FLUTTER

FLUTTER #10

CLASSES DE OBJETOS

CLASSES

O conceito de **classe**, que nada mais é do que a representação de **objetos** com características e comportamentos em comum. Simplificando, a classe é um “molde” ou uma “forma” (no sentido de forma de bolo, por exemplo).

```
class Funcionario {
  final String nome;
  final int idade;
  final String funcao;
  final String hobby;
}
```

Esta classe (funcionário) possui quatro atributos, que são nome, idade, cargo e hobby. Agora podemos criar uma instância (um objeto) dessa classe

CLASSES – Construtor Padrão

Para criar uma instância de class é necessário um **construtor.**
Os construtores “constroem” objetos (instâncias) a partir de uma classe.

```dart
class Funcionario {
  String? nome;
  int? idade;
  String? funcao;
  String? hobby;

  Funcionario(String nome, int idade, String funcao, String hobby) {
    this.nome = nome;
    this.idade = idade;
    this.funcao = funcao;
    this.hobby = hobby;
  }
}
```

Caso você não implemente seu construtor em uma classe, o Dart gera um **construtor padrão vazio.**

CLASSES – Construtor Syntax Sugar

Syntax Sugar é definido como uma forma de "adocicar" o código, ou seja, **simplificar e facilitar a escrita e a leitura/compreensão do código**.

```
class Funcionario {
  String nome;
  int idade;
  String funcao;
  String hobby;

  Funcionario(this.nome, this.idade, this.funcao, this.hobby);
}
```

Observe que, nesse caso, não usamos o sinal **?** na declaração dos nossos atributos, pois o objeto Funcionario, ao ser inicializado, terá necessariamente valores não nulos

>

CLASSES – Instância

Criando o objeto funcionário.

```
Funcionario funcionario = Funcionario("Pam", 26, "Recepcionista", "Artes");
```

Parâmetros em construtores

Quando falamos sobre construtores em Dart, é essencial comentar sobre os quatro principais parâmetros que podemos utilizar:

1. **Opcionais posicionais**;
2. **Opcionais nomeados**;
3. **Obrigatórios posicionais**; e
4. **Obrigatórios nomeados**.

DART

Construtor const

Este construtor usa um **modificador de imutabilidade** chamado const, que determina que o objeto não pode inicializar sem um valor e também que, após receber um valor, ele não pode ser alterado.

```dart
class Funcionario {
  final String nome;
  final int idade;
  final String funcao;
  final String hobby;

  const Funcionario(this.nome, this.idade, this.funcao, this.hobby);
}
```

HERANÇA

Copyright © 2024 Accenture. All rights reserved.

DART

Herança

```dart
// Classe mãe
class Funcionario {
  String nome;
  int idade;
  String hobby;

  Funcionario(this.nome, this.idade, this.hobby);
}
```

```dart
// Classe filha
class Gerente extends Funcionario {
  Gerente(super.nome, super.idade, super.hobby);
}
```

```dart
// Classe filha
class Vendedor extends Funcionario {
  Vendedor(super.nome, super.idade, super.hobby);
}
```

DART

Factory

Usando a palavra chave **factory** na frente do construtor nomeado e retornando para cada caso uma instância diferente.

```dart
// Classe mãe
class Funcionario {
  String nome;
  int idade;
  String hobby;

  Funcionario(this.nome, this.idade, this.hobby);

  // Construtor de fábrica
  factory Funcionario.criar(String nome, int idade, String hobby,
      {String funcao = ""}) {
    switch (funcao) {
      case "Gerente":
        return Gerente(nome, idade, hobby);
      case "Vendedor(a)":
        return Vendedor(nome, idade, hobby);
      default:
        return Funcionario(nome, idade, hobby);
    }
  }
}
```

Factory – Construtor nomeado

```
// Instância de Gerente:
Funcionario gerente = Funcionario.criar("Michael Scott", 47, "Improviso", funcao: "Gerente");
 // Instância de Vendedor:
Funcionario vendedor = Funcionario.criar("Jim Halpert", 26, "Pegadinhas", funcao: "Vendedor(a)");
```

CONCEITOS DE O.O.

Conceitos de Orientação à objetos

***Encapsulamento**: Agrupar variáveis (propriedades) e funções (métodos) em unidades chamadas objetos. Isso reduz a complexidade e aumenta a capacidade de reutilização.*

***Abstração**: você não deve ser capaz de modificar diretamente as propriedades ou acessar todos os métodos – em vez disso, pense em escrever uma interface simples para seu objeto. Isso ajuda a isolar o impacto das alterações feitas dentro dos objetos.*

***Herança**: Elimine código redundante herdando coisas de outro objeto ou classe. Isso ajuda você a manter sua base de código menor e mais fácil de manter.*

***Polimorfismo**: por causa da herança, uma coisa pode se comportar de maneira diferente dependendo do tipo do objeto referenciado. Isso ajuda você a refatorar e eliminar ifs feios e instruções switch/case*

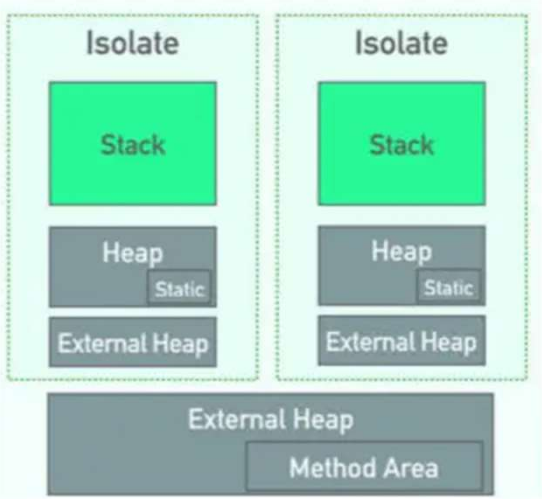
ISOLATES

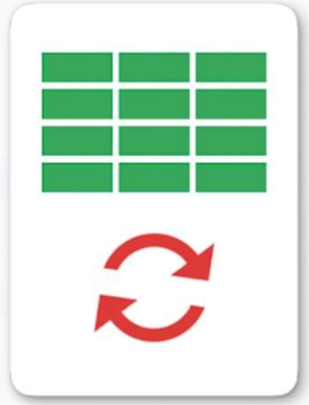
Os códigos Dart são sempre executados dentro de um Isolate, que nada mais é do que uma thread e uma região de memória isolada dedicada a sua execução.

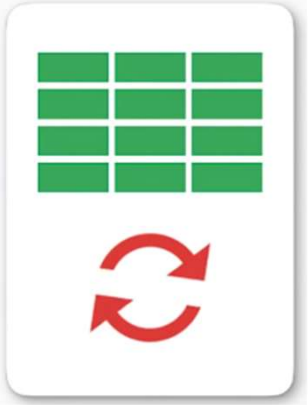
Um Isolate é como se fosse uma fatia da sua máquina dedicada à execução dos códigos Dart.
Um Isolate possui apenas uma thread, uma região dedicada de memória e seu próprio *event-loop*, totalmente independente de outros Isolates. Como há apenas uma thread, não há concorrência no acesso à memória nem problemas de mutabilidade de dados, e os clássicos problemas de bloqueios, *dead-locks* etc, comuns em programação paralela simplesmente não ocorrem aqui. **Por isso *non-bloking***

ISOLATES

Assim como em Java, Dart possui um método *main()*, que possui um Isolate padrão, e todo o código que roda a partir da *main()* roda nessa mesma Isolate, sem concorrência no acesso a memória e com seu *event-loop* gerenciando a execução.







DART oferece duas formas de execução.

Just-In-Time (JIT), com o auxílio da Dart VM, temos um interpretador de nossos códigos Dart, e a tradução para linguagem de máquina (binário) ocorre durante a execução do programa e ele se comporta basicamente como uma linguagem interpretada, funcionando de forma bastante similar ao JavaScript.

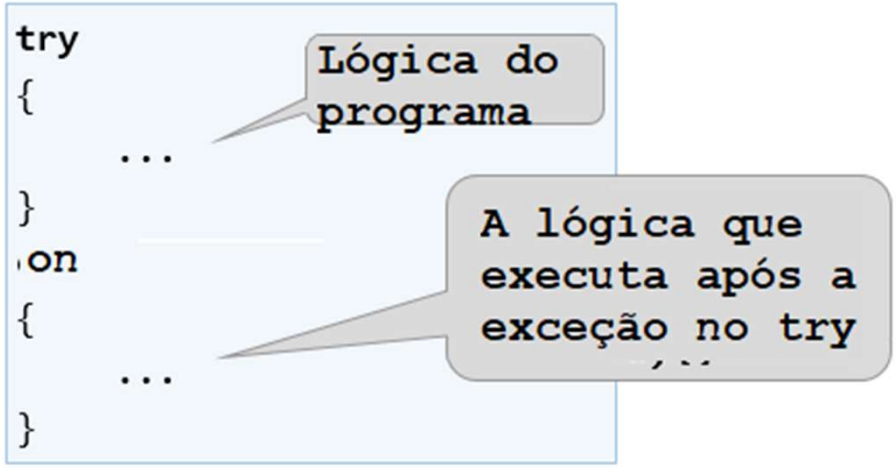
Ahead-of-time (AOT), que nos dá a opção de compilar nossos códigos e gerar nosso binários, com o auxílio da ferramenta dart2native, produz código nativo para as mais diversas plataformas, como Android, IOS, Linux, Windows e macOS

Tratamento de exceções

Copyright © 2023 Accenture. All rights reserved.

Tratamento de exceções

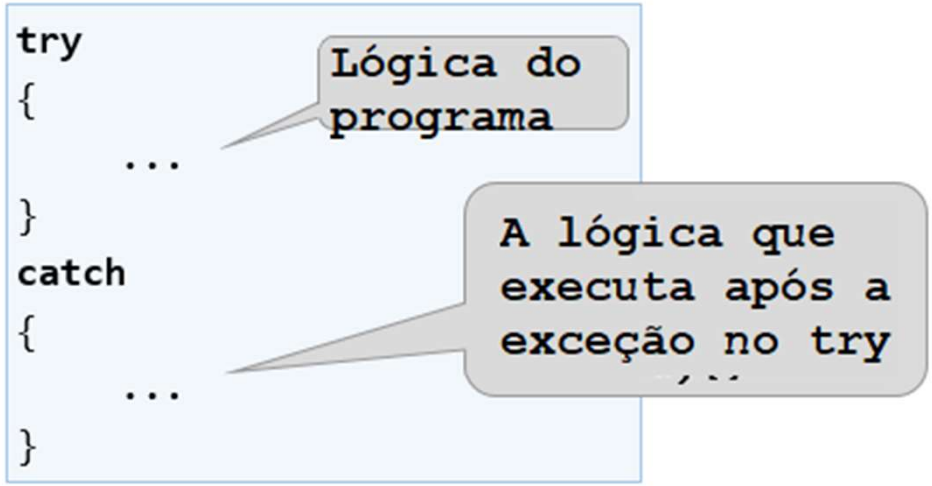
As exceções são uma notificação de um evento que viola a execução normal de um programa.




```dart
try {
  breedMoreLlamas();
} on OutOfLlamasException {
  // A specific exception
  buyMoreLlamas();
} on Exception catch (e) {
  // Anything else that is an exception
  print('Unknown exception: $e');
} catch (e) {
  // No specified type, handles all
  print('Something really unknown: $e');
}
```

Tratamento de exceções

Pega todas as exceções do mundo.




```dart
try {
  breedMoreLlamas();
} catch (e) {
  // ... handle exception ...
} finally {
  // Always clean up, even if an exception is thrown.
  cleanLlamaStalls();
}
```

Coleções

Copyright © 2022 Accenture. All rights reserved.

DART

Lista

No Dart, arrays são objetos List.

List<*tipo_de_dado*> *nome_lista*

Lista de Strings e inteiros

```dart
void main() {
  List<String> nomes = ['Julio', 'Cesar'];
  print(nomes);

  List<int> idades = [10,29,37];
  print (idades);

}
```

Run

Console

```
[Julio, Cesar]
[10, 29, 37]
```

Lista dinâmica que aceita qualquer tipo de dados.

```dart
void main() {
  List nomes = ['Julio', 'Cesar', 10, false, 8.5];
  print(nomes);
}
```

Run

Console

```
[Julio, Cesar, 10, false, 8.5]
```

DART

Lista

Os métodos da lista vem embarcados no DART

Nome_lista.métodos

```
5   List<int> idades = [10,29,37];
6   print (idades);
7
8   idades.
9 }
10
```

```
length
reversed
hashCode
runtimeType
first
firstOrNull
indexed
isEmpty
isNotEmpty
iterator
last
lastOrNull
nonNulls
single
singleOrNull
add(int value) → void
addAll(Iterable<int> iterable) → void
```

```
1 void main() {
2   List<int> idades = [10,29,37];
3   print (idades);
4
5   int tamanho = idades.length;
6   print(tamanho);
7 }
8
```

Run

Console

```
[10, 29, 37]
3
```

DART

Lista

Manipulando a lista

assert()

```dart
void main() {
  List<int> lista = [1, 2, 3];
  assert(lista.length == 4);
  assert(lista[1] == 2);

  lista[1] = 1;
  assert(lista[1] == 1);

}
```

Run

Console

```
Uncaught Error: Assertion failed
```

add() e remove()

```dart
void main() {
  List<int> lista = [1, 2, 3];

  lista.add(6);
  lista.remove(1);

  print(lista);
}
```

Run

Console

```
[2, 3, 6]
```

add() e remove()

```dart
void main() {
  List<int> lista = [1, 2, 3];

  lista.add(6);
  lista.remove(1);

  print(lista);
}
```

Run

Console

```
[2, 3, 6]
```

contains()

```dart
void main() {
  List<int> lista = [1, 2, 3];

  lista.add(6);
  lista.remove(1);

  print(lista.contains(10));
}
```

Run

Console

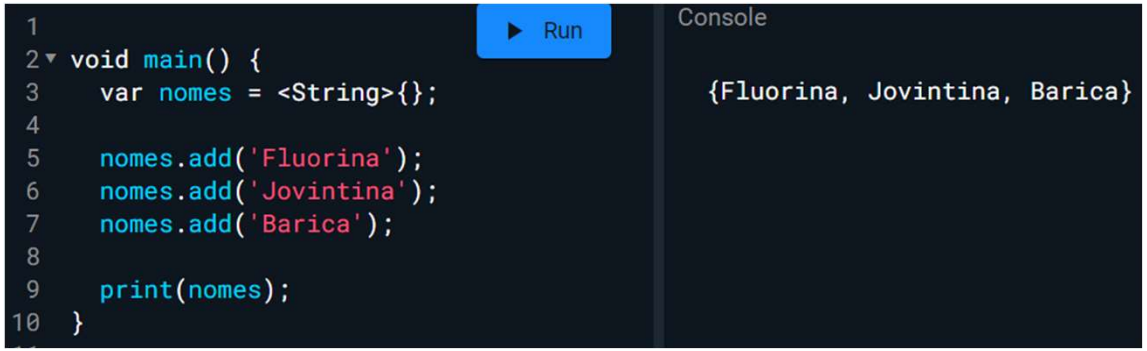
```
false
```

DART

Set

Um set no Dart é uma coleção não ordenada de itens exclusivos.

var nome_do-conjuto = <*tipo_de_dados*>{*valores*}



Copyright © 2023 Accenture. All rights reserved.

DART

Map

Um mapa é um objeto que associa chaves e valores.

var nome_do_mapa = **Map**<*chave, valor*>()