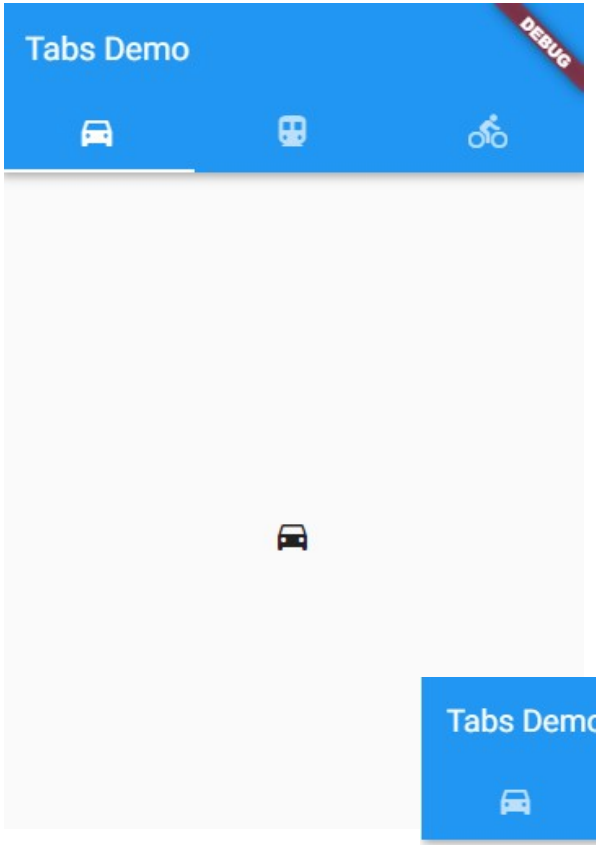
Componentes Visuais Básicos
Parte 3

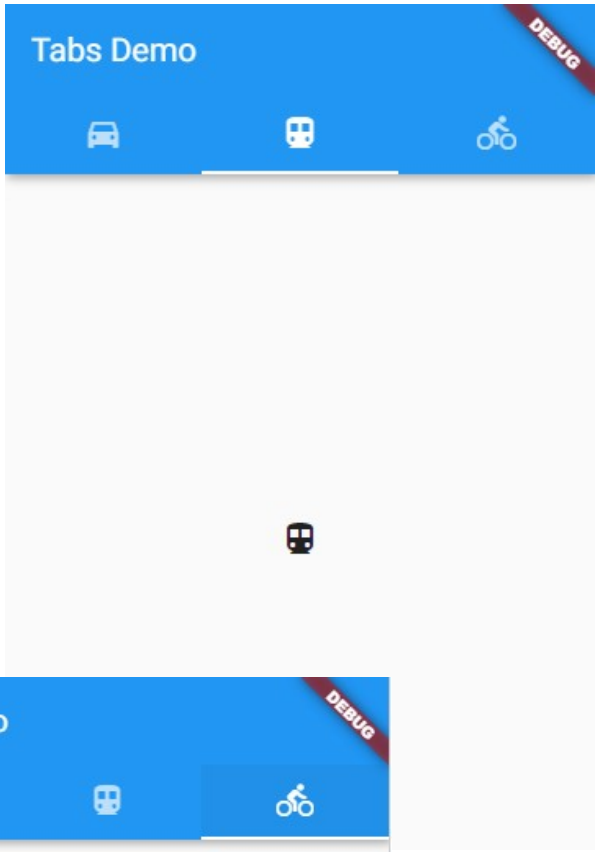
Instituto Federal do Espirito Santo – Campus Colatina
Desenvolvimento para sistemas móveis - Flutter
Professor: Giovany Frossard Teixeira

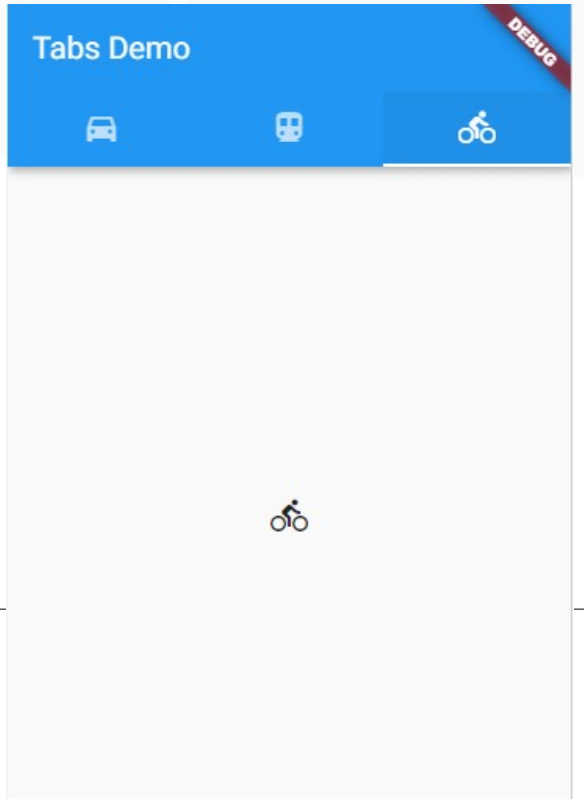
TabBar

- TabBar é um widget usado para colocarmos Tabs (abas) em nossos aplicativos.
- O seguinte link: https://flutter.dev/docs/cookbook/design/tabs da documentação do Flutter mostra como trabalhar com Tabs. O primeiro exemplo que será apresentado (próximo slide) foi obtido dessa documentação.
- O primeiro passo é a definição do **TabController** (nesse caso usando **DefaultTabController**). Uma configuração importante é o atributo **length** que define o numero de Tabs.
- Dentro da TabBar temos o atributo **tabs** que é uma lista de widgets e é usado para criarmos nossas Tabs. Posteriormente, usando **TabBarView** definimos o conteúdo dessas Tabs.

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

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

class TabBarDemo extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: DefaultTabController(
        length: 3,
        child: Scaffold(
          appBar: AppBar(
            bottom: TabBar(
              tabs: [
                Tab(icon: Icon(Icons.directions_car)),
                Tab(icon: Icon(Icons.directions_transit)),
                Tab(icon: Icon(Icons.directions_bike)),
              ],
            ),
            title: Text('Tabs Demo'),
          ),
          body: TabBarView(
            children: [
              Icon(Icons.directions_car),
              Icon(Icons.directions_transit),
              Icon(Icons.directions_bike),
            ],
          ),
        ),
      ),
    );
  }
}
```

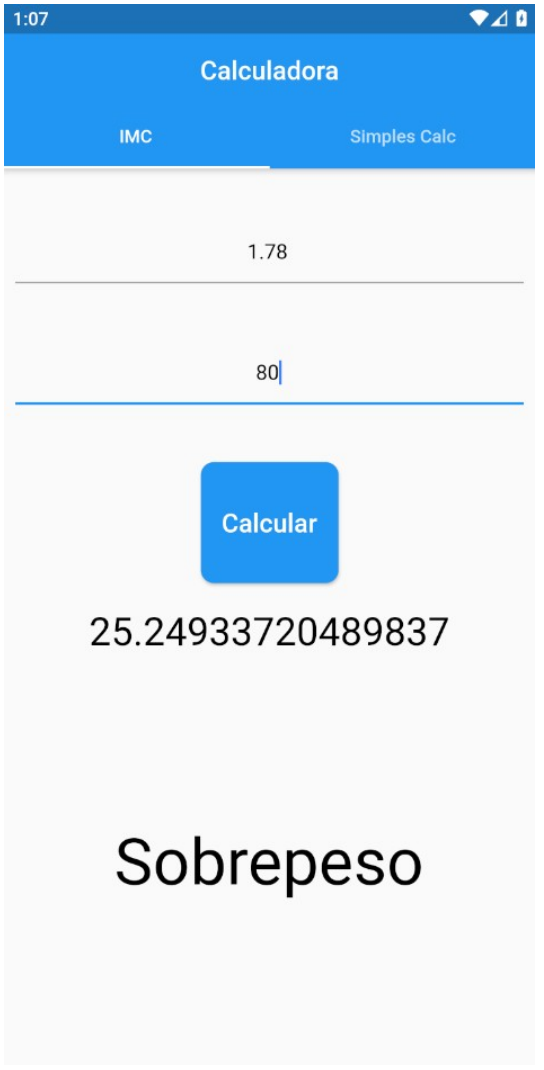


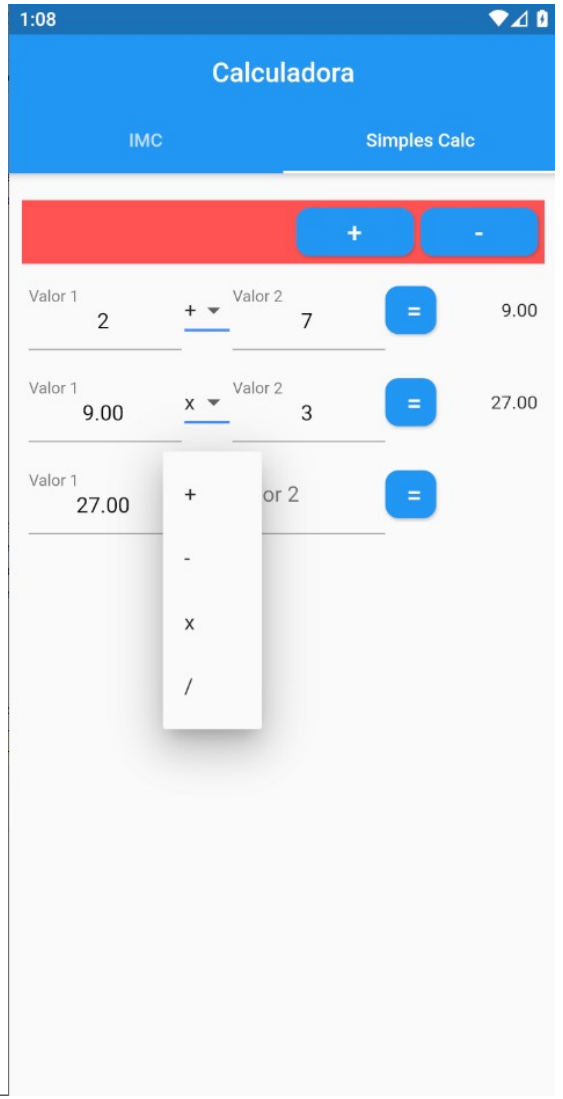




Tabs em um aplicativo simples

- Será apresentado um aplicativo que possui 2 abas. Uma faz o cálculo de IMC (Índice de Massa Corpórea) e a outra é uma calculadora de linha.
- Além da TabBar trabalharemos outros widgets como DropdownButton e alertas como SnackBar e Toast.





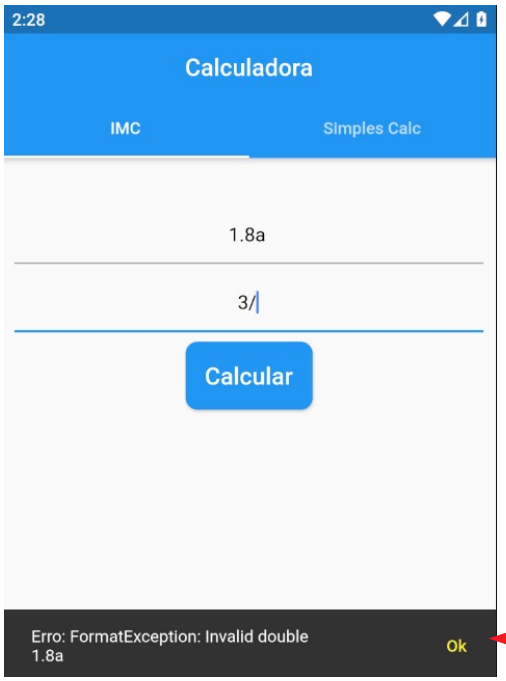
main.dart


```dart
import 'package:calculadoraimcflutter/tabs/tab_imc.dart';
import 'package:calculadoraimcflutter/tabs/tab_simples_calc.dart';
import 'package:flutter/material.dart';

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

class CalculadoraIMC extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      theme: ThemeData(
        primaryColor: Colors.blue,
      ), // ThemeData
      home: HomePage(),
    ); // MaterialApp
  }
}
```

- O atributo **debugShowCheckedModeBanner** vem por padrão como true. Esse atributo serve para mostrar uma faixa escrito DEBUG que vimos nos aplicativos anteriores
- Já o atributo **theme** aplicará um tema na aplicação. Esse tema define, por exemplo, cores padrões para o widget.
- O abributo **home** como já vimos define nossa “tela inicial”.

HomePage

- Temos aqui a definição das Tabs que terão os labels “IMC” e “Simples Calc”. O título da tela é “Calculadora” e ele foi centralizado.
- TabIMC() é o widget que desenha a primeira Tab e TabSimplesCalc() o widget que desenha a segunda.



SnackBar

Toast

```dart
class HomePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return DefaultTabController(
      length: 2,
      child: Scaffold(
        appBar: AppBar(
          centerTitle: true,
          title: Text("Calculadora"),
          bottom: TabBar(
            tabs: <Widget>[
              Tab(
                text: "IMC",
              ), // Tab
              Tab(text: "Simples Calc"),
            ], // <Widget>[]
          ), // TabBar
        ), // AppBar
        body: TabBarView(
          children: <Widget>[
            TabIMC(),
            TabSimplesCalc(),
          ], // <Widget>[]
        ), // TabBarView
      ), // Scaffold
    ); // DefaultTabController
  }
}
```


TabIMC

```
class TabIMC extends StatefulWidget {

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

class _TabIMCState extends State<TabIMC> {
  final controladorPeso = TextEditingController();
  final controladorAltura = TextEditingController();
  String valor_imc = "";
  String situacao_imc = "";

  @override
  void initState() {
    // TODO: implement initState
    super.initState();
  }

  @override
  void dispose() {
    // TODO: implement dispose
    super.dispose();
  }
```

- É um **StatefullWidget**. Isso significa que haverá (ou melhor, poderá haver) mudança de estado. Na prática isso significa que será possível chamar o método **setState**. Esse método permite alteração de elementos na tela e após sua chamada a tela será redesenhada, ou seja, chamará o método build novamente.
- **controladorPeso** e **controladorAltura** serão usados para termos acesso aos valores digitados.
- Já **valor_imc** é o texto com o resultado da conta para ser exibido na tela.
- **situacao_imc** será utilizado para exibir a situação daquele imc (Sobrepeso, por exemplo).
- **initstate()** é chamado quando a tela é criada e **dispose()** quando ela está sendo destruída.

TabIMC

```
Widget build(BuildContext context) {
  return Container(
    // Define as margens do Container
    margin: EdgeInsets.only(top: 40, left: 10, right: 10),
    child: Column(
      // No eixo y (principal da coluna) vai colocar os objetos no topo
      mainAxisAlignment: MainAxisAlignment.start,
      children: <Widget>[
        Expanded(
          // Peso desse componente (como no Android)
          flex: 1,
          // Notar que, como não estamos num formulário, o
          // validador desse campo não será acionado
          child: CampoEdicaoDouble(
            controlador: controladorAltura,
            texto_dica: "Informe a altura aqui:",
          ), // CampoEdicaoDouble
        ), // Expanded
        Expanded(
          flex: 1,
          child: CampoEdicaoDouble(
            controlador: controladorPeso,
            texto_dica: "Informe o peso aqui:",
          ), // CampoEdicaoDouble
        ), // Expanded
```

- O método build é quem desenha a tela.
- Inicialmente temos a definição das margens e o alinhamento vertical como **start** (ou seja, os widgets serão organizados a partir do topo da tela).
- Na sequência são apresentados os dois campos de edição para peso e altura (notar que o atributo controlador é setado em ambos widgets).
- Tem-se também o uso do **Expanded** para que ocorra a distribuição proporcional dos elementos na tela.
- Notar que **CampoEdicaoDouble** já implementa internamente a questão da validação (se é um número, se o campo não está vazio, etc.) mas ela não será acionada pois não estamos utilizando o widget **Form**.

TabIMC – setState

```
Expanded(
  flex: 1,
  child: BotaoAzul(
    texto: "Calcular",
    ao_clicar: () {
      setState(() {
        double imc = _calcularIMC();
        if(imc == null) {
          valor_imc = "";
          situacao_imc = "";
        }else {
          valor_imc = imc.toString();
          situacao_imc = _determinarSituacaoIMC(imc);
        }
      });
    },
  ), // BotaoAzul
), // Expanded
```

- Aqui temos um widget chamado **BotaoAzul**. Esse widget é uma customização do **ElevatedButton**.
- O primeiro atributo/parâmetro a ser setado é o texto do botão. O segundo parâmetro é uma função a ser acionada quando o botão for clicado.
- Quando o botão é clicado chama-se o **setState**. Esse método aciona o cálculo do IMC e se foi possível fazer a conta ele seta **valor_img** e **situacao_imc**, esses dos atributos serão usados na apresentação do resultado (próximo slide).
- Quando **setState** termina ele redesenha a tela e esses valores serão atualizados visualmente.

TabIMC

- Aqui temos a parte inferior da tela (os últimos widgets).
- Notar que a String do Text que mostra o imc é **valor_imc**.
- Notar que a String do Text que mostra a situação do imc é **situacao_imc**.
- Inicialmente **valor_imc** e **situacao_imc** são “”. Quando o usuário clica no botão (slide anterior) é feito o cálculo do IMC e essas Strings são atualizadas.

```
        Expanded(
          flex: 2,
          child: Container(
            margin: EdgeInsets.only(top: 20),
            child: Text(
              valor_imc,
              style: TextStyle(
                color: Colors.black,
                fontSize: 30,
              ), // TextStyle
            ), // Text
          ), // Container
        ), // Expanded
        Expanded(
          flex: 2,
          child: Text(
            situacao_imc,
            style: TextStyle(
              color: Colors.black,
              fontSize: 50,
            ), // TextStyle
          ), // Text
        ), // Expanded
      ], // <Widget>[]
    ), // Column
  ); // Container
}
```


Cálculo do IMC

```
double _calcularIMC() {
  try {
    double altura = double.parse(controladorAltura.text);
    double peso = double.parse(controladorPeso.text);
    double imc = peso / (altura * altura);
    return imc;
  } on FormatException catch (erro) {
    // O SnackBar precisa de um context definido dentro do Scaffold
    Scaffold.of(context).showSnackBar(
      SnackBar(
        content: Text("Erro: $erro"),
        duration: Duration(seconds: 5),
        action: SnackBarAction(
          textColor: Colors.yellow,
          label: "Ok",
          onPressed: () {
            setState(() {
              _reinicializarCampos();
            });           },
        ), // SnackBarAction
      ), // SnackBar
    );
  }
}
```

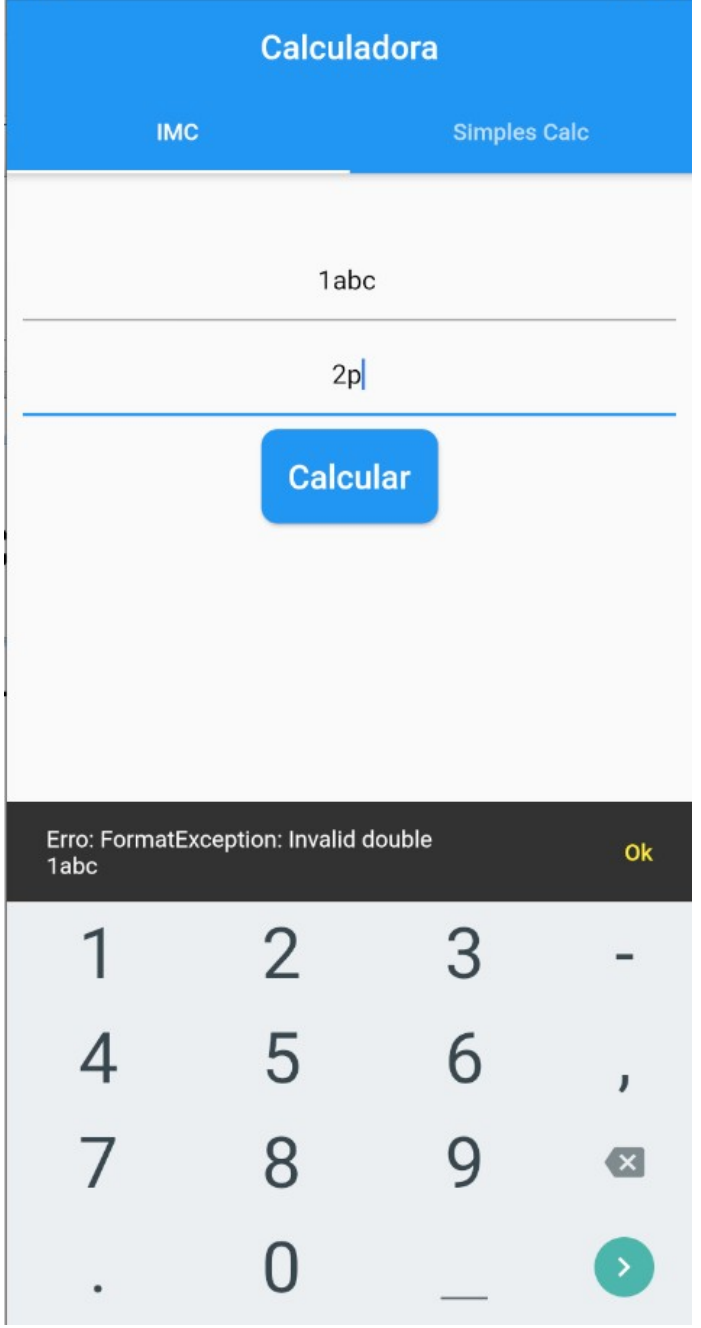
- Aqui temos o método que calcula o IMC. Notar que ele usa os controladores de altura e peso para obter os valores digitados pelo usuário.
- A conta é feita e se tudo estiver correto o IMC é retornado.
- Entretanto o usuário pode digitar algo que não é um número e dessa forma não será possível fazer a conta.
- Quando isso ocorre é disparada uma exceção e criamos um SnackBar para apresentar a mensagem de erro.

SnackBar

- O SnackBar é uma barra que aparece na parte inferior da tela. É um widget presente no Flutter (não é um plugin externo).

```
} on FormatException catch (erro) {
  // O ScnackBar precisa de um context definido dentro do Scaffold
  Scaffold.of(context).showSnackBar(
    SnackBar(
      content: Text("Erro: $erro"),
      duration: Duration(seconds: 5),
      action: SnackBarAction(
        textColor: Colors.yellow,
        label: "Ok",
        onPressed: () {
          setState(() {
            _reinicializarCampos();
          });          },
      ), // SnackBarAction
    ), // SnackBar
  );
}
```

```
void _reinicializarCampos() {
  controladorPeso.text = "";
  controladorAltura.text = "";
  valor_imc = "";
  situacao_imc = "";
}
```



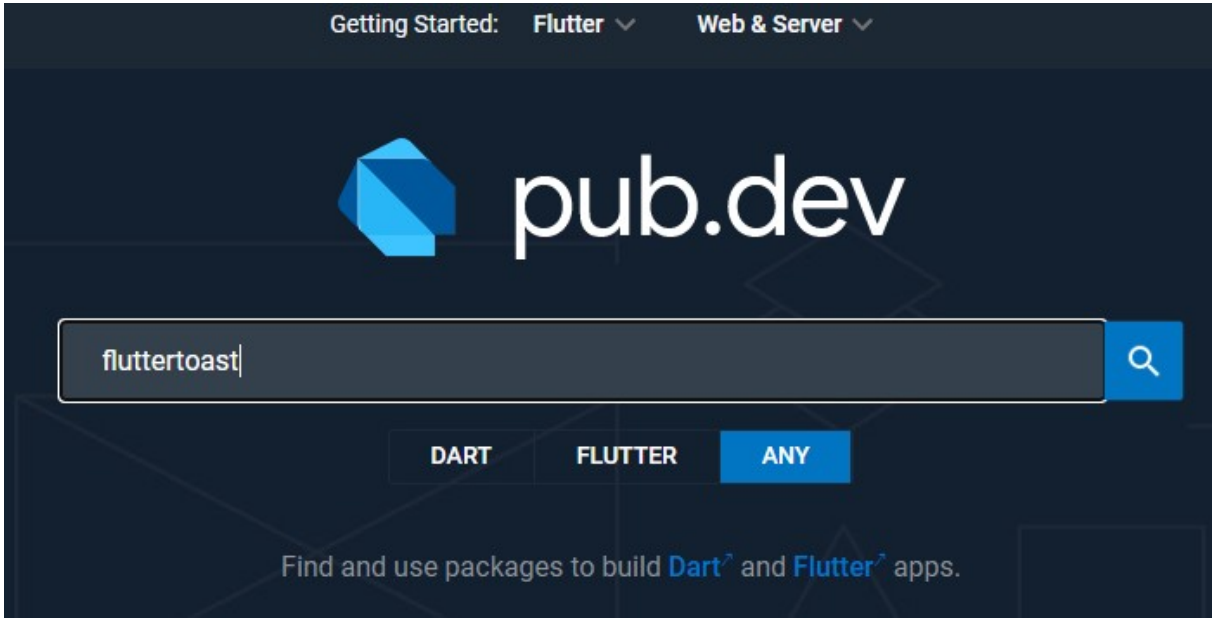
Finalizando a TabIMC

- Faltou apenas apresentar o método que define, a partir do IMC, a situação do indivíduo.

```
String _determinarSituacaoIMC(double imc) {
  if (imc < 18.5)
    return "Abaixo do peso";
  else if (imc < 24.9)
    return "Peso normal";
  else if (imc < 29.9)
    return "Sobrepeso";
  else if (imc < 34.9)
    return"Obesidade grau 1";
  else if (imc < 39.9)
    return "Obesidade grau 2";
  else
    return "Obesidade grau 3";
}
```

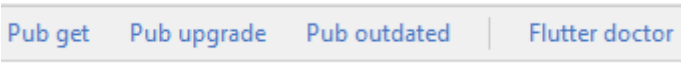

Bibiliotecas externas para Flutter

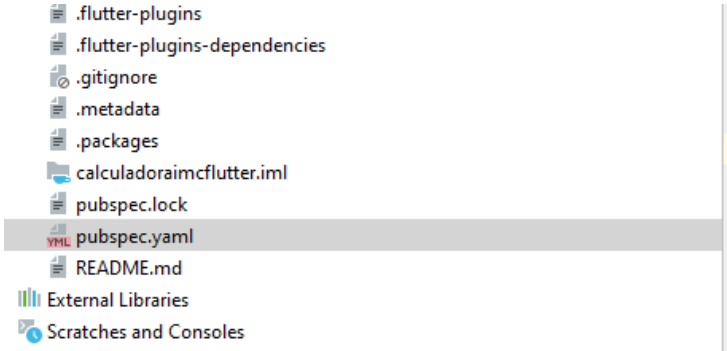
- No site pub.dev é possível obter plugins para utilizarmos em nossos projetos.
- Especificamente no projeto da Calculadora utilizamos o **fluttertoast**. Para instalar esse o plugin o primeiro passo é ir ao site pub.dev. Em seguida buscamos pelo plugin no site pub.dev.

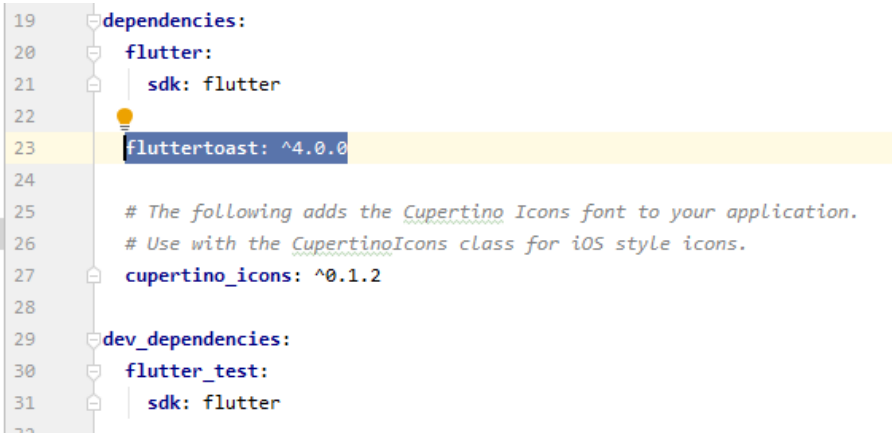


fluttertoast - Instalação

- Para usar o fluttertoast devemos seguir as orientações descritas na documentação do plugin (https://pub.dev/packages/fluttertoast#-installing-tab-).
- O primeiro passo é acrescentar a dependência do fluttertoast no arquivo **pubspec.yaml** (esses passos vão valer para praticamente todos os plugins que formos colocar nos projetos).
- Em seguida devemos usar $ **flutter pub get** no prompt ou de forma mais prática clicar no menu Pub get.







Toast

- Com o plugin obtido basta fazer referência a ele no arquivo onde for utilizado:

```
import 'package:fluttertoast/fluttertoast.dart';
```

- O Toast é uma mensagem que ocorre por um tempo e depois desaparece sozinha. No próximo slide mostraremos o código do Toast usado na calculadora.
- O Toast é possivelmente a forma mais conhecida de emitir uma mensagem de alerta temporária no Android Nativo (Java/Kotlin).

Toast

```
_makeToast(String mensagem){
  Fluttertoast.showToast(
    msg: "Erro: $mensagem",
    toastLength: Toast.LENGTH_SHORT,
    gravity: ToastGravity.CENTER,
    timeInSecForIosWeb: 5,
    backgroundColor: Colors.red,
    textColor: Colors.white,
    fontSize: 16.0,
  );
}
```

- msg: define o texto que aparecerá no Toast.
- toastLength: define o tempo de exibição (SHORT ou LONG). Essa configuração não afeta o iOS.
- gravity: define o alinhamento do Toast.
- timeInSecForIosWeb: define o tempo do Toast no iOS (e possivelmente na web – nunca utilizei).
- backgroundColor: define a cor de fundo do Toast.
- textColor: define a cor do texto.
- fontSize: define o tamanho da fonte do texto.

Tab_simples_calc

```dart
@override
Widget build(BuildContext context) {
  return SingleChildScrollView(
    // Define as margens do Container
    padding: EdgeInsets.only(top: 20, left: 10, right: 10),
    child: Column(
      // No eixo y (principal da coluna) vai colocar
      // os objetos no topo
      mainAxisAlignment: MainAxisAlignment.start,
      children: _linhas(),
    ), // Column
  ); // SingleChildScrollView
}
```

- Inicialmente definimos um **SingleChildScrollView** para permitir scroll se necessário (número de linhas crescer muito.
- Em seguida, definimos as margens internas (padding).
- Os objetos serão colocados a partir do topo na coluna (MainAxisAlignment.start).
- Por fim chamamos um método privado _linhas().

Tab_simples_calc

- Para trabalhar com as linhas da calculadora foi necessário definir um controlador para gerenciar a lista de resultados.
- Isso é importante para acrescentar, excluir e modificar linhas.

```
class ControladorLinhaConta{
  List<String> resultados;

  ControladorLinhaConta(){
    resultados = List<String>();
  }

  void adicionarNovoResultado(String resultado){
    resultados.add(resultado);
  }

  int obterQuantidadeResultados(){
    return resultados.length;
  }

  String obterUltimoResultado(){
    return resultados[resultados.length - 1];
  }

  void removerUltimoResultado(){
    resultados.removeLast();
  }

  void modificarUltimoResultado(String resultado) {
    resultados[resultados.length - 1] = resultado;
  }
}
```


_linhas()

- **linhas == null** vai acontecer apenas na primeira vez. Nesse caso serão colocados os botões “+” e “-” com seus comportamentos (e chamadas a **setState()**.
- **LinhaConta** é um widget personalizado usado para desenhar uma linha.
- O controlador precisa ser passado para o construtor de LinhaConta. É esse controlador que fará a interação da Tab com as linhas dentro dela.

```
List<Widget> _linhas() {
  if (linhas == null) {
    linhas = <Widget>[
      Container(
        color: Colors.redAccent,
        padding: EdgeInsets.only(right: 5),
        child: Row(
          mainAxisAlignment: MainAxisAlignment.end,
          children: <Widget>[
            BotaoAzul(
              texto: "+",
              ao_clicar: () {
                setState(() {
                  // Condição para evitar que o botão de + acione novas linhas sem que
                  // já tenhamos resultados das linhas anteriores
                  if ((linhas.length-1) == controlador.obterQuantidadeResultados()) {
                    linhas.add(
                      LinhaConta(
                        controladorLinhaConta: controlador,
                      ), // LinhaConta
                    );
                  } else {
                    Fluttertoast.showToast(
                        msg: "Você deve terminar a operação anterior ( clicar em '=' ) ",
                        toastLength: Toast.LENGTH_SHORT,
                        gravity: ToastGravity.CENTER,
                        timeInSecForIosWeb: 5,
                        backgroundColor: Colors.red,
                        textColor: Colors.white,
                        fontSize: 16.0
                    );
                  }
                });
              },
            ), // BotaoAzul
```


_linhas()

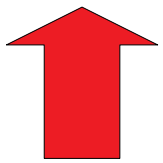
- Aqui temos o botão de “-” e seu comportamento.
- Além do código de criação da primeira linha

```
        SizedBox(
          width: 5,
        ), // SizedBox
        BotaoAzul(
          texto: "-",
          ao_clicar: () {
            setState(() {
              linhas.removeLast();
              // Posso ter clicado ou não no operador "=" da LinhaConta
              // se foi clicado a linha já tem valor no controlador, caso contrário não
              // por isso se faz necessária essa validação
              if (linhas.length ==
                  controlador.obterQuantidadeResultados())
                controlador.removerUltimoResultado();
            });
          },
        ), // BotaoAzul
      ], // <Widget>[]
    ), // Row
  ), // Container
  LinhaConta(controladorLinhaConta: controlador),
 ]; // <Widget>[]
 }
 return linhas;
}
```


LinhaConta

- A classe LinhaConta possui um Form e dessa forma é possível usar a validação nos campos.

```dart
// Variável utilizada para validar os campos do formulá
final _formkey = GlobalKey<FormState>();

@override
Widget build(BuildContext context) {
  _inicializarValor1();
  return Form(
    // Com o formulário conseguimos acionar os validado
    // edição
    key: _formkey,
```



```dart
Expanded(
  // Peso desse componente (como no Android)
  flex: 1,
  child: BotaoAzul(
    texto: "=",
    ao_clicar: (){
      setState(() {
        // Chamando os validadores dos campos de edição
        bool formOk = _formkey.currentState.validate();
        if (formOk) {
          _executarOperacao();
          if (resultado_ja_adicionado) {
            widget.controladorLinhaConta.modificarUltimoResultado(
                _resultado);
          } else {
            widget.controladorLinhaConta.adicionarNovoResultado(
                _resultado);
            resultado_ja_adicionado = true;
          }
        }
      });
    },
    marcador_foco: _focoBotaoIgual,
  ), // BotaoAzul
), // Expanded
```


LinhaConta

```
Expanded(
  // Peso desse componente (como no Android)
  flex: 3,
  child: CampoEdicaoDouble(
    controlador: controladorValor1,
    texto_label: "Valor 1",
    recebedor_foco: _focoValor2,
  ), // CampoEdicaoDouble
), // Expanded
```

Validação padrão

```
Expanded(
  // Peso desse componente (como no Android)
  flex: 3,
  child: CampoEdicaoDouble(
    controlador: controladorValor2,
    texto_label: "Valor 2",
    marcador_foco: _focoValor2,
    recebedor_foco: _focoBotaoIgual,
    validador: (String text){
      if((text.isEmpty) || (text.trim().isEmpty))
        return "Não preenchido";
      try{
        double valor2 = double.parse(controladorValor2.text);
        if ((_operacao_selecionada == "/") && (valor2 == 0)) {
          _makeToast("Não é possível dividir por 0");
          return "Divisão por 0";
        }else
          return null;
      }on FormatException catch (erro){
          return "Digite um número";
      }
    },
  ), // CampoEdicaoDouble
), // Expanded
```

Validação modificada

SeletorOpcoes

- É um widget customizado a partir de um `DropdownButton`.

```
Expanded(
  // Peso desse componente (como no Android)
  flex: 1,
  child: Center(
    child: SeletorOpcoes(
      opcoes: _operacoes,
      valor_selecionado: _operacao_selecionada,
      ao_mudar_opcao: (String novoItemSelecionado){
        _operacao_selecionada = novoItemSelecionado;
        _resultado = "";
      },
    ), // SeletorOpcoes
  ), // Center
), // Expanded
```

```
// Dados para o DropdownButton
var _operacoes = ["+", "-", "x", "/"];
var _operacao_selecionada = "+";
```


SeletorOpcoes

```
class SeletorOpcoes extends StatefulWidget {
  List<String> opcoes;
  Icon icone = Icon(Icons.arrow_drop_down);
  String valor_selecionado;
  Function ao_mudar_opcao;

  SeletorOpcoes({
    Key key,
    @required this.opcoes,
    this.icone, this.valor_selecionado,
    this.ao_mudar_opcao}): super(key: key);

  _SeletorOpcoesState createState() => _SeletorOpcoesState();

}
```

- O DropdownButton (equivalente ao ComboBox do Java Swing) é um widget que trabalha com uma listagem de itens.
- Para facilitar seu uso foi criado um SeletorOpcoes que já deixa pré-configuradas algumas funcionalidades.

SeletorOpcoes

- items é do tipo `List<DropdownMenuItem<T>>` para facilitar usamos o método **map** para transformar a lista de Strings na lista de **DropdownMenuItem**.
- Outro ponto importante é o uso da palavra **widget**. Para obter atributos definidos no StatefulWidget é necessário usar essa palavra reservada.

```dart
class _SeletorOpcoesState extends State<SeletorOpcoes> {
  @override
  Widget build(BuildContext context) {
    return DropdownButton<String>(
      value: widget.valor_selecionado,
      icon: widget.icone,
      iconSize: 24,
      elevation: 16,
      underline: Container(
        height: 2,
        color: Colors.blueAccent,
      ), // Container
      items : widget.opcoes.map((String dropDownStringItem) {
        return DropdownMenuItem<String>(
          value: dropDownStringItem,
          child: Text(dropDownStringItem),
        ); // DropdownMenuItem
      }).toList(),
      onChanged: ( String novoItemSelecionado) {
        setState(() {
          widget.valor_selecionado = novoItemSelecionado;
          widget.ao_mudar_opcao(novoItemSelecionado);
        });
      },
    ); // DropdownButton
  }
}
```


Dúvidas?