

Dart/Flutter



Informações



Próxima semana não há aulas - Tecweb

- Quem participar no Cody (concurso de programação) terá a nota de participação equivalente a um quiz teórico

Dart

Linguagem criada pela Google em 2011, de alto nível, similar ao Kotlin e Swift

Ganhou popularidade a partir de 2017, quando foi lançado o Flutter

[illegible]

Dart

Dart

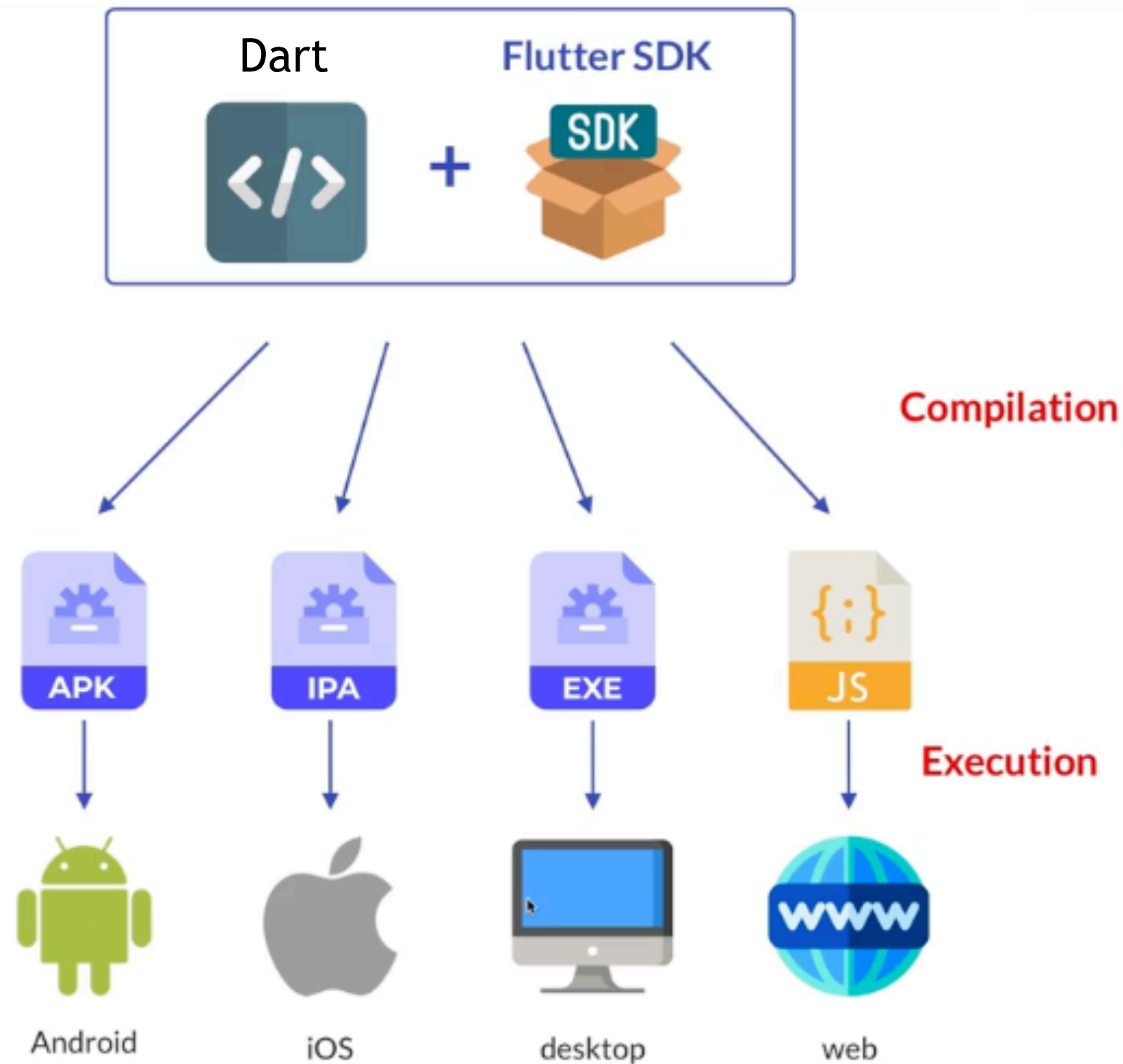
Rapidez de execução ▲ ▲

Compilada para nativo na plataforma final (AOT compiler)

Rapidez de desenvolvimento ▲ ▲

Interpretada através de uma Dart VM (JIT compiler), permite hot reload

Dart é multi-plataforma



Dart é multi-paradigma

Paradigma imperativo

função main() inicia a execução sequencial de instruções

Paradigma orientado a objetos

classes, encapsulamento, herança

Paradigma funcional

funções como variáveis, lambdas

Ferramentas

- Dart SDK (compilador, dart vm)

<https://dart.dev/get-dart>

Quando se instala o Flutter, vem incluído:

<pasta do flutter>/bin/cache/dart-sdk

- Dartpad - “IDE” online

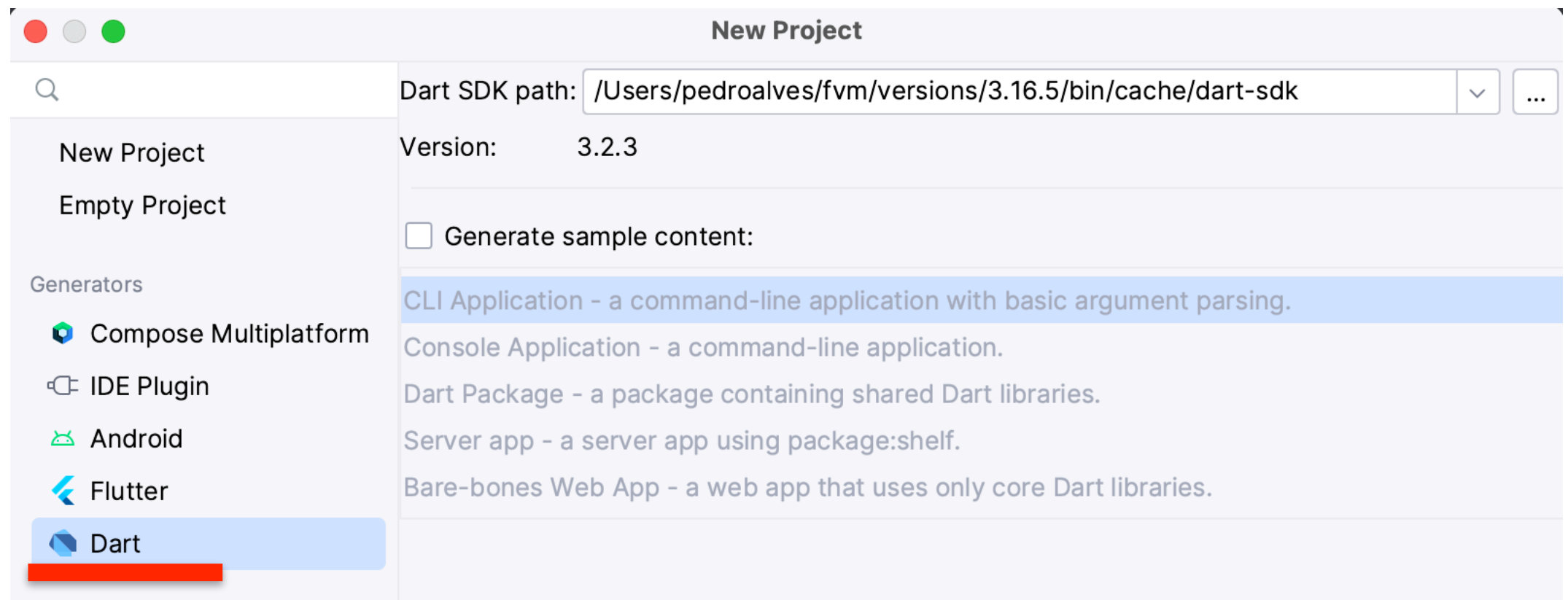
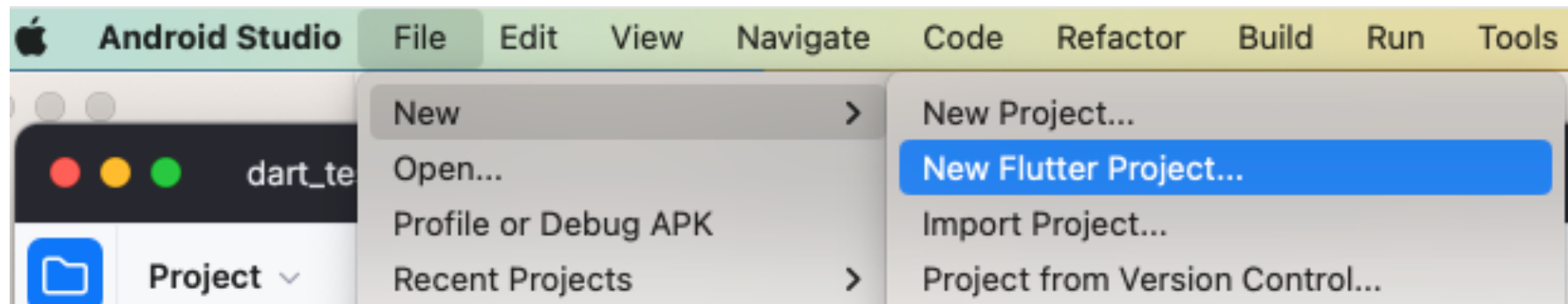
<https://dartpad.dev/>

limitado, útil para pequenas experiências

- IntelliJ, Android Studio, VS Code

Os principais IDEs trazem plugins para Dart

Criar um projeto Dart no Android Studio



Exemplo Dart

live coding

Exemplo Dart

class abstract, não pode ser instanciada, apenas os filhos

```
abstract class Widget {  
  String? key;  
  bool visible = true;  
  String? additionalInfo;
```

arguments “named” (dentro de chavetas)

```
Widget({this.key, this.additionalInfo});
```

```
String build();  
}
```

método abstrato, tem que ser overridden pelas sub-classes

Exemplo Dart

herda de Widget

construtor inicializa os argumentos do pai
...e os próprios. neste caso é obrigatório

```
class Text extends Widget {  
  String text;  
  
  Text({super.key, super.additionalInfo, required this.text});  
  
  @override  
  String build() {  
    return 'building text: $text';  
  }  
}
```

obrigatório implementar o build()

Exemplo Dart

```
class Button extends Widget {  
  Function onPressed;  
  Widget child;
```

a variável `onPressed` é do tipo `Function`,
representa um bloco de código que é executado
quando se prime o botão

utilização de composição: o “conteúdo” do
botão é ele próprio um widget

```
  Button({super.key, super.additionalInfo,  
        required this.onPressed,  
        required this.child});
```

```
@override  
String build() {  
  return 'building button with child: ${child.build()}';  
}  
}
```

para construir o `Button`, tem que construir o filho
(por composição)

Exemplo Dart

composição aplicada a uma lista de widgets que são alinhados na horizontal

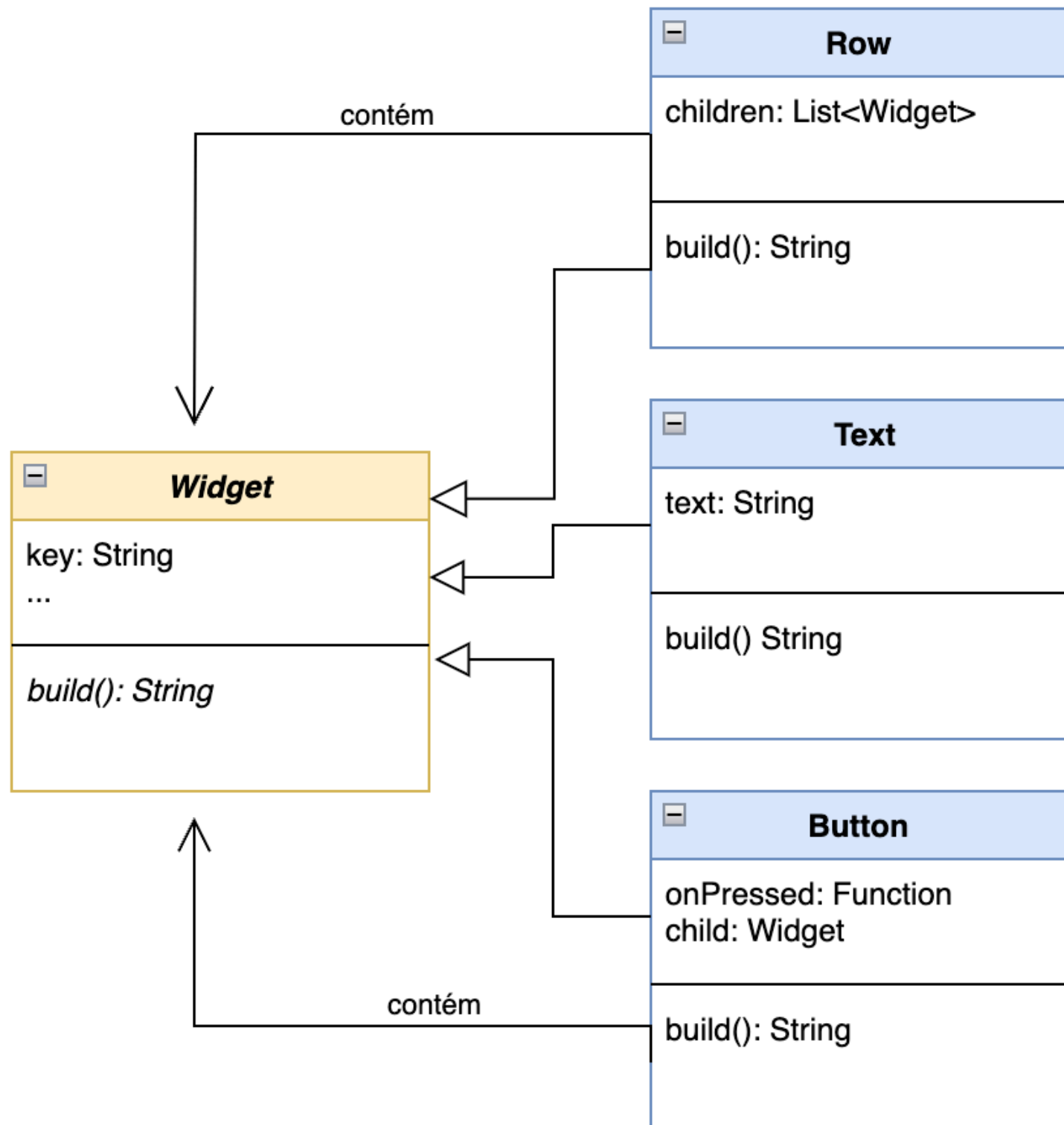
```
class Row extends Widget {  
  List<Widget> children;  
  
  Row({required this.children});  
  
  @override  
  String build() {  
    String result = 'build row with children:\n';  
    for (var child in children) {  
      result += '\t' + child.build() + '\n';  
    }  
    return result;  
  }  
}
```

Exemplo Dart

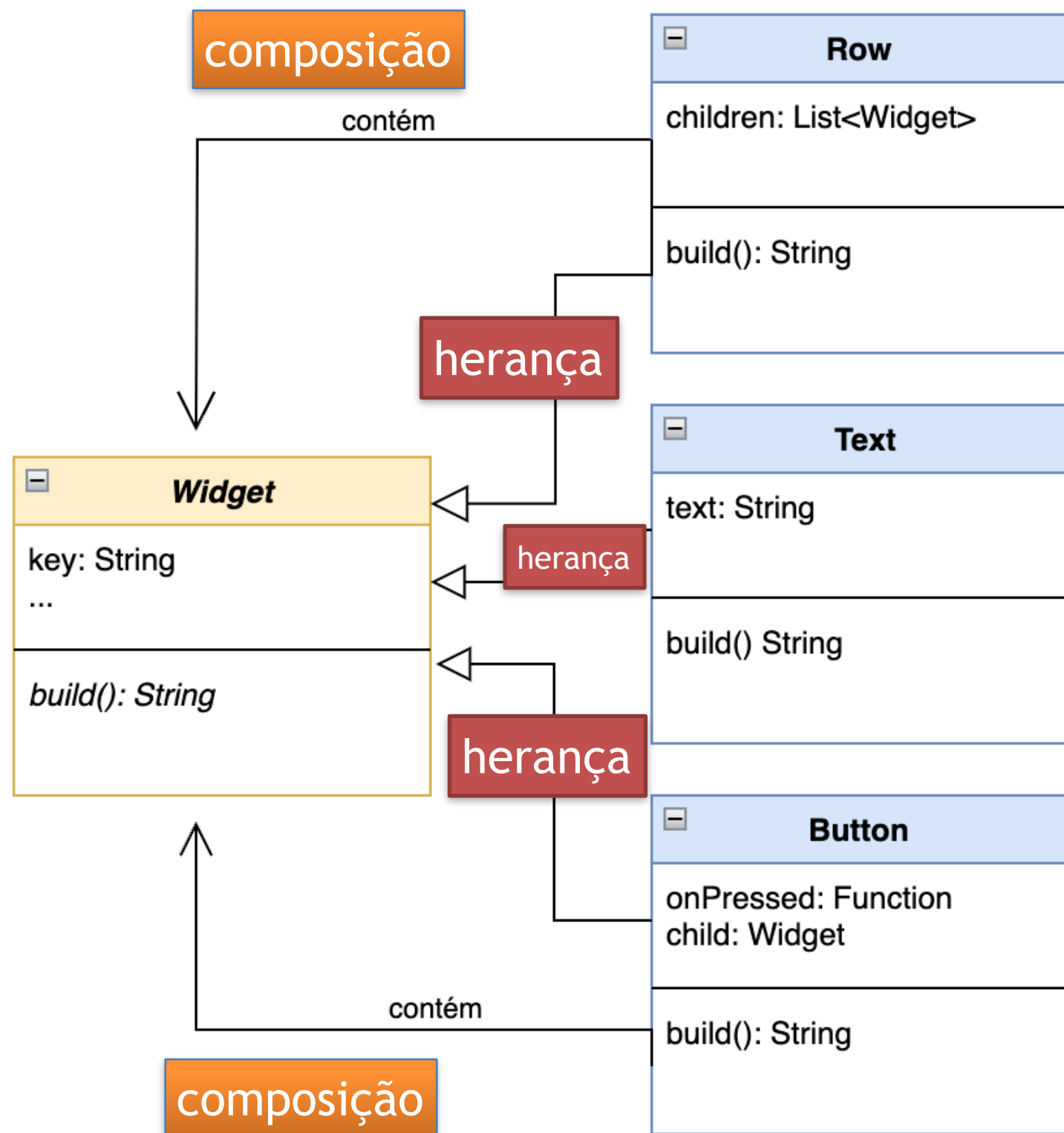
- 1 - é possível combinar widgets diferente numa única lista desde que herdem todos de Widget (polimorfismo)
- 2 - por composição, consegue-se combinar widgets

```
main() {  
  List<Widget> widgets = [  
    Row(children: [  
      Text(key: 'texto', text: 'Um texto qualquer'),  
      Button(  
        onPressed: () {  
          print('premi botão');  
        },  
        child: Text(text: 'Gravar'))  
    ])  
  ];  
  
  for (var widget in widgets) {  
    print(widget.build());  
  }  
}
```


Exemplo Dart



Exemplo Dart



Listas e Dicionários

```
var list = [1, 2, 3];
```

```
List<int> list = [1, 2, 3];
```

```
print(list[2]); // 3  
print(list.length); // 3
```

```
Map<String,String> alunos = {  
    'a22101234': 'José',  
    'a22104321': 'Sara'  
};
```

```
print(alunos['a22101234']);
```


Trailing commas

O Dart permite que o último elemento de uma lista/dicionário termine com ','
Isso terá efeito na formatação


```
var list = [1, 2, 3];
```

```
var alunos =  
    {'a22101234': 'José', 'a22104321': 'Sara'};
```

```
var list = [  
    1,  
    2,  
    3,  
];
```



```
var alunos = {  
    'a22101234': 'José',  
    'a22104321': 'Sara',  
};
```



Records

Permitem guardar vários valores numa única variável. A diferença para as listas é que:

- não permitem adicionar/remover valores
- podem guardar valores de tipos diferentes

```
var aluno = (numero: 'a22101234', nome: 'Sara', idade: 21);
```

```
print(aluno.numero);  
print(aluno.nome);  
print(aluno.idade);
```

Functional Dart

```
void main() {  
  [1,2,3,4,5].forEach((int numero) => print(numero));  
}
```

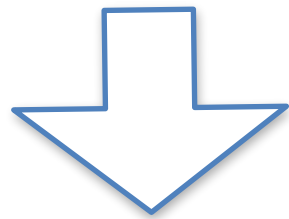
Lambdas em Dart

```
int incrementa(int numero) {  
    return numero + 1;  
}
```

```
print([1, 2, 3, 4, 5].map(incrementa));
```


Lambdas em Dart

```
int incrementa(int numero) {  
    return numero + 1;  
}
```

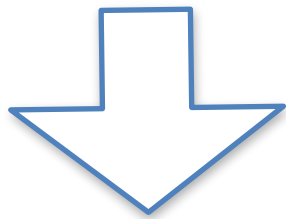


```
int incrementa(int numero) => numero + 1;
```

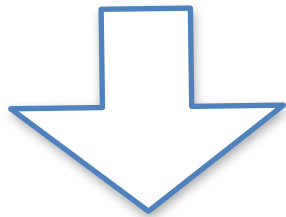
```
print([1, 2, 3, 4, 5].map(incrementa));
```

Lambdas em Dart

```
int incrementa(int numero) {  
    return numero + 1;  
}
```



```
int incrementa(int numero) => numero + 1;
```

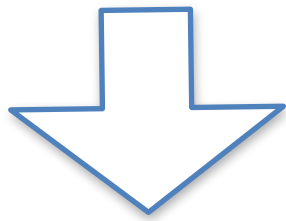


```
(int numero) => numero + 1    (lambda)
```

```
print([1, 2, 3, 4, 5].map((int numero) => numero + 1));
```

Lambdas em Dart

```
(int numero) => numero + 1 (lambda)
```



```
(int numero) {  
  return numero + 1;  
}) (lambda com bloco de código)
```

```
print([1,2,3,4,5].map((int numero) {  
  return numero + 1;  
}));
```

OOP em Dart

```
class Point {  
  int x = 0;  
  int y = 0;  
  String? color;  
}
```

Nota: Todos os atributos têm que estar inicializados ou ser nullable

```
(...)  
var point = Point();  
point.x = 5;  
point.y = 3;
```

OOP em Dart

Construtor clássico

```
class Point {  
  int x;  
  int y;  
  String? color;  
  
  Point(int x, int y, String color) {  
    this.x = x;  
    this.y = y;  
    this.color = color;  
  }  
}
```

Ao criar o construtor, já não é necessário inicializar os atributos



OOP em Dart

Construtor “compacto” posicional

```
class Point {  
  int x;  
  int y;  
  String? color;  
  
  Point(this.x, this.y, this.color);  
}
```

```
var point = Point(3, 4, null);
```

OOP em Dart

Construtor “compacto” “named”

```
class Point {  
  int x;  
  int y;  
  String? color;  
  
  Point({ required this.x, required this.y, this.color });  
}  
  
var point = Point(x: 3, y: 4);
```


OOP em Dart

```
class Point {  
  int x;  
  int y;  
  String? color;  
  
  Point(this.x, this.y, this.color);  
}
```

E se precisar de vários construtores?

OOP em Dart

Posso acrescentar “named constructors”

```
class Point {  
  int x;  
  int y;  
  String? color;  
  
  Point(this.x, this.y, this.color);  
  
  Point.xy(this.x, this.y);  
}
```

...

```
var point = Point.xy(3, 4);
```

OOP em Dart

Se quiser criar o objeto a partir de outras estruturas (ex: Map)

```
class Point {  
  int x;  
  int y;  
  String? color;  
  
  Point.xy(this.x, this.y);  
  
  Point.fromMap(Map<String,int> map):  
    x = map['x']??0,  
    y = map['y']??0;  
}
```

?? é “Elvis operador” do Dart

if (map['x'] != null) map['x'] else 0

OOP em Dart

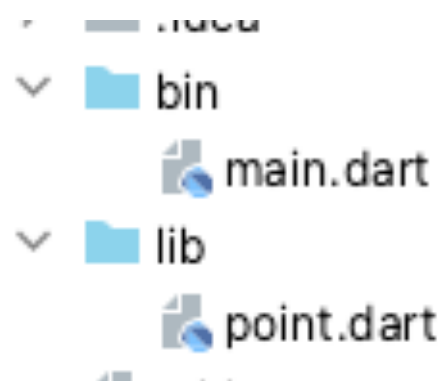
Ou usar um factory constructor que retorna o novo objeto

```
class Point {  
  int x;  
  int y;  
  String? color;  
  
  Point.xy(this.x, this.y);  
  
  factory Point.fromMap(Map<String,int> map) {  
    return Point.xy(map['x']??0, map['y']??0);  
  }  
}
```

OOP em Dart

Em Dart as variáveis são “públicas” a menos que comecem por ‘_’.

Nesse caso ficam privadas mas continuam a ser visíveis dentro da mesma library (*)



```
void main() {
  var point = Point.xy(3, 4);
  print(point._x);
  point._x = 5;
}
```

Erro!!

```
class Point {
  int _x;
  int _y;

  Point.xy(this._x, this._y);
}
```

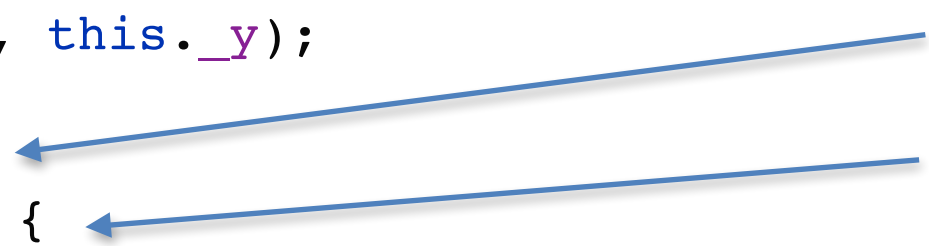
(*) cada ficheiro dart é uma library

OOP em Dart

Em Dart podem-se criar getters e setters para as variáveis “privadas”

```
class Point {  
  int _x;  
  int _y;  
  
  Point.xy(this._x, this._y);  
  
  int get x => _x;  
  set x(int value) {  
    if (value >= 0) {  
      _x = value;  
    } else {  
      throw Exception("Point.x must be positive");  
    }  
  }  
}
```

...



getter para _x
setter para _x

Herança em Dart

```
abstract class FiguraGeometrica {  
  int x;  
  int y;  
  
  FiguraGeometrica(this.x, this.y);  
  
  int calculaArea();  
}
```

```
class Rectangulo extends FiguraGeometrica {  
  
  int lado1;  
  int lado2;  
  
  Rectangulo(super.x, super.y, this.lado1, this.lado2);  
  
  @override  
  int calculaArea() {  
    return lado1 * lado2;  
  }  
}
```

```
class Circulo extends FiguraGeometrica {  
  
  int raio;  
  
  Circulo(super.x, super.y, this.raio);  
  
  @override  
  int calculaArea() {  
    return (raio * raio * pi).toInt();  
  }  
}
```

```
void main() {  
  
  var figuras = [  
    Rectangulo(3,4,20,10),  
    Circulo(6,7,5),  
  ];  
  
  figuras.forEach((figura) => print(figura.calculaArea()));  
}
```


Singleton em Dart

```
class FavoritesModel {  
  
  // Private constructor  
  FavoritesModel._();  
  
  // Singleton instance  
  static final FavoritesModel _instance = FavoritesModel._();  
  
  // Getter to access the singleton instance  
  static FavoritesModel get instance => _instance;  
  
  var favorites = <Quote>[ ];  
  
}  
  
(...)  
FavoritesModel.instance.favorites.add(...);
```

Cábula

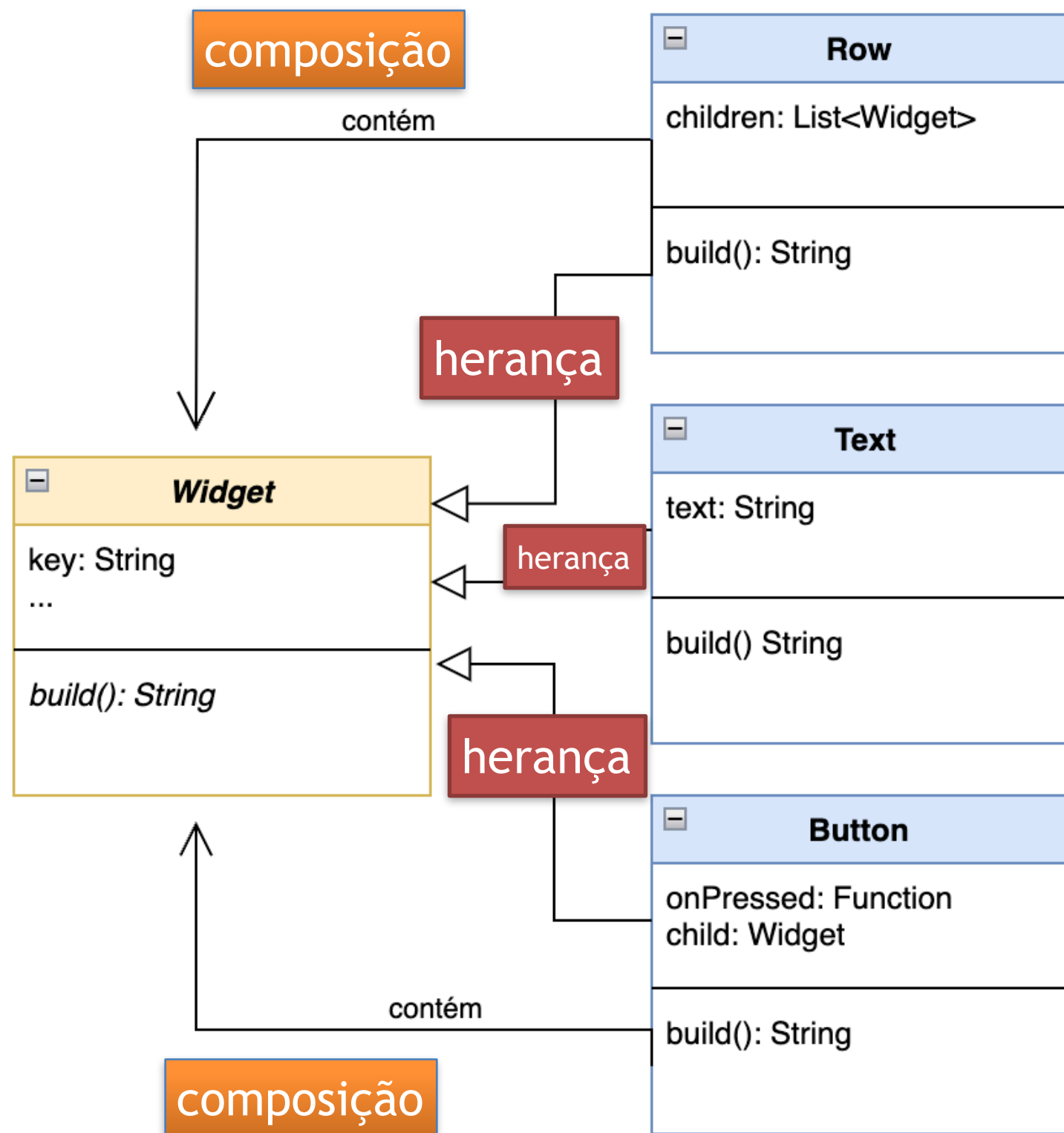
Cábula Kotlin-Dart disponível no Moodle

Unit testing em Dart

Junit	Dart Unit Tests
<code>assertEquals(number, 1)</code> <code>assertEquals(text, "hello")</code>	<code>expect(number, 1)</code> <code>expect(text, "hello")</code>
<code>assertNull(obj)</code> <code>assertNotNull(obj)</code>	<code>expect(obj, isNull)</code> <code>expect(obj, isNotNull)</code>
<code>assertTrue(expr)</code> <code>assertFalse(expr)</code>	<code>expect(expr, isTrue)</code> <code>expect(expr, isFalse)</code>

<https://cms.invertase.io/wp-content/uploads/2023/03/cheat-sheet.png>

Exemplo Dart



Flutter

Arquitetura baseada em composição de Widgets

A aplicação é ela própria um Widget, que contém widgets que contém widgets (composição)

```
void main() {  
  runApp(const MyApp());  
}  
  
class MyApp extends StatelessWidget {  
  const MyApp({super.key});  
  
  @override  
  Widget build(BuildContext context) {  
    return MaterialApp(  
      title: 'Awesome Quotes',  
      theme: themeLight(),  
      home: MainPage(),  
    );  
  }  
}
```

Widget

Widget

Widget

Quote

A little learning is a
dangerous thing.

- *Alexander Pope* -

 Like

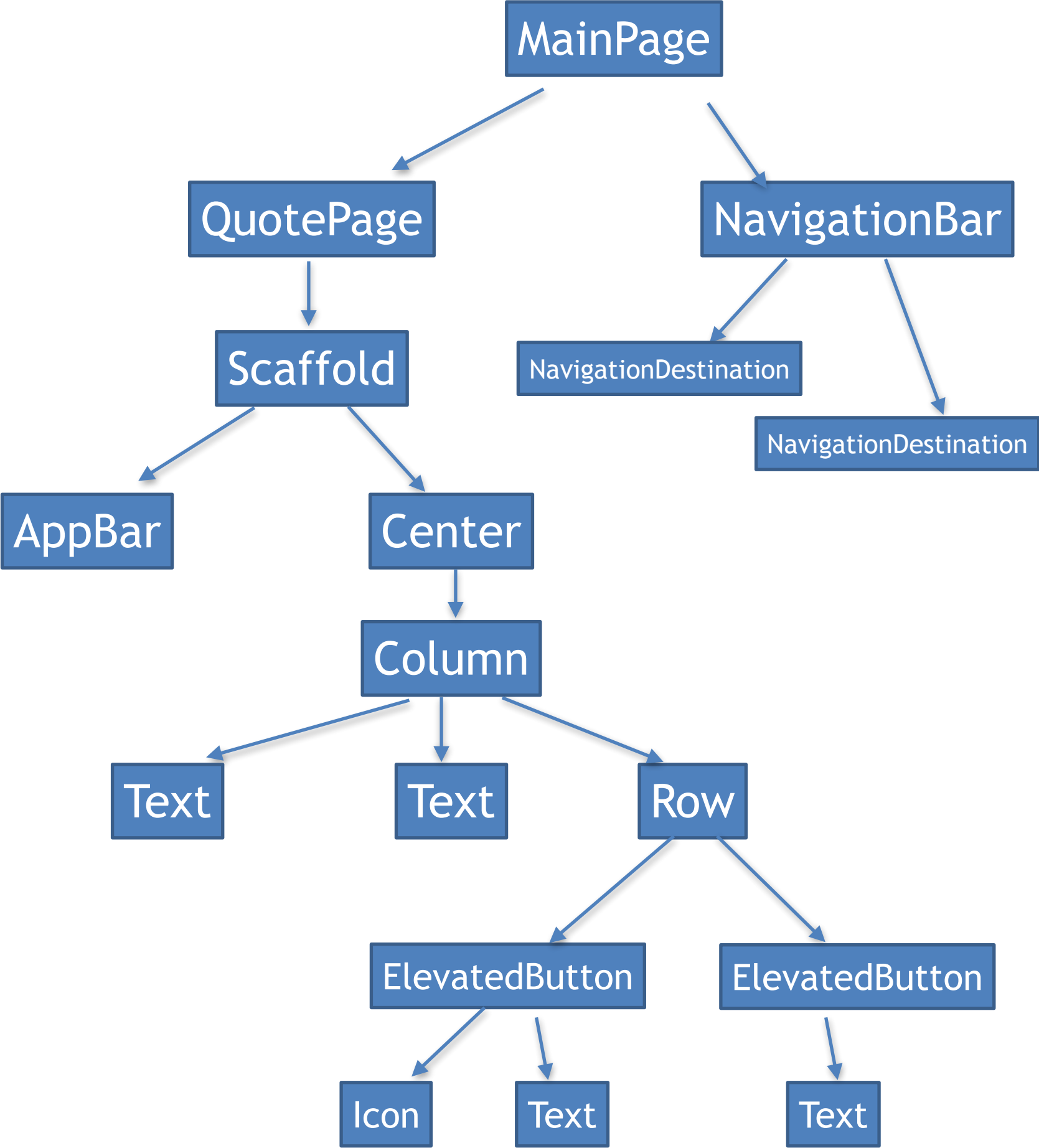
Next



Quote



Favorites



Material de estudo

- Desenvolvimento com widgets em Flutter - parte 1

<https://www.youtube.com/watch?v=tKTusGq46nQ>

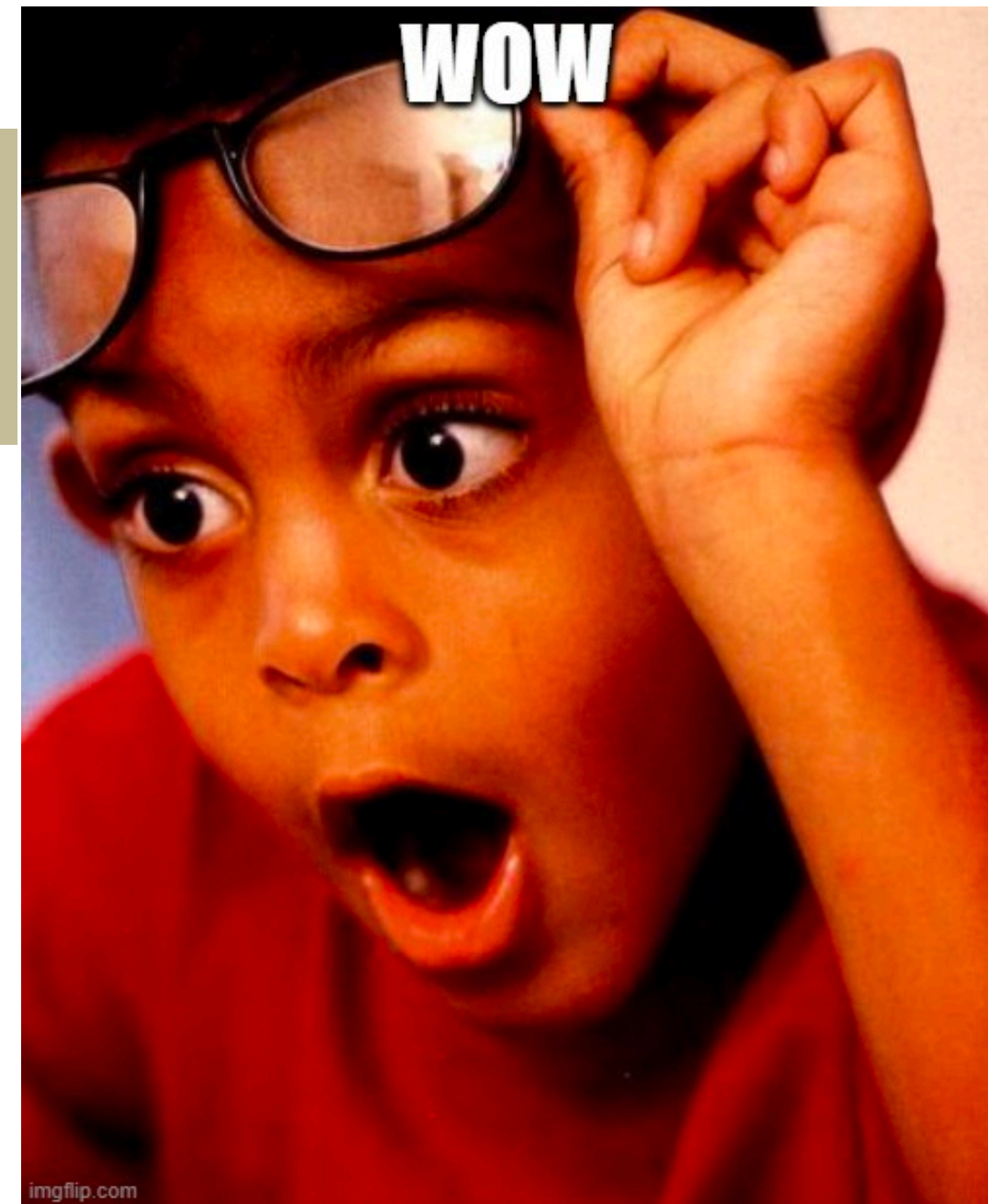
- Flutter Tutorial for Beginners

<https://www.youtube.com/playlist?list=PL4cUxeGkcC9jLYyp2Aoh6hcWuxFDX6PBJ>

Desafio para as próximas 2 semanas

Podes ganhar até 0,5 valores
extra na parte prática apenas
assistindo a vídeos do youtube

Fácil!!



Desafio

1. Assistir a este curso de Flutter - <https://www.youtube.com/playlist?list=PL4cUxeGkcC9jLYyp2Aoh6hcWuxFDX6PBJ>
2. Detetar informação desatualizada (na versão atual de Flutter há coisas que já não funcionam tal e qual é explicado no curso)
3. Reportar aqui <https://forms.gle/i3Av3DMyTzFJzcmWA>
4. Os reports vão sendo publicados aqui: <https://docs.google.com/spreadsheets/d/e/2PACX-1vSEcNmEBf4DT1KhLuNFTUXKvr0jkVjiB0rxhcf-qgGzLsnk5BH6CwOzrhapi2atfTnbvFwwDow45dW/pubhtml>

Regras:

Cada report vale 0,25 extra na nota final da componente prática. Mas:

- Cada aluno pode reportar no máximo 2 vezes
- Cada problema pode ser reportado por um único aluno (o primeiro que reportar).
Ou seja, não deverão haver reports repetidos
- Este desafio termina no dia 25 de Março às 23h59