de contextos. O contexto de um widget que contém outros widgets é a raiz dos contextos desses widgets.

Contexto



Estado



O conjunto de valores associados a um widget é chamado de estado.

Flutter possui widgets que não permitem a alteração de valores após a sua criação (**StatelessWidget**) e widgets que permitem a alteração de valores dinamicamente durante a execução da aplicação (**StatefulWidget**).

O termo estado é associado à **StatefulWidget**. Para **StatelessWidget**, os valores são parâmetros passados na criação do widget.

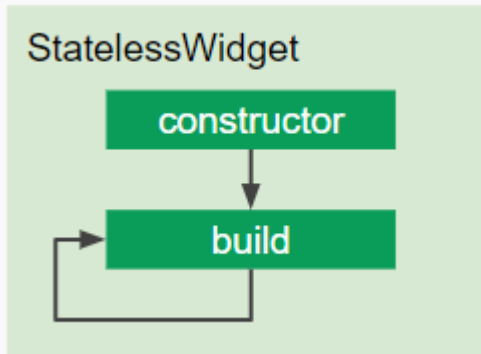
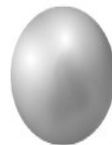
Para cada **StatefulWidget** é criado um objeto **State** que contém o estado do widget.

Stateless Widget



StatelessWidget são widgets que não se alteram depois de criados. **StatelessWidget** não são recriados baseados em eventos ou ações do usuário.

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

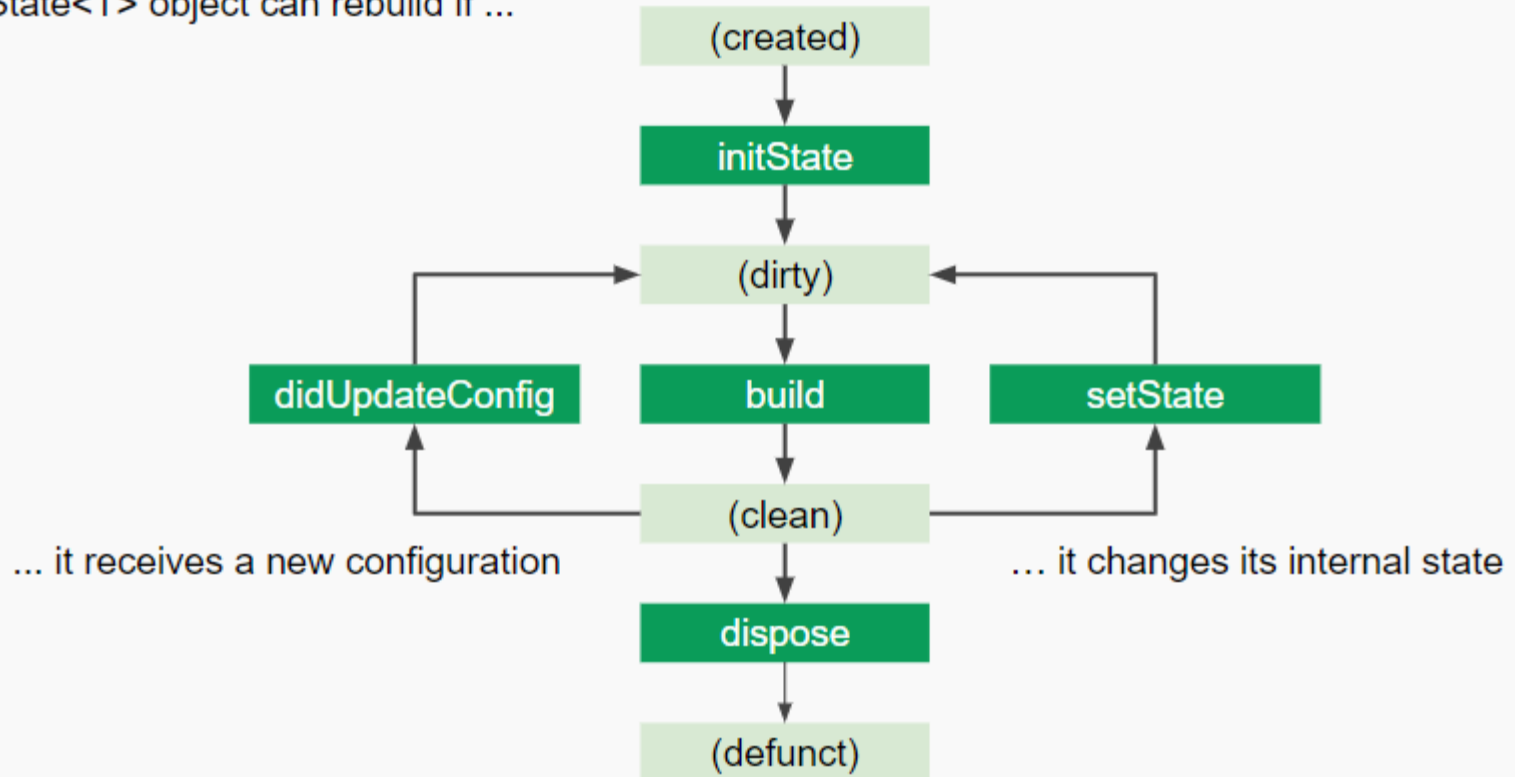


Statefull Widget



StatefulWidget são widgets que são alterados dinamicamente durante a execução da aplicação. Quando o estado do widget é alterado, o widget deve ser reconstruído.

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



Statefull Widget



StatefulWidget devem criar um objeto **State** para gerenciar as alterações no widget.

```
class MyHomePage extends StatefulWidget {  
  MyHomePage({Key key, this.title}) : super(key: key);  
  
  final String title;  
  
  @override  
  _MyHomePageState createState() => _MyHomePageState();  
}
```

Statefull Widget



O objeto **State** gerencia a alteração do estado do widget.

```
class _MyHomePageState extends State<MyHomePage> {  
  int _counter = 0;  
  
  void _incrementCounter() {  
    setState(() {  
      _counter++;  
    });  
  }  
}
```

Statefull Widget



O método **build** do objeto **State** deve retornar o widget a ser exibido na tela quando o widget for construído.

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text(widget.title), ),
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          Text(
            'You have pushed the button this many times:',
          ),
          Text(
            '$_counter',
            style: Theme.of(context).textTheme.headline4,
          ),
        ],
      ),
    ),
  ),
```


Statefull Widget



```
floatingActionButton: FloatingActionButton(  
  onPressed: _incrementCounter,  
  tooltip: 'Increment',  
  child: Icon(Icons.add),  
),  
);  
}  
}
```

Hot Reload

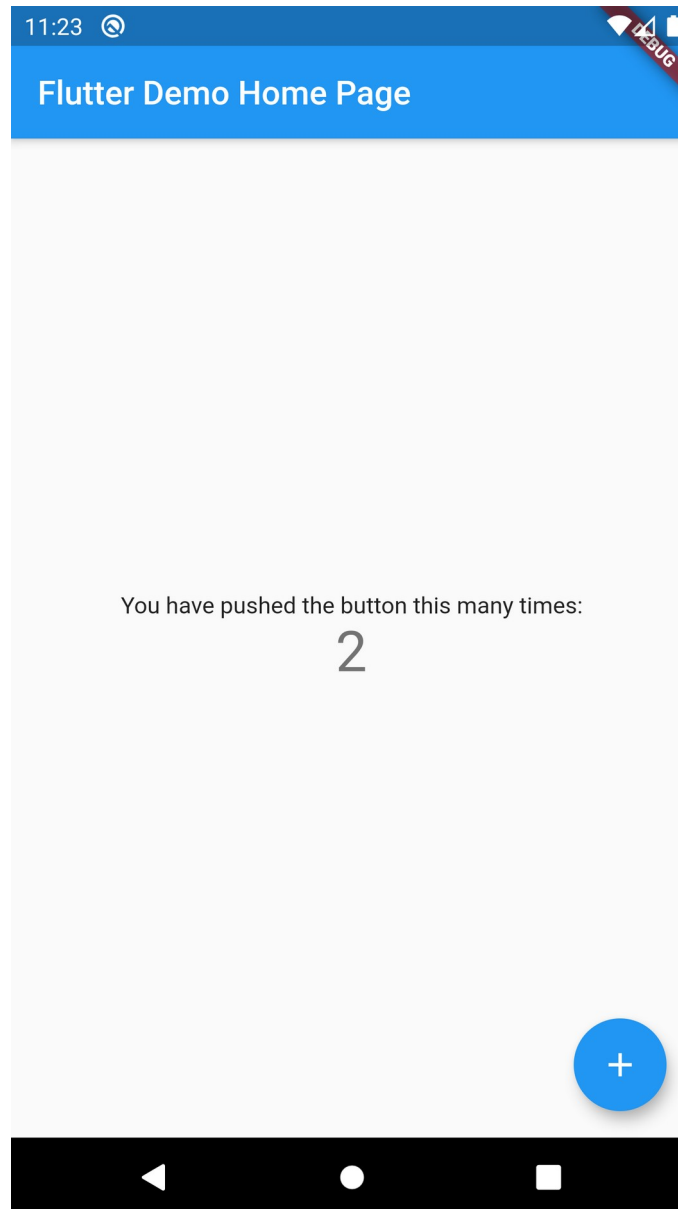


O Flutter permite que as alterações do código sejam rapidamente carregadas na aplicação que está sendo executada sem a necessidade de encerrar a aplicação e iniciar novamente.

A operação de **Hot Reload** permite carregar as alterações da aplicação sem reinicializar os estados dos elementos da aplicação.

Inicie a aplicação e clique no botão para incrementar a contagem algumas vezes.

Hot Reload



Hot Reload



Na classe **MyApp**, acrescente a linha:

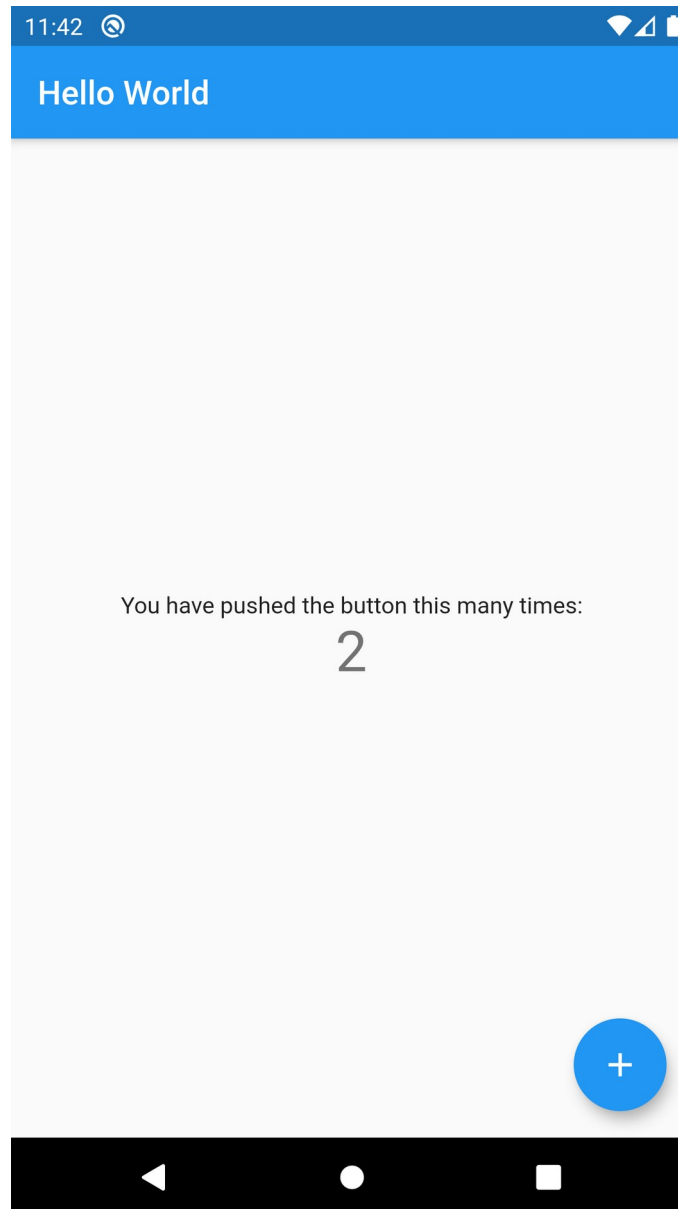
```
return MaterialApp(  
  debugShowCheckedModeBanner: false,  
  title: 'Flutter Demo',
```

E altere o título da home page da aplicação.

```
home: MyHomePage(title: 'Hello World'),
```

Execute o Hot Reload digitando **r** no terminal onde está sendo executada a aplicação. O emulador irá refletir a alteração sem reiniciar os estados da aplicação.

Hot Reload



Hot Restart



A operação de **Hot Restart** permite carregar as alterações, reiniciando os estados dos elementos da aplicação.

Altere a linha

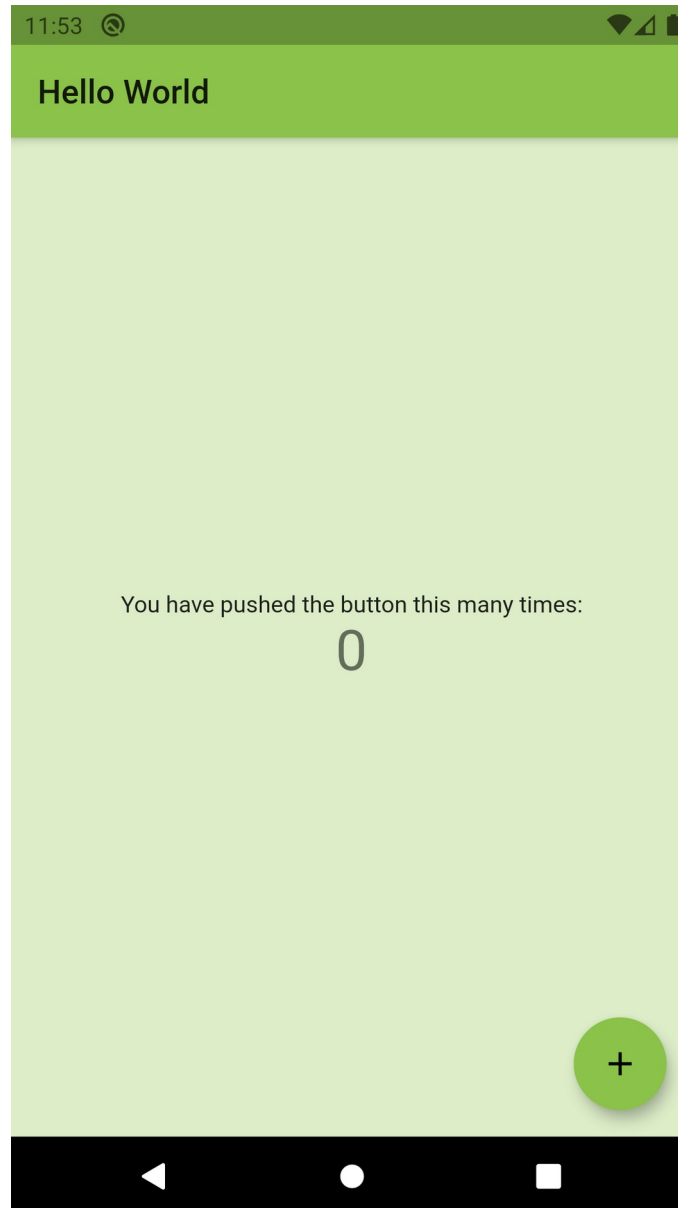
```
primarySwatch: Colors.blue,
```

Para

```
primarySwatch: Colors.lightGreen,  
canvasColor: Colors.lightGreen.shade100,
```

Execute o Hot Restart digitando R no terminal onde está sendo executada a aplicação. O emulador irá refletir a alteração, reiniciando os estados da aplicação.

Hot Restart



Estrutura de Diretórios



Ao criar uma nova aplicação, é criada uma estrutura de diretórios padrão:

raiz do projeto – contém arquivos de configuração do projeto

- **android** – contém arquivos e código específicos para plataforma Android

- **build** – contém arquivos criados na construção do projeto

- **ios** - contém arquivos e código específicos para plataforma iOS

- **lib** – contém o código do projeto

- **test** – contém código da unidade de testes

É recomendado seguir esse padrão para utilização desses diretórios. Podemos criar outros diretórios e subdiretórios para organizar o projeto.

Estrutura de Diretórios



Para criar um novo diretório podemos criar o diretório diretamente pelo sistema operacional.

Uma sugestão de diretórios a utilizar pode ser:

- **assets** – para armazenar arquivos de recursos diversos como imagens, fontes, ícones e outros arquivos diversos
- **assets/images** – para armazenar as imagens
- **lib/pages** – para armazenar as classes das páginas da aplicação
- **lib/models** – para armazenar as classes de modelo de dados da aplicação
- **lib/utils** – para armazenar as classes com funções diversas
- **lib/services** – para armazenar as classes para acesso a serviços da Internet

Estrutura de Diretórios



Seguindo a estrutura proposta, criaremos o diretório **lib/pages** e moveremos o código da classe **MyHomePage** para o arquivo **home.dart** no diretório **lib/pages** e alteramos o nome da classe para **Home**.

É necessário incluir a importação do pacote **material.dart** do flutter no arquivo.

Estrutura de Diretórios



home.dart:

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

class Home extends StatefulWidget {
  Home({Key key, this.title}) : super(key: key);

  final String title;

  @override
  _HomeState createState() => _HomeState();
}

class _HomeState extends State<Home> {
  int _counter = 0;

  void _incrementCounter() {
    setState(() {
      _counter++;
    });
  }
}
```

Estrutura de Diretórios



```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text(widget.title),
    ),
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          Text(
            'You have pushed the button this many times:',
          ),
          Text(
            '$_counter',
            style: Theme.of(context).textTheme.headline4,
          ),
        ],
      ),
    ),
  ),
),
```

Estrutura de Diretórios



```
floatingActionButton: FloatingActionButton(  
  onPressed: _incrementCounter,  
  tooltip: 'Increment',  
  child: Icon(Icons.add),  
),  
);  
}  
}
```


Estrutura de Diretórios



No arquivo **main.dart** é necessário alterar o nome da classe e incluir a importação para o arquivo **home.dart**.

Estrutura de Diretórios



main.dart:

```
import 'package:flutter/material.dart';
import 'package:hello_world/pages/home.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.lightGreen,
        canvasColor: Colors.lightGreen.shade100,
        visualDensity:
VisualDensity.adaptivePlatformDensity,
      ),
      home: Home(title: 'Hello World'),
    );
  }
}
```

Estrutura de Diretórios

