ISW-011
Desenv. para Dispositivos Móveis 1 (DDM1)

DDM-1 – ISW-011
Prof. Davi dos Reis

Aula 06 – Flutter – componentes básicos

Boas práticas para organização do código

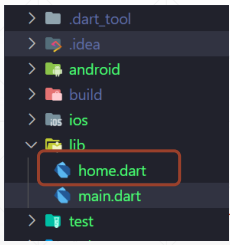
Flutter – boas práticas de organização

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

class WidgetHome extends StatelessWidget {
  const WidgetHome({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Hello, world!"),
        centerTitle: true,
      ),
      body: Container(
        color: Colors.white,
        child: Center(
          child: Text("Hello, turma do 5º. SI!",
              style: TextStyle(
                fontSize: 30,
                color: Colors.blue,
                fontWeight: FontWeight.bold,
                fontStyle: FontStyle.italic,
                decoration: TextDecoration.underline,
                decorationColor: Colors.red,
              )),
        ),
      ),
    );
  }
}
```

- **Dica 1**: A cada **novo widget** personalizado, criar uma nova classe (arquivo ".dart" separado)



Flutter – boas práticas de organização

- Note que o nome do arquivo deve ser iniciando com letra minúscula: "home.dart".
- Entretanto, o nome da classe inicia com letra maiúscula class WidgetHome

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

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

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      theme: ThemeData(
          primaryColor: Colors.red
      ),
      home: WidgetHome(),
    );
  }
}
```

Flutter – boas práticas de organização

- **Dica 2**: Separar todos os widgets padrão do Flutter em métodos privados na mesma classe.
- Vamos refatorar agora a classe “home.dart”, separando grandes widgets padrão do Flutter em **métodos privados** dentro da mesma classe:
- Nota: o que torna o método privado?
- Resp.: o “_” e a letra minúscula

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

class WidgetHome extends StatelessWidget {
  const WidgetHome({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Hello, world!"),
        centerTitle: true,
      ),
      body: _body(),
    );
  }

  _body(){
    return Container(
      color: Colors.white,
      child: Center(
        child: _text(),
      ),
    );
  }
```

```dart
_text(){
    return Text("Hello, turma do 5º. SI!",
        style: TextStyle(
          fontSize: 30,
          color: Colors.blue,
          fontWeight: FontWeight.bold,
          fontStyle: FontStyle.italic,
          decoration: TextDecoration.underline,
          decorationColor: Colors.red,
        ));
  }
}
```

Prof. Davi ISW-011 (DDM-1) 5

Componentes básicos

Prof. Davi ISW-011 (DDM-1) 6

Flutter – ListView

- O ListView é uma lista de widgets organizados linearmente, sendo um dos widgets de rolagem mais usados.
- Ele exibe seus filhos um após o outro na direção da rolagem.
- Existem quatro opções para construir um ListView: construtor padrão (“List<Widget>”), ListView.builder, ListView.separated e ListView.custom. Iniciaremos com o **ListView padrão**.

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

void main() => runApp(MyApp());
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);
```

```dart
@override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      theme: ThemeData(primaryColor: Colors.blue),
      home: Scaffold(
        appBar: AppBar(
          title: Text("Hello world, SI!"),
          backgroundColor: Colors.redAccent,
          centerTitle: true,
        ),
        body: Container(
          color: Colors.white,
          child: ListView(
            children: [
              Text("Nome: Rubens Lara",
                style: TextStyle(
                  fontSize: 20,
                  color: Colors.blue
                ),
              ),
              Text("Formação: Sistemas para Internet"),
              Text("Disciplina: DDM1 (Flutter)"),
            ],
          ),
        ),
      )); } }
```


Flutter – ElevatedButton

- No Flutter, é possível criar um botão através do widget ElevatedButton.

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

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

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      theme: ThemeData(primaryColor: Colors.blue),
      home: Scaffold(
        appBar: AppBar(
          title: Text("'Hello, World' com Flutter"),
          centerTitle: true,
        ),
        body: _body(),
```

```dart
      ),
    );
  }

  _body() {
    return Container(
      color: Colors.white,
      child: Center(
        child: _button(),
      ),
    );
  }
```

Flutter – ElevatedButton

- No Flutter, é possível criar um botão através do widget ElevatedButton.

```
  _button() {
    return ElevatedButton(
      style: ElevatedButton.styleFrom(
        backgroundColor: Colors.red,
      ),
      onPressed: onClick,
      child: Text(
        "Confirmar",
        style: TextStyle(
          color: Colors.white,
          fontSize: 30,
        ),
      ),
    );
  }

  onClick() {
    print("Clicou aqui");
  }
}
```


Flutter – Row e Column

- Os posicionamentos dos componentes na tela no Flutter são feitos através de Columns e Rows.
- Em resumo:
  - Columns organizam os componentes um embaixo do outro;
  - Rows organizam os componentes um ao lado do outro.
- **Exemplo Row:**

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

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

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      theme: ThemeData(primaryColor: Colors.blue),
      home: Scaffold(
        appBar: AppBar(
          title: Text("'Hello world', com Flutter"),
          centerTitle: true,
        ),
        body: _body(),
      ),
    );
  }
```

Flutter – Row e Column

- Os posicionamentos dos componentes na tela no Flutter são feitos através de Columns e Rows.
- Em resumo:
  - Columns organizam os componentes um embaixo do outro;
  - Rows organizam os componentes um ao lado do outro.
- **Exemplo Row:**
- É possível notar que a Row pode ter vários filhos (*children*) dentro (no caso, os componentes que aparecem na tela).

```
_body() {
    return Container(
      color: Colors.yellow,
      child: Row(
        children: <Widget>[
          _button(),
          _button(),
          _button(),
        ],
      ),
    );
  }

  _button() {
    return ElevatedButton(
      style: ElevatedButton.styleFrom(
        backgroundColor: Colors.red,
      ),
      onPressed: onClick,
      child: Text(
        "OK",
        style: TextStyle(
          color: Colors.white,
          fontSize: 20,
        ),
      ),
    );
  }

  onClick() {
    print("Clicou aqui");
  }
}
```


Flutter – Row e Column

- Os posicionamentos dos componentes na tela no Flutter são feitos através de Columns e Rows.
- Em resumo:
  - Columns organizam os componentes um embaixo do outro;
  - Rows organizam os componentes um ao lado do outro.
- **Exemplo Column:**

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

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

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      theme: ThemeData(primaryColor: Colors.blue),
      home: Scaffold(
        appBar: AppBar(
          title: Text("'Hello world', com Flutter"),
          centerTitle: true,
        ),
        body: _body(),
      ),
    );
  }
```

Flutter – Row e Column

- Os posicionamentos dos componentes na tela no Flutter são feitos através de Columns e Rows.
- Em resumo:
  - Columns organizam os componentes um embaixo do outro;
  - Rows organizam os componentes um ao lado do outro.
- **Exemplo Column:**
- Tal como na Row, a Column também pode ter vários filhos (*children*) dentro (no caso, os componentes que aparecem na tela).

```
_body() {
   return Container(
     color: Colors.yellow,
     child: Column(
       children: <Widget>[
         _button(),
         _button(),
         _button(),
       ],
     ),
   );
 }
_button() {
   return ElevatedButton(
    style: ElevatedButton.styleFrom(
      backgroundColor: Colors.red,
    ),
```

```
   onPressed: onClick,
     child: Text(
       "OK",
       style: TextStyle(
         color: Colors.white,
         fontSize: 20,
       ),
     ),
   );
 }

 onClick() {
   print("Clicou aqui");
 }
}
```


Flutter – MainAxisSize e MainAxisAlignment

- Ao trabalhar com Columns e Rows, além de organizarmos eles na tela, podemos também alinhar os elementos dentro deles da forma que preferimos. Para isso utilizamos as propriedades **MainAxisSize** e **MainAxisAlignment**.
- **MainAxisSize:**
  - min;
  - max.

```
_body() {
 return Container(
   color: Colors.yellow,
   child: Row(
     //mainAxisSize: MainAxisSize.min,
     mainAxisSize: MainAxisSize.max,
     children: <Widget>[
```

Flutter – MainAxisSize e MainAxisAlignment

- Ao trabalhar com Columns e Rows, além de organizarmos eles na tela, podemos também alinhar os elementos dentro deles da forma que preferimos. Para isso utilizamos as propriedades **MainAxisSize** e **MainAxisAlignment**.
- **MainAxisAlignment:**
  - start;
  - center;
  - end.

```
_body() {
  return Container(
    color: Colors.yellow,
    child: Row(
      //mainAxisSize: MainAxisSize.min,
      mainAxisSize: MainAxisSize.max,
      //mainAxisAlignment: MainAxisAlignment.start,
      mainAxisAlignment: MainAxisAlignment.center,
      //mainAxisAlignment: MainAxisAlignment.end,
      children: <Widget>[
```


Flutter – CrossAxisAlignment

- Assim como o MainAxisAlignment, o CrossAxisAligment também é utilizado para alinhamento, porém, sempre na direção inversa. Ou seja:
  - Quando usamos uma **Row**, o MainAxisAlignment alinha os componentes na horizontal e o CrossAxisAligment alinha na vertical.
  - Quando usamos uma **Column**, o MainAxisAlignment alinha os componentes na vertical e o CrossAxisAligment alinha na horizontal.
- **CrossAxisAlignment:**
  - start;
  - center;
  - end.

```
_body() {
  return Container(
    height: double.infinity,
    color: Colors.yellow,
    child: Row(
      mainAxisAlignment: MainAxisAlignment.center,
      //crossAxisAlignment: CrossAxisAlignment.start,
      //crossAxisAlignment: CrossAxisAlignment.center,
      crossAxisAlignment: CrossAxisAlignment.end,
      children: <Widget>[
```

Obrigado!