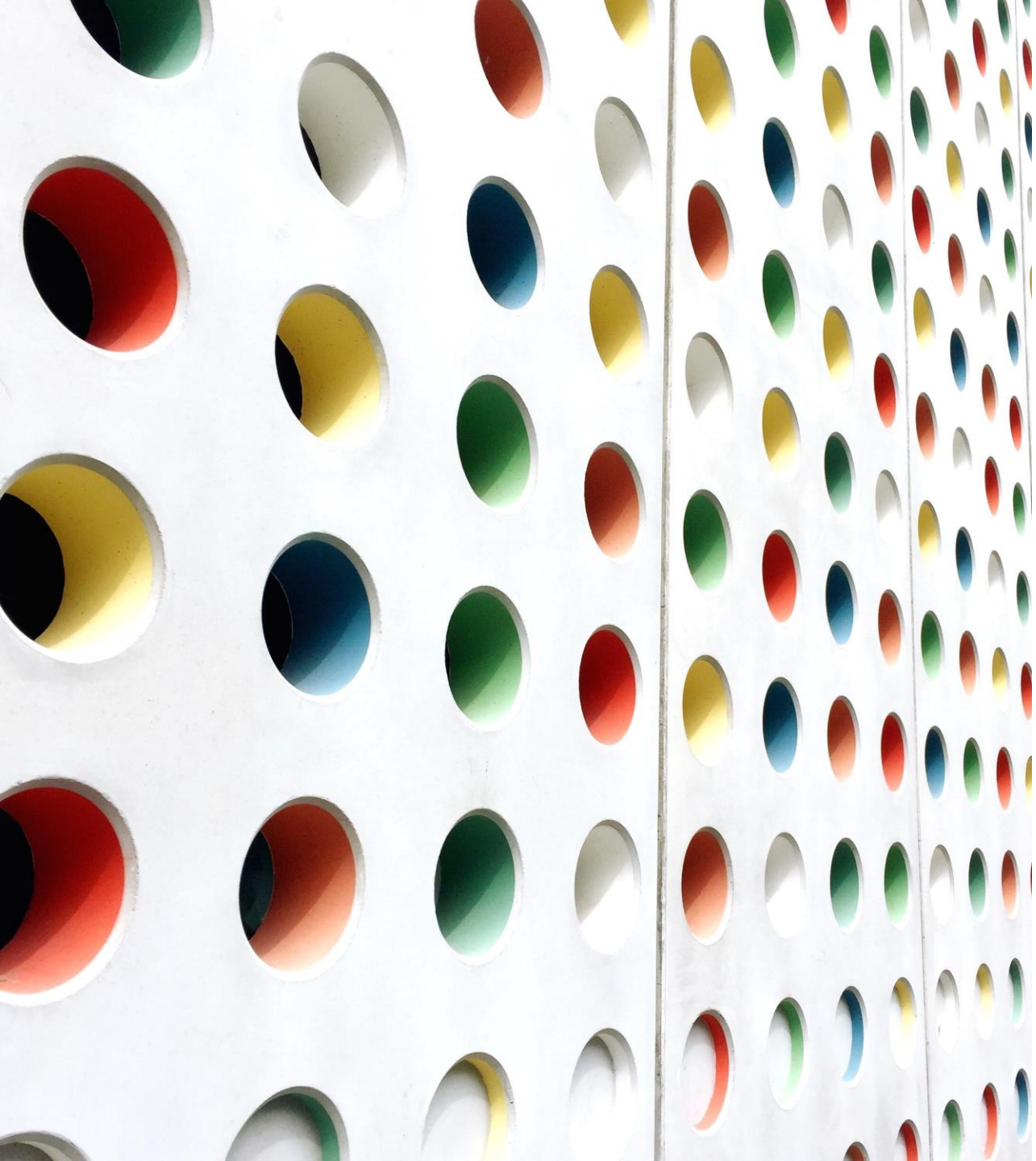


# Programação para Dispositivos Móveis

## FLUTTER E DART

Professor Jean Carlo Wagner

14/03/2023

# Agenda

- Dart: variáveis
- Dart: constantes
- Dart: *types*
- Dart: valores numéricos
- Dart: valores booleanos
- Dart: listas e mapas
- '*as*' e '*is*'
- Controle sobre a UI: exemplo básico
- Navegação: exemplo básico

# Dart: variáveis

- Tudo no Dart é um objeto, mesmo simples números, funções e até null, ou seja, todos são instâncias de classes e todos são estendidos a partir de uma classe comum Object.
- Podemos declarar de duas formas:

```
var x;
var x = “Belize”;
```

```
<tipo específico> x;
String x = “Belize”;
```

- Dart infere o tipo a partir do valor atribuído.
- Há uma diretriz de estilo que diz que devemos declarar variáveis locais usando var enquanto as outras devem usar uma declaração de tipo (opcional).

# Dart: variáveis

- Outra opção é a declaração de tipo **dynamic**, informando ao Dart que o que x referencia pode mudar com o tempo:

```
dynamic x = “42”;
x = 19;
```

O que não gera erro após a mudança de tipo String para inteiro.

- Uma última opção é:

```
Object x = “Belize”;
```

- Já que tudo no Dart é estendido a partir da classe comum Object, essa alternativa também funciona. Contudo se uma variável for de tipo Object e tentarmos chamá-la na referência a um método que não existe, observaremos um erro de tempo de compilação. Com dynamic isso não ocorre e veremos o problema no tempo de execução.

# Dart: constantes

- As palavras-chave const e final definem uma variável como sendo uma constante, ou seja, um valor final imutável:

```
const x = "Brasil";
const String = "Brasil";
final x = "Brasil";
```

- As variáveis const são constantes no tempo de compilação, o que significa que seu valor não pode depender de nada que venha do tempo de execução:

```
const x = DateTime.now();
```

isso não funcionaria, mas a linha a seguir funcionaria:

```
final x = DateTime.now();
```

# Dart: constantes

- Basicamente, final significa que podemos definir a variável uma vez, mas no tempo de execução, enquanto const indica que ela só pode ser definida uma vez e seu valor já deve ser conhecido no tempo de compilação.
- Ainda sobre const, podemos aplicá-la tanto a valores quanto a variáveis. Por exemplo:

```
List lst = const [1, 2, 3];
print(lst);
lst = [4, 5, 6];
print(lst);
lst[0] = 666;
print(lst);
```

- A lista inicial de valores (1,2,3) é exibida, em seguida uma nova lista é referenciada e exibida (4,5,6), e finalmente o primeiro elemento é atualizado, e a lista é exibida novamente (666,5,6). No entanto o que ocorreria se movêssemos a linha lst[0] = 666; para antes da reatribuição de lst na terceira linha? Veremos uma exceção, pois estaremos tentando alterar uma lista que foi marcada como const.

# Dart: constantes

- Variáveis e outros identificadores podem começar com uma letra ou *underscore* seguido por qualquer combinação de letras e números. Os identificadores que começam com um *underscore* têm um significado especial: é privado da biblioteca (ou classe) em que se encontra.
- Dart não tem palavras-chave de visibilidade como as palavras public e private, que existem em linguagens como Java, mas um *underscore* no início tem o mesmo significado que private tem em Java e em outras linguagens.

# Dart: *types*

- Dart é uma linguagem fortemente tipada, porém não é preciso explicitar os tipos. Eles são opcionais, e isso ocorre porque o Dart executa a inferência de tipos quando suas anotações (declarações) não estão presentes.


- Strings podem incluir expressões com o uso da sintaxe ${expressão}. Se a expressão referenciar um identificador, podemos omitir as chaves. Assim:

```
String n1 = "Bahamas";
String n2 = "${n1} vamos nós!";
String n3 = '$n1 vamos nós!';
print (n2);
print (n3);
```

# Dart: *types*

- Podemos ver as aspas duplas e simples aqui, além dos dois tipos de expressões (ou tokens);
- A concatenação de Strings pode usar o operador +, como ocorre em outras linguagens, ou strings literais adjacentes, assim:

```
return “Hanks,” “Tom”;
```

# Dart: valores numéricos

- Os valores numéricos inteiros são de tipo int. o intervalo de valores é $-2^{63}$ a $+2^{63}-1$ na VM Dart (passará a ser o intervalo de números JavaScript quando o Dart for compilado para essa linguagem e nunca será maior do que 64 bits, dependendo da plataforma).
- Um número de ponto flutuante de precisão dupla, como especificado pelo padrão IEEE 754, tem tipo double.
- Tanto int quanto double são subclasses de num, logo podemos definir uma variável como num w = 5; ou num x = 5.5; assim como poderia usar int y = 5; ou double z = 5.5; e o Dart saberá que x é um double baseado em seu valor da mesma forma que sabe que z também o é porque foi especificado.
- Um valor numérico pode ser transformado em uma string com o uso do método toString() das classes int e double:

```
int i = 5;
double d = 5.5;
String si = i.toString();
String sd = d.toString();
print(i);
```

# Dart: valores numéricos

```
print(d);
print(si);
print(sd);
```

- E uma *string* pode ser transformada em um número com o método parse() das classes int e double:

```
String si = "5";
String sd = "5.5";
int i = int.parse(si);
double d = double.parse(sd);
print(si);
print(sd);
print(i);
print(d);
```

# Dart: valores booleanos

- Os valores booleanos são de tipo bool e só dois objetos têm esse tipo de valor: as palavras-chave true e false (que são constantes do tempo de compilação).
- A segurança de tipos do Dart não nos permite escrever códigos desta forma:

```
if (alguma_variavel_nao_booleana)
```

Em vez disso, escrevemos:

```
if (alguma_variavel_nao_booleana.algumMetodo())
```

- A avaliação de uma instrução lógica não pode ter um valor considerado verdadeiro quando avaliado em um contexto booleano, como em algumas linguagens, como JavaScript. No Dart, ela deve sempre resultar em um dos valores bool.

# Dart: listas e mapas

- A classe List do Dart é semelhante ao *array* da maioria das linguagens. Sua instância é uma lista de valores que são definidos com sintaxe idêntica à do JavaScript:

```
List lst = [1,2,3];
```

Nota: Geralmente escrevemos list, assim como set e map, ao referenciar uma instância das classes Map, Set ou List, e só usamos maiúsculas ao referenciar a classe real. Também podemos usar estes formatos:

```
var lst1 = [1,2,3];
Object lst2 = [1,2,3];
```

- Uma lista usa um esquema de indexação baseado em zero, logo, list.length-1 fornece o índice do último elemento. Podemos acessar os elementos pelo índice:

```
print (lst[1]);
```

# Dart: listas e mapas

- A lista tem vários métodos disponíveis. Seguem alguns:

```
List lst = [9,2,13];
lst.add(4);
lst.sort((a,b) => a.compareTo(b));
lst.removeLast();
print(lst.indexOf(4));
print(lst);
```

- O Dart também oferece uma classe Set, que é semelhante a List, mas é uma lista não ordenada, o que significa que não podemos recuperar elementos pelo índice. É preciso usar os métodos contains() e containsAll():

```
Set cookies = Set();
cookies.addAll(['mel','chocolate','manteiga']);
cookies.add('mel');
cookies.remove('chocolate');
print(cookies);
print(cookies.contains('mel'));
print(cookies.containsAll(['chocolate','manteiga']));
```

# Dart: listas e mapas

- A chamada a contains() retorna true, enquanto a chamada a containsAll() retorna false já que chocolate foi removido com remove(). Observe que adicionar um valor que já existe no conjunto não causa problemas.

- A linguagem também tem uma classe Map, às vezes chamada de dicionário ou hash ou objeto literal em JavaScript, cuja instância pode ser criada de algumas maneiras:

```
 var atores = {
  'Bruce Lee' : 'O vôo do dragão',
  'Steven Seagal' : 'Fúria mortal'
};
print(atores);
```

```
 var atrizes = new Map();
atrizes['Scarlett Johansson'] = 'Lucy';
atrizes['Zoë Saldaña'] = 'Avatar';
print(atrizes);
```

# Dart: listas e mapas

```
var filmes = Map<String, int>();
filmes['Homem de ferro'] = 3;
filmes['Thor'] = 3;
print(filmes);

print(atores['Ryan Reynolds']);
print(atrizes['Elizabeth Olsen']);
filmes.remove('Thor');
print(filmes);
print(atores.keys);
print(atrizes.values);

Map sequels = { };
print(sequels.isEmpty);
sequels['The Winter Soldier  '] = 2;
sequels['Civil War  '] = 3;
sequels.forEach((k,v){
  print(k + 'sequel #' + v.toString());
});
```

# Dart: listas e mapas

- O primeiro mapa, atores, é criado com o uso de chaves e com dados definidos imediatamente dentro dele.
- O segundo, atrizes, usa a palavra new para criar uma nova instância de Map explicitamente. Os elementos são adicionados a ele com o uso da notação de colchete, em que o valor dentro do colchete é a *chave* e o valor após o sinal de igualdade é o mapeamento para essa *chave*.
- A terceira versão define tipos para as *chaves* e valores de um mapa. Nesse caso, se usar:

```
filmes[3] = 'Iron Man';
```

verá um erro de compilação porque 3 é um int, mas o tipo da *chave* foi definido como String e, da mesma forma, o tipo do *valor* foi definido como int, mas estamos tentando inserir uma String.

- O método remove() remove um elemento de um mapa.
- Podemos obter uma lista das *chaves* e *valores* lendo os atributos keys e values, o que na verdade significa chamar um método getter, ainda que não haja parênteses como normalmente ocorre após uma chamada de método.
- O método isEmpty() informa se o mapa está ou não vazio (também há um método isNoEmpty()).
- Um mapa também fornece os métodos contains() e containsAll(), como ocorre com a lista.
- O método forEach() permite executar uma função arbitrária para cada elemento do mapa (a função fornecida receberá a *chave* e o *valor*).

# Dart: listas e mapas

- Há um tipo dynamic especial, o qual desativa o sistema de tipos do Dart. Suponha que:

```
Object obj = some_object;
```

- Podemos chamar alguns métodos em obj como toString() e hashCode(), pois são definidos pela classe Object, da qual todos os objetos são estendidos. Se tentarmos chamar obj.fakeMethod(), verá um aviso, pois o Dart pode ver, no tempo de compilação, que fakeMethod() não é um método da classe Object, ou da classe da qual some_object é instância. Contudo, se escrevêssemos

```
dynamic obj = some_object;
```

- e se escrevêssemos obj.fakeMethod(), não veríamos um aviso no tempo de compilação, mas um erro no tempo de execução.
- Pense em dynamic como uma maneira de dizer ao Dart: "*sou eu que mando aqui e sei o que estou fazendo*".
- Normalmente o tipo dynamic é usado com valores de retorno de atividades interoperacionais.

# ‘*as*’ e ‘*is*’

- A palavra-chave ‘is’ permite determinar se uma referência é de um tipo específico (basicamente se implementa determinada interface) e ‘as’ permite tratar uma referência de tipo específico como outra, supondo que seja uma superclasse (é aquela classe que será derivada, é a classe mãe ou base como também é chamada). Por exemplo:

```
if (shape is Circle) {
  print(circle.circumference);
}
```

- Esse código só exibirá (com print(), que exibe conteúdo no console) a circunferência se o objeto referenciado por shape for de tipo Circle.
- Por outro lado podemos usar ‘as’ assim:

```
(shape as Circle).circumference = 20;
```

- Nesse caso, se shape for de tipo Circle, o código funcionará como esperado e, se shape puder ser convertido para Circle, ele também funcionará (shape poderia ser de tipo Oval, que é uma subclasse (é a classe derivada, é a classe filha que foi herdada de uma superclasse) de Circle, por exemplo).
- Observe, no entanto, que, no exemplo de ‘is’, nada acontecerá se shape não for de tipo Circle, mas, no exemplo de ‘as’, uma exceção será lançada se shape não puder ser convertido para Circle.

# Controle sobre a UI: exemplo básico

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

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

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter UI Controls',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: MyHomePage(title: 'Flutter UI Controls Home Page'),
    );
  }
}

class MyHomePage extends StatefulWidget {
  final String title;
  const MyHomePage({Key? key, required this.title}) : super(key: key);

  @override
  _MyHomePageState createState() => _MyHomePageState();
}
```

1

# Controle sobre a UI: exemplo básico

```dart
class _MyHomePageState extends State<MyHomePage> {
  final GlobalKey<ScaffoldState> _scaffoldKey = new GlobalKey<ScaffoldState>();
  String? _dropdownButtonValue = 'One';
  String? _popupMenuButtonValue = 'One';
  bool? _checkboxValue = true;
  String? _radioBoxValue = 'One';
  double _sliderValue = 10;
  bool _switchValue = false;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      key: _scaffoldKey,
      appBar: AppBar(
        title: Text(widget.title),
      ),
      body: Container(
        padding: EdgeInsets.only(left: 10, right: 10),
        child: SingleChildScrollView(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.start,
            crossAxisAlignment: CrossAxisAlignment.start,
            children: <Widget>[
              // Buttons
              Column(
                mainAxisAlignment: MainAxisAlignment.start,
                crossAxisAlignment: CrossAxisAlignment.start,
```

# Controle sobre a UI: exemplo básico

```
children: <Widget>[
  _GroupText('Buttons'),
  ButtonBar(
    alignment: MainAxisAlignment.start,
    mainAxisSize: MainAxisSize.min,
    buttonMinWidth: 200,
    buttonHeight: 30,
    buttonAlignedDropdown: true,
    layoutBehavior: ButtonBarLayoutBehavior.padded,
    buttonPadding: EdgeInsets.symmetric(vertical: 10),
    children: <Widget>[
      RaisedButton(
        onPressed: () => _showToast('Clicked on RaisedButton'),
        child: Text('Raised Button'),
      ),
      FlatButton(
        onPressed: () => _showToast('Clicked on FlatButton'),
        child: Text('Flat Button'),
      ),
      OutlineButton(
        onPressed: () => _showToast('Clicked on OutlineButton'),
        child: Text('OutlineButton'),
      ),
      Row(
        children: <Widget>[
          Text('IconButton: '),
          IconButton(
```

# Controle sobre a UI: exemplo básico



```
          onPressed: () =>
              _showToast('Clicked on IconButton'),
          icon: Icon(Icons.build),
        ),
      ],
    ),
    Row(
      children: <Widget>[
        Text('DropdownButton: '),
        DropdownButton(
            value: _dropdownButtonValue,
            onChanged: (String? value) {
              setState(() {
                _dropdownButtonValue = value;
              });
              _showToast(
                  'Changed value of dropdown button to $value');
            },
            items: ['One', 'Two', 'Three', 'Four']
                .map<DropdownMenuItem<String>>(
                    (String value) =>
                        DropdownMenuItem<String>(
                          value: value,
                          child: Text(value),
                        ))
                .toList()),
      ],
    ),
```

# Controle sobre a UI: exemplo básico



```
Row(
  children: <Widget>[
    Text('PopupMenuButton: '),
    Text(_popupMenuButtonValue!),
    PopupMenuButton<String>(
        onSelected: (String result) {
          setState(() {
            _popupMenuButtonValue = result;
          });
          _showToast(
              'Selected \'$result\' item on PopupMenuButton');
        },
        itemBuilder: (BuildContext context) =>
            <PopupMenuEntry<String>>[
              const PopupMenuItem<String>(
                value: 'One',
                child: Text('One'),
              ),
              const PopupMenuItem<String>(
                value: 'Two',
                child: Text('Two'),
              ),
              const PopupMenuItem<String>(
                value: 'Three',
                child: Text('Three'),
              ),
```

# Controle sobre a UI: exemplo básico



```
                        const PopupMenuItem<String>(
                          value: 'Four',
                          child: Text('Four'),
                        ),
                      ]),
                ],
              ),
            ],
          )
        ],
      ),
      _SpaceLine(),
      // Checkbox
      Column(
          mainAxisAlignment: MainAxisAlignment.start,
          crossAxisAlignment: CrossAxisAlignment.start,
          children: <Widget>[
            _GroupText('Checkbox'),
            Row(
              children: <Widget>[
                Text('Simple checkbox'),
                Checkbox(
                    value: _checkboxValue,
                    onChanged: (bool? newValue) {
                      setState(() {
                        _checkboxValue = newValue;
                      });
```

# Controle sobre a UI: exemplo básico



```
                    _showToast(
                        'Changed value of checkbox to $_checkboxValue');
                  }),
            ],
          )
        ]),
    _SpaceLine(),
    // Radio[Box]
    Column(
      mainAxisAlignment: MainAxisAlignment.start,
      crossAxisAlignment: CrossAxisAlignment.start,
      children: <Widget>[
        _GroupText('Radio[Box]'),
        RadioListTile(
            title: const Text('One'),
            value: 'One',
            groupValue: _radioBoxValue,
            onChanged: (String? value) {
              setState(() {
                _radioBoxValue = value;
              });
              _showToast('Changed value of Radio[Box] to $value');
            }),
        RadioListTile(
            title: const Text('Two'),
            value: 'Two',
            groupValue: _radioBoxValue,
            onChanged: (String? value) {
```

# Controle sobre a UI: exemplo básico



```
                setState(() {
                  _radioBoxValue = value;
                });
                _showToast('Changed value of Radio[Box] to $value');
              }),
          RadioListTile(
              title: const Text('Three'),
              value: 'Three',
              groupValue: _radioBoxValue,
              onChanged: (String? value) {
                setState(() {
                  _radioBoxValue = value;
                });
                _showToast('Changed value of Radio[Box] to $value');
              }),
          RadioListTile(
              title: const Text('Four'),
              value: 'Four',
              groupValue: _radioBoxValue,
              onChanged: (String? value) {
                setState(() {
                  _radioBoxValue = value;
                });
                _showToast('Changed value of Radio[Box] to $value');
              }),
        ],
      ),
```

# Controle sobre a UI: exemplo básico



```
_SpaceLine(),
// Slider
Column(
  mainAxisAlignment: MainAxisAlignment.start,
  crossAxisAlignment: CrossAxisAlignment.start,
  children: <Widget>[
    _GroupText('Slider'),
    Slider(
      value: _sliderValue,
      onChanged: (newValue) {
        setState(() {
          _sliderValue = newValue;
        });
      },
      min: 0,
      max: 100,
      divisions: 50,
      label: _sliderValue.toInt().toString(),
    ),
  ],
),
_SpaceLine(),
// Switch
Column(
  mainAxisAlignment: MainAxisAlignment.start,
  crossAxisAlignment: CrossAxisAlignment.start,
  children: <Widget>[
    _GroupText('Switch'),
```

# Controle sobre a UI: exemplo básico



```
      Row(
        children: <Widget>[
          Text('Simple switch'),
          Switch(
            value: _switchValue,
            onChanged: (newValue) {
              setState(() {
                _switchValue = newValue;
              });
              _showToast('Changed value of Switch to $newValue');
            },
          )
        ],
      ),
    ],
  ),
  _SpaceLine(),
  // TextField
  Column(
    mainAxisAlignment: MainAxisAlignment.start,
    crossAxisAlignment: CrossAxisAlignment.start,
    children: <Widget>[
      _GroupText('TextField'),
      TextField(
        obscureText: false,
        decoration: InputDecoration(
          border: OutlineInputBorder(),
          labelText: 'TextField',
```

# Controle sobre a UI: exemplo básico



```
        ),
      ),
    ],
  ),
  _SpaceLine(),
  // DateTimePicker
  Column(
    mainAxisAlignment: MainAxisAlignment.start,
    crossAxisAlignment: CrossAxisAlignment.start,
    children: <Widget>[
      _GroupText('DateTimePicker'),
      OutlinedButton(
        onPressed: () {
          showDatePicker(
            context: context,
            initialDate: DateTime.now(),
            firstDate: DateTime(1970),
            lastDate: DateTime.now(),
          ).then((value) {
            _showToast('Selected date $value');
          });
        },
        child: Text('Open DatePicker'),
      ),
      OutlinedButton(
        onPressed: () {
          showTimePicker(
            context: context,
```

# Controle sobre a UI: exemplo básico



```
                    initialTime: TimeOfDay.now(),
                  ).then((value) {
                    _showToast('Selected time $value');
                  });
                },
                child: Text('Open TimePicker'),
              ),
            ],
          ),
          SizedBox(
            height: 50,
          )
        ],
      ),
    ),
    floatingActionButton: FloatingActionButton(
      onPressed: () => _showToast('Clicked on float action button'),
      tooltip: 'Increment',
      child: Icon(Icons.add),
    ),
  );
}

void _showToast(String text) {}

RaisedButton({required void Function() onPressed, required Text child}) {}
```

# Controle sobre a UI: exemplo básico



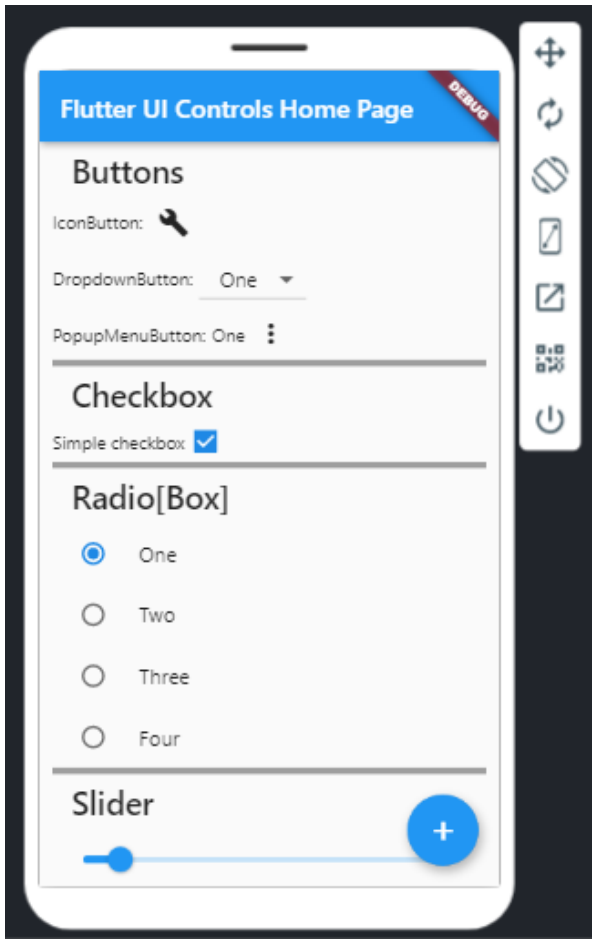
```
  FlatButton({required void Function() onPressed, required Text child}) {}

  OutlineButton({required void Function() onPressed, required Text child}) {}
}

class _GroupText extends StatelessWidget {
  final String text;

  const _GroupText(this.text);

  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: EdgeInsets.symmetric(vertical: 5, horizontal: 15),
      child: Text(
        text,
        style: TextStyle(fontSize: 25, fontWeight: FontWeight.w500),
      ),
    );
  }
}
```

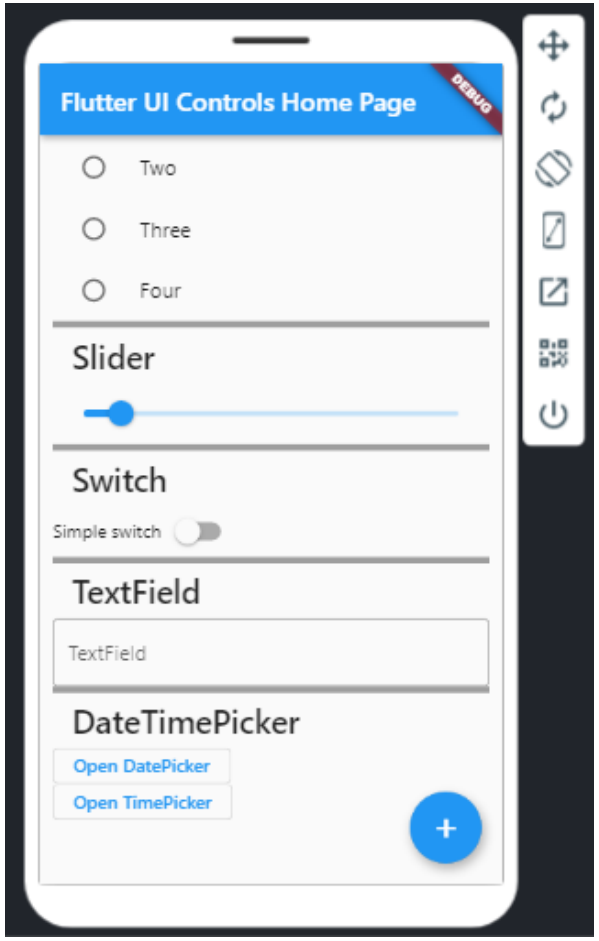
# Controle sobre a UI: exemplo básico



```
class _SpaceLine extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return SizedBox(
      height: 5,
      child: Container(
        color: Colors.grey,
      ),
    );
  }
}
```

# Controle sobre a UI: exemplo básico



# Navegação: exemplo básico

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

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

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Navigation Over Screens',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
//      home: MainPage(),

      // Declare routes
      routes: {
        // Main initial route
        '/': (context) => MainPage(),
        // Second route
        '/second': (context) => SecondPage(),
      },
      initialRoute: '/',
    );
  }
}
```

1

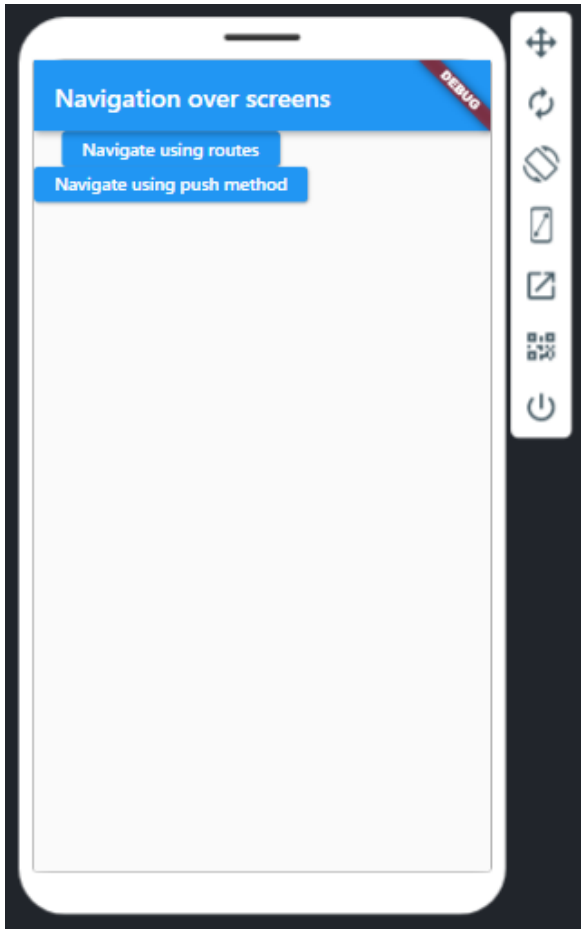
# Navegação: exemplo básico

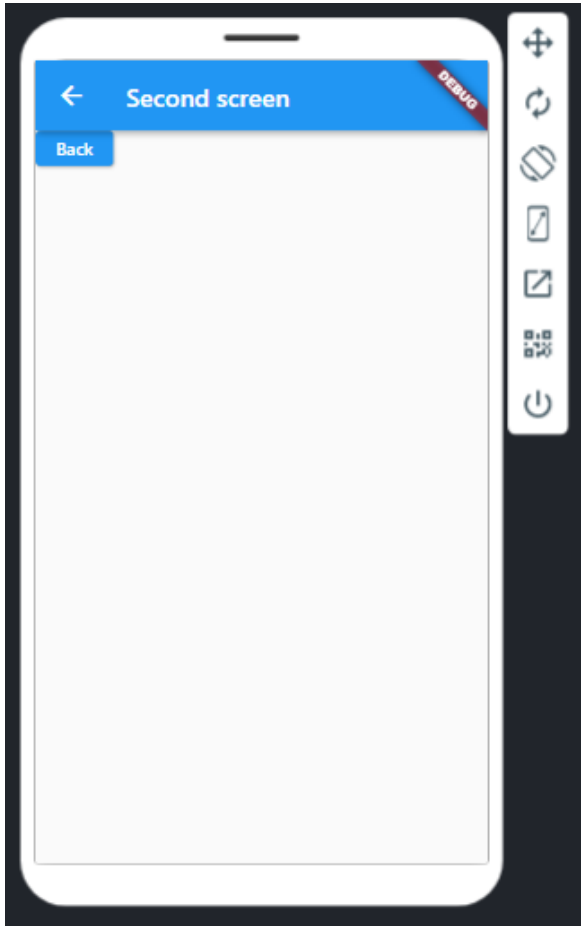
```
class MainPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) =>
      Scaffold(
        appBar: AppBar(
          title: Text('Navigation over screens'),
        ),
        body: Container(
          child: Column(
            children: <Widget>[
              // Navigate using declared route name
              ElevatedButton(
                onPressed: () => Navigator.pushNamed(context, '/second'),
                child: Text('Navigate using routes'),
              ),
              // Navigate using simple push method
              ElevatedButton(
                onPressed: () =>
                    Navigator.push(
                      context,
                      MaterialPageRoute(builder: (context) => SecondPage()),
                    ),
```

# Navegação: exemplo básico

```
                child: Text('Navigate using push method'),
              )
            ],
          ),
        ),
      );
}

class SecondPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Second screen'),
      ),
      body: Container(
        child: ElevatedButton(
          onPressed: () => Navigator.pop(context),
          child: Text('Back'),
        ),
      ),
    );
  }
}
```

# Navegação: exemplo básico



# Dúvidas!?