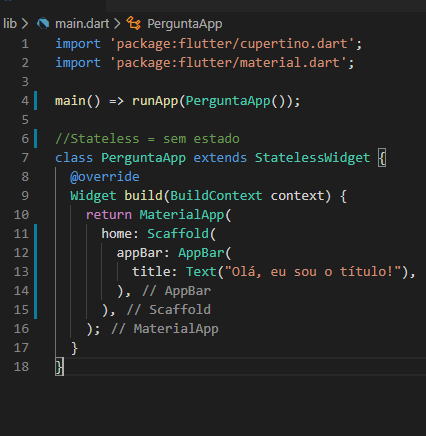
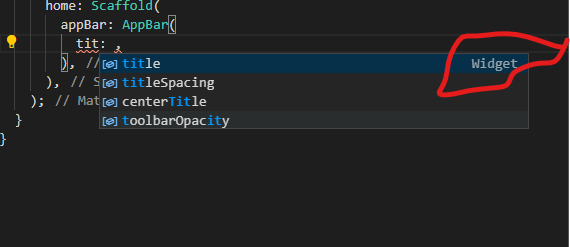
|  |
| --- |
| Imports de libs usadas no projeto, onde cupertino.dart é o padrão do iOS e material.dart padrão do Android. |
| Função main(), principal função, ela que inicia seu projeto. |
| Class do tipo Stateless é uma class sem estado e uma Class Stateful é com estado. |
| Todo Widget tem o construtor build que você deve passar o contexto. |
| MaterialApp geralmente usado para criar rotas, mas não só para isso. |
| Scaffold é a estrutura do app. |





|  |
| --- |
| Uma maneira de identificar se você deve informar um widget ou outro tipo de elemento é ao digitar o argumento vc vê o tipo que o argumento espera, nesse caso widget. |

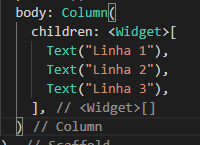


**Usando Column para organizar aplicação**

Conceito de flexBox.

Recebe vários widget’s como filhos, nesse caso uma lista.

Ex de implementação:



**Passando funções como parâmetro:**

Observe que passar uma função como parâmetro é diferente de chamar uma função, vamos ao ex:

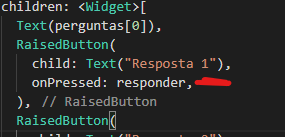
Chamando uma função:

funcao();

Repare que passo os parênteses normalmente.

Chamando uma função como parâmetro:

Apenas escreva -> responder, veja o ex onde passo a função responder como parâmetro, pois onPressed espera isso.



Se quiser passar como função, responder(), você pode fazer uma função que retorna um função, vejamos:

    void Function() funcaoQueRetornaUmaOutraFuncao () {

      return () {

        print("Pergunta Respondida!");

      };

    }

Agora, você pode chamar no onPressed: funcaoQueRetornaUmaOutraFuncao()

Temos então, 3 formas diferentes para passar uma função para o argumento onPressed do RaisedButton, vejamos:

            RaisedButton(

              child: Text("Resposta 1"),

              onPressed: responder,

            ),

            RaisedButton(

              child: Text("Resposta 2"),

              onPressed: () {

                print("Função do tipo normal")

              },

            ),

            RaisedButton(

              child: Text("Resposta 3"),

              onPressed: () => {

                print("Função do tipo Arrow Function")

              },

Algo importante a ser anotado é que quando passamos null para onPressed o botão fica desabilitado.

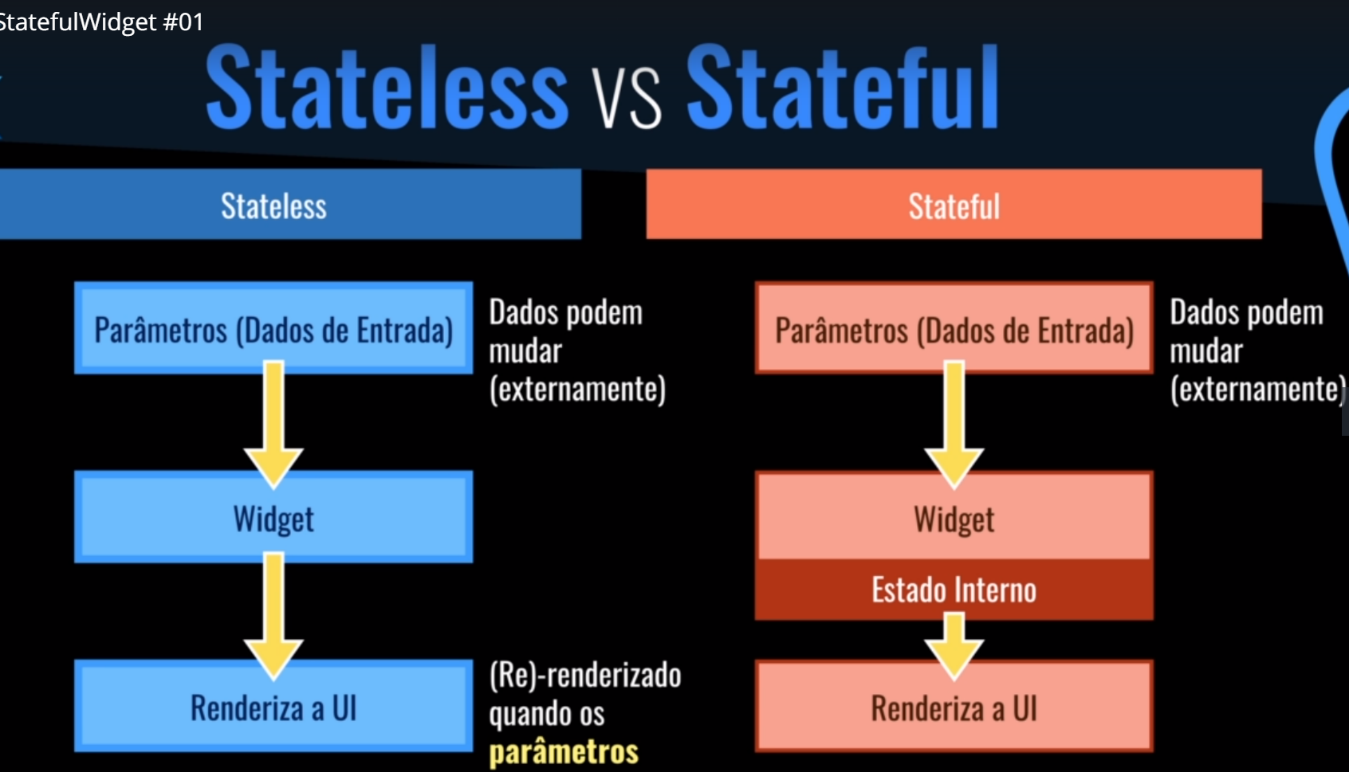
Widgets do tipo Stateless são sem estado e por isso você não deve fazer alterações de estado/layout no mesmo, pois não irão funcionar e não tente burlar isso.

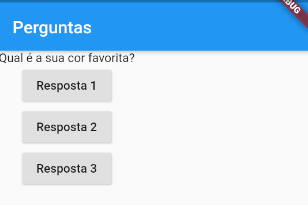
Use o Widget do tipo Stateful para isso.



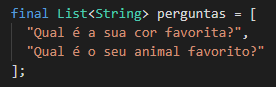
Stateless só muda de estado com ações exeternas.

Stateful muda de estado com ações externas e internas dele próprio.



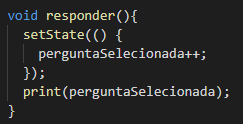
Como funciona uma aplicação Stateful, vejamos o App abaixo:

A ideia é alterar a pergunta quando clicar em algum botão, para isso precisamos passar uma função que receba como parâmetro o número da pergunta na lista, a lista:



Estamos chamando a pergunta em tela assim: 

Repare que passamos o argumento de posição a variável (perguntaSelecionada), onde o valor inicial era 0 e sempre é incrementada ao chamar a função responder, vejamos a função responder:



Repare que dentro de responder() temos o setState() que sempre que incrementamos um valor em (perguntaSelecionada) a pergunta em tela é atualizada conforme a lista, pois cada clique no botão incrementa um novo número que um é uma nova pergunta.

Podíamos fazer isso com Stateless, porém não teríamos o setState para atualizar as coisas em tela.

Tipo privados, como funcionam no Flutter

No Flutter, usamos a convenção underline (\_) para definir tipos privados, vejamos um ex:

Void \_teste(){

}

Essa função só poderá ser acessar pelo seu próprio arquivo, ela fica restrita ao seu espaço.

Já uma Class, você pode criar get e set para liberar o uso de recursos privados, ex:

class Pessoa {

  String nome;

  String \_cpf;

  get cpf{

    return \_cpf;

  }

  set cpf(String cpf){

    this.\_cpf = cpf;

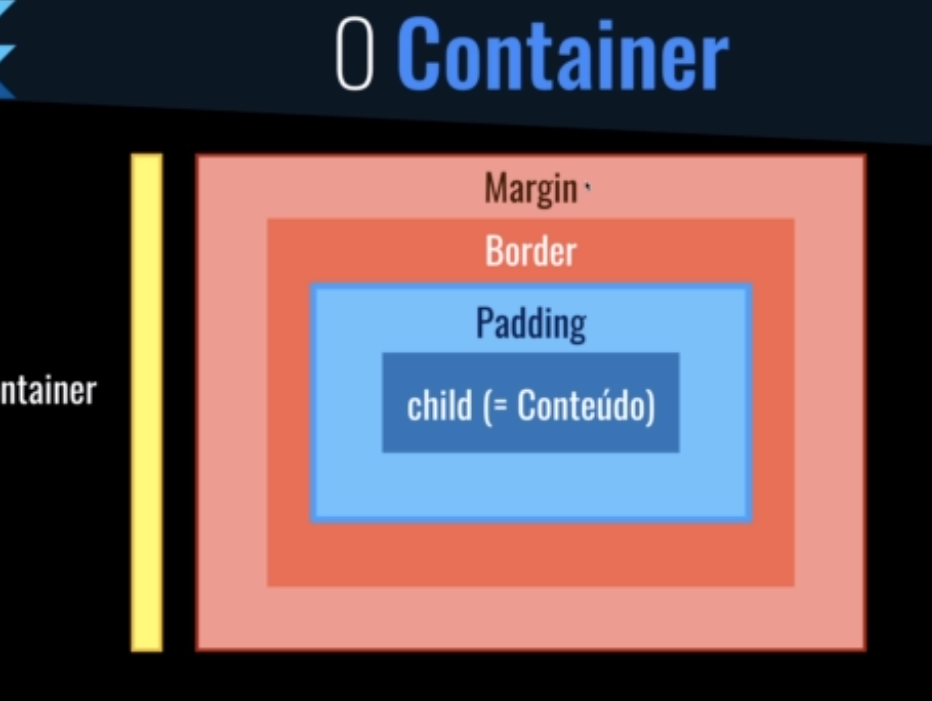
  }

}

Repare que no set e no get não precisamos passar o underline, apenas no retorno mesmo.

Sempre tente criar suas funções internas de cada tela privado.

**Container**



Código básico de um container envolvendo um Text:

 return Container(

      width: double.infinity,

      margin: EdgeInsets.all(10),

      child: Text(

        texto,

        style: TextStyle(fontSize: 22),

        textAlign: TextAlign.center,

        ),

    );

Criando um Widget que recebe uma função como parâmetro

import 'package:flutter/material.dart';

class Resposta extends StatelessWidget {

  final String texto;

  final void Function() quandoSelecionado;

  Resposta(this.texto, this.quandoSelecionado);

  @override

  Widget build(BuildContext context) {

    return Container(

      margin: EdgeInsets.fromLTRB(10,10,10,0),

      width: double.infinity,

      child: RaisedButton(

        textColor: Colors.white,

        color: Colors.blue,

        child:  Text(texto),

        onPressed: quandoSelecionado,

      ),

    );

  }

}

Criando uma lista de Map que contem String e Objetos:

    //Não é necessáio informar o tipo abaixo (List<Map<String,Object>>) pois o Flutter infere pra v c! xD

    final List<Map<String,Object>> perguntas = [

      {

        "texto": "Qual é sua cor favorita?",

        "resposta":['Preto','Vermelho','Azul','Branca'],

      },

      {

        "texto:":"Qual é o seu animal favorito?",

        "resposta":['Gato','Cachorro','Pato','Pinguim'],

      },

      {

        "texto": "Quantos filhos você tem?",

        "resposta":['Gato','Cachorro','Pato','Pinguim']

      }

    ];

Essa lista acima pode ser acessada assim:

perguntas[posicaoArrayPergunta]['texto']; //Sendo o texto ou resposta a escolha, porém resposta é outro array.

Como criar uma lista de Widget para armazenar outros widgets.

List<Widget> respostas = [];

//Populando essa lista com um exemplo

//Aqui no for estamos percorrendo a lista respostas, onde definimos acima e em cada interação adicionamos um Widget do tipo Resposta, onde serão as opões de respota para cada pergunta da lista perguntas

for(String textoResp in perguntas[\_perguntaSelecionada]['respostas']){

      respostas.add(Resposta(textoResp, \_responder));

}

Com o código acima teremos um array de widgets, que você deve chamar no seu componente para exibir em tela, mas isso é uma lista de widgets e como vc pode inserir isso dentro do Widget Column que estamos usando? Vejamos o operador **spread**, mais conhecido por **...,** isso aí, **três pontos**.

<https://medium.com/flutter-community/whats-new-in-dart-2-3-1a7050e2408d>

Column(

          children: <Widget>[

            Questao(perguntas[\_perguntaSelecionada]['texto']),

            ...respostas,

          ],

Isso que o spread faz acima é como se fizesse isso abaixo:

Resposta(perguntas[\_perguntaSelecionada]['respostas'], \_responder),

            Resposta(perguntas[\_perguntaSelecionada+1]['respostas'], \_responder),

            Resposta(perguntas[\_perguntaSelecionada+2]['respostas'], \_responder),

            Resposta(perguntas[\_perguntaSelecionada+3]['respostas'], \_responder),

Repare que o for prepara cada Widget Resposta e o spread adiciona ele na lista de filhos do Column.

**Um ponto importante, o que é um código imperativo e qual a diferença para um declarativo?**

Código imperativo: Código que atende a solicitação, ele não declara o que será feito, ele faz o que foi declaro, por exemplo: uma consulta sql -> SELECT \* FROM users;

Essa consulta sql é declarativa, porém que vai executar ele é o mecanismo do banco de dados que é imperativo.

Ex.: cod Flutter imperativo:

    //Lista de Widget

    List<Widget> respostas = [];

    for(String textoResp in perguntas[\_perguntaSelecionada]['respostas']){

      respostas.add(Resposta(textoResp, \_responder));

    }

Código declarativo:

Declarativo é todo código que declara ações, exemplo da consulta sql acima.

Ex.: cod Flutter declarativo:

//Lista respostas

    List<String> respostas = perguntas[\_perguntaSelecionada]['respostas'];

    //Lista de Widget

    List<Widget> widgetRespostas = respostas.map((t) => Resposta(t, \_responder)).toList();

Aqui eu só declaro para o map e ele faz o trabalho de converter em list de widget.

App Perguntas e Respostas

