Introdução ao Dart

Usabilidade, Desenvolvimento Web, Mobile e Jogos

Prof. Gustavo Custodio
gustavo.custodio@anhembi.br

Introdução à Linguagem Dart

Introdução ao Dart

Introducao ao Dart

Coleções

Coleções

- Coleções são ferramentas para agrupar elementos do mesmo tipo em um conjunto.

Coleções

- Coleções são ferramentas para agrupar elementos do mesmo tipo em um conjunto.
- Incluem:
  - Listas;
  - Maps;
  - Conjuntos.

Listas e Conjuntos

- As listas em Dart são declaradas usando a palavra reservada `List`.

Listas e Conjuntos

- As listas em Dart são declaradas usando a palavra reservada `List`.

```dart
void brincandoComListas() {
    // Criando uma lista de números
    List<int> numeros = [1, 2, 3, 5, 7];
    numeros.add(10); // Adicionando um número na lista
    numeros.addAll([4, 1, 35]); // Adiciona vários números
    print(numeros);
}
```

Listas e Conjuntos

- No exemplo anterior, é mostrada uma lista contendo apenas elementos do tipo inteiro.

Listas e Conjuntos

- No exemplo anterior, é mostrada uma lista contendo apenas elementos do tipo inteiro.
  - No entanto, é possível criar uma lista sem explicitamente definir um tipo.
    - `List idades = ['quarenta', 35, 25]`.

Listas e Conjuntos

- No exemplo anterior, é mostrada uma lista contendo apenas elementos do tipo inteiro.
  - No entanto, é possível criar uma lista sem explicitamente definir um tipo.
    - `List idades = ['quarenta', 35, 25]`.
  - Neste caso a lista será do tipo `dynamic`.

Listas e Conjuntos

- `dynamic` é um tipo que aceita qualquer tipo de valor em Dart.

Listas e Conjuntos

- `dynamic` é um tipo que aceita qualquer tipo de valor em Dart.
  - Pode ter seu valor e tipo alterados.
  - Similar ao `Object` do Java.
  - Podemos declarar variáveis `dynamic`:

Listas e Conjuntos

- `dynamic` é um tipo que aceita qualquer tipo de valor em Dart.
  - Pode ter seu valor e tipo alterados.
  - Similar ao `Object` do Java.
  - Podemos declarar variáveis `dynamic`:

```
dynamic varDinamica = 10;
varDinamica = 'dez';
```

Listas e Conjuntos

- Conjunto é o tipo menos frequente de coleção utilizada.

Listas e Conjuntos

- Conjunto é o tipo menos frequente de coleção utilizada.
  - Funciona de maneira similar a uma lista, mas não permite elementos duplicados.
  - Delimitado por chaves { } ao invés de colchetes [ ].

Listas e Conjuntos

- Conjunto é o tipo menos frequente de coleção utilizada.
  - Funciona de maneira similar a uma lista, mas não permite elementos duplicados.
  - Delimitado por chaves { } ao invés de colchetes [ ].

```
void brincandoComConjuntos() {
    Set<int> numeros = {1, 2, 3, 5, 7};
    numeros.add(3); // Adicionando um elemento novo
    numeros.addAll({4, 6, 7}); // Adicionando vários
    print(numeros);
}
```

Maps

- Mapas (`Map`) armazenam dois elementos para cada item: **uma chave e um valor**.

Maps

- Mapas (`Map`) armazenam dois elementos para cada item: **uma chave e um valor**.
  - A chave serve para “consultar” o valor dentro do mapa.
    - Funciona de maneira similar a um índice numérico em um vetor.

Maps

```
// Criando um mapa com string como chave e inteiro como valor
Map<String, int> pontuacao = {
    'AAA': 10000,
    'Gabriel': 5000,
    'Pedro': 2000,
};

// Alterando a pontuação do AAA
pontuacao['AAA'] = 30000;
print(pontuacao);
```

Maps

- Elementos de coleções podem ser percorridos utilizando o `for`.

Maps

- Elementos de coleções podem ser percorridos utilizando o `for`.

```
// Percorrendo lista
for(int numero in numeros) {
    print(numero);
}
// Percorrendo Map
for (MapEntry e in pontuacao.entries) {
    print("Chave ${e.key}, Valor ${e.value}");
}
```

Exercício 1

- Crie um `Map` que associa países com suas respectivas populações.
- Adicione as populações de 7 países diferentes.
- No final, percorra o `Map` e mostre os países ao lado de suas respectivas populações.

Introducao ao Dart

Funções

Funções

- Funções são o bloco fundamental de linguagens de programação, com o Dart não sendo exceção.

Funções

- Funções são o bloco fundamental de linguagens de programação, com o Dart não sendo exceção.
- Sintaxe:

```
tipoRetornoOpcional nomeFuncao(tipoOpcional parametro1, ...) {
    // codigo aqui
}
```

Funções

```
void funcaoClassica(String parametro) {
    print('${parametro}');
}
```

Funções

```
void funcaoClassica(String parametro) {
    print('${parametro}');
}
```

- O exemplo acima é o exemplo mais comum de uma função, que possui um parâmetro obrigatório.

Funções com Parâmetros Opcionais

- O Dart possui a capacidade de incluir **parâmetros opcionais**.

Funções com Parâmetros Opcionais

- O Dart possui a capacidade de incluir **parâmetros opcionais**.
  - Eles podem ser omitidos sem causar erros de execução.
  - São delimitados por colchetes.

Funções com Parâmetros Opcionais

- O Dart possui a capacidade de incluir **parâmetros opcionais**.
  - Eles podem ser omitidos sem causar erros de execução.
  - São delimitados por colchetes.

```dart
void funcaoComOpcionais([String? nome, int? idade]) {
  final nomeVerdadeiro = nome ?? 'Desconhecido';
  final idadeVerdadeira = idade ?? 0;
  print('${nomeVerdadeiro} tem ${idadeVerdadeira} anos.');
}
```

Funções com Parâmetros Opcionais

- O Dart possui a capacidade de incluir **parâmetros opcionais**.
  - Eles podem ser omitidos sem causar erros de execução.
  - São delimitados por colchetes.

```dart
void funcaoComOpcionais([String? nome, int? idade]) {
  final nomeVerdadeiro = nome ?? 'Desconhecido';
  final idadeVerdadeira = idade ?? 0;
  print('${nomeVerdadeiro} tem ${idadeVerdadeira} anos.');
}
```

  - O símbolo ?? é utilizado para verificar se um valor é nulo.
    - Caso seja, a variável recebe o que está após esse símbolo (Desconhecido e 0).

Funções com Parâmetros Opcionais

- O exemplo anterior inclui parâmetros opcionais **sem nome**.

Funções com Parâmetros Opcionais

- O exemplo anterior inclui parâmetros opcionais **sem nome**.
  - Portanto, se quisermos invocar a função com parâmetros opcionais:

Funções com Parâmetros Opcionais

- O exemplo anterior inclui parâmetros opcionais **sem nome**.
  - Portanto, se quisermos invocar a função com parâmetros opcionais:

```
funcaoComOpcionais('Frederico', 8);
```

Funções com Parâmetros Opcionais

- O exemplo anterior inclui parâmetros opcionais **sem nome**.
  - Portanto, se quisermos invocar a função com parâmetros opcionais:

```
funcaoComOpcionais('Frederico', 8);
```

  - No entanto, essa não é a forma mais comum de utilizar parâmetros opcionais em Dart.

Funções com Parâmetros Opcionais

- Em Dart, é muito comum instanciar widgets que possuem muitos parâmetros opcionais.

Funções com Parâmetros Opcionais

- Em Dart, é muito comum instanciar widgets que possuem muitos parâmetros opcionais.

```
Container(
    margin: const EdgeInsets.all(10.0),
    color: Colors.red,
    height: 48.0,
    child: Text('Parâmetros nomeados!'),
);
```

Funções com Parâmetros Opcionais

- Em Dart, é muito comum instanciar widgets que possuem muitos parâmetros opcionais.

```
Container(
    margin: const EdgeInsets.all(10.0),
    color: Colors.red,
    height: 48.0,
    child: Text('Parâmetros nomeados!'),
);
```

  - Há mais parâmetros opcionais do que esses, só o `Container` possui mais de 10.
  - Agora imagine saber ordem correta dos parâmetros sem os nomes à esquerda.

Funções com Parâmetros Opcionais

- Neste contexto, temos os parâmetros **nomeados opcionais**.

Funções com Parâmetros Opcionais

- Neste contexto, temos os parâmetros **nomeados opcionais**.
- Eles são delimitados por chaves ({ }) na assinatura da função.

Funções com Parâmetros Opcionais

- Neste contexto, temos os parâmetros **nomeados opcionais**.
- Eles são delimitados por chaves ({ }) na assinatura da função.

```
void parametrosComNome(String mensagem,
                       {int vezesMostrar = 1, String? despedida})
{
    for (int i = 0; i < vezesMostrar; i++) {
        print(mensagem);
    }

    String mensagemDespedida = despedida ?? 'Tchau';
    print(mensagemDespedida);
}
```

Funções com Parâmetros Opcionais

- Nessa função temos um parâmetro obrigatório (mensagem) e outros dois parâmetros nomeados opcionais.

Funções com Parâmetros Opcionais

- Nessa função temos um parâmetro obrigatório (mensagem) e outros dois parâmetros nomeados opcionais.
- Vamos testar a seguinte chamada para a função:

Funções com Parâmetros Opcionais

- Nessa função temos um parâmetro obrigatório (mensagem) e outros dois parâmetros nomeados opcionais.
- Vamos testar a seguinte chamada para a função:

```
parametrosComNome('Temos muitos parâmetros.', vezesMostrar: 5);
```

- Será mostrada 5 vezes a mensagem “Temos muitos parâmetros”, seguida da mensagem “Tchau”.

Closures

- Dart permite o uso de *closures* (também chamado finals de funções de primeira classe).
  - Funções que são tratadas como objetos e podem ser passadas como parâmetros de funções.

```
final responder = (String nome) => print('Olá, $nome!');
```

  - Neste caso, criamos uma função que recebe um parâmetro do tipo String e não possui retorno.

Closures

```
void conversar(String nome, Function(String) responder) {
    print('Muito prazer, meu nome é$nome');
    responder(nome);
}

void main() {
    final responder = (String nome) => print(
        'Olá, $nome! Prazer em te conhecer.'
    );
    conversar('Alexandre', responder);
}
```

Closures

```
void conversar(String nome, Function(String) responder) {
    print('Muito prazer, meu nome é$nome');
    responder(nome);
}

void main() {
    final responder = (String nome) => print(
        'Olá, $nome! Prazer em te conhecer.'
    );
    conversar('Alexandre', responder);
}
```

- Passamos a função `responder` como uma parâmetro para a função `conversar`.

Exercício 2

- Crie duas funções em Dart:
  - `converterCelsiusParaFahrenheit(double temp)`
  - `converteCelsiusParaKelvin(double temp)`
- Crie uma terceira função `converterTemperatura`, que recebe como parâmetro:
  - A temperatura em Celsius;
  - Uma das duas funções criadas anteriormente, dependendo do tipo de conversão

Introducao ao Dart

Classes e Construtores

Classes

- O Dart permite POO (programação orientada a objetos), ou seja, é possível criar classes e objetos.

Classes

- O Dart permite POO (programação orientada a objetos), ou seja, é possível criar classes e objetos.
- Classes em Dart são declaradas com um padrão muito similar ao Java.

Classes

- O Dart permite POO (programação orientada a objetos), ou seja, é possível criar classes e objetos.
- Classes em Dart são declaradas com um padrão muito similar ao Java.
  - Permite herança;
  - Permite implementação de interfaces.

Classes

- O Dart permite POO (programação orientada a objetos), ou seja, é possível criar classes e objetos.
- Classes em Dart são declaradas com um padrão muito similar ao Java.
  - Permite herança;
  - Permite implementação de interfaces.
- Assim como o Java, a palavra chave **this** é utilizada no construtor para se referir a atributos de uma classe.

Classes

```
class Nome {
    // O _ indica um atributo privado
    final String _primeiro;
    final String _sobrenome;

    const Nome(this._primeiro, this._sobrenome);

    @override
    String toString() {
        return "$_primeiro $_sobrenome";
    }
}
```

Classes

- O construtor mostrado recebe dois parâmetros: nome e sobrenome.
  - Que são passados para os **atributos** `_primeiro` e `_sobrenome`.

Classes

- O construtor mostrado recebe dois parâmetros: nome e sobrenome.
  - Que são passados para os **atributos** `_primeiro` e `_sobrenome`.
- Ao invés das palavras chave `public` ou `private`, definimos um atributo privado com um *underscore* (_) no início.

Classes

- O construtor mostrado recebe dois parâmetros: nome e sobrenome.
  - Que são passados para os **atributos** `_primeiro` e `_sobrenome`.
- Ao invés das palavras chave `public` ou `private`, definimos um atributo privado com um *underscore* (_) no início.
- Vemos na classe também um método `toString()`, utilizado para definir o que será mostrado quando mandarmos printar um objeto.

Classes

```
void main() {
    Nome nomeCompleto = Nome('Marcelo', 'da Silva');
    print(nomeCompleto);
}
```

Classes

```
void main() {
    Nome nomeCompleto = Nome('Marcelo', 'da Silva');
    print(nomeCompleto);
}
```

- Neste trecho de código é impresso na tela o que o `toString()` retorna.
- No Dart não é necessário utilizar a palavra chave new para instanciar uma classe.

Classes

- Vamos voltar ao exemplo do *Hello World* do Flutter:

Classes

- Vamos voltar ao exemplo do *Hello World* do Flutter:

```
class HelloWorld extends StatelessWidget {
    const HelloWorld({super.key});
    ...
}
```

Classes

- Vamos voltar ao exemplo do *Hello World* do Flutter:

```
class HelloWorld extends StatelessWidget {
    const HelloWorld({super.key});
    ...
}
```

- Assim como em Java, a palavra chave `extends` é utilizada para informar que a classe *Hello World* **herda de** *StatelessWidget*.
- O construtor recebe um parâmetro opcional `key` vindo da superclasse.

Exercício 3

- Crie uma classe smartphone.
- Pense nos atributos e comportamentos dos objetos dessa classe.
- Crie uma outra classe que tem um método main.

Exercício 4

- Crie uma lista de tarefas;
  - crie um método main.
  - crie uma lista que armazene objetos do tipo String;
  - cada tarefa será representada por uma string.
  - adicione várias tarefas a sua lista.
  - imprima o conteúdo de todas as mensagens.

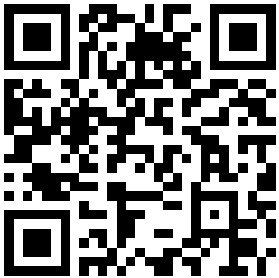
Exercício 5

- Crie uma classe chamada Biblioteca e outra classe chamada Livro.
- Livro:
  - Atributos:
    - nome;
    - ISBN.
  - Métodos:
    - toString();
- Biblioteca:
  - Atributos:
    - List<Livro> livros.
  - Métodos:
    - adicionarLivro(Livro livro);
- Imprima todo o conteúdo dessa biblioteca.

Referências

Simone Alessandria and Brian Kayfitz.
Flutter Cookbook: Over 100 proven techniques and solutions for app development with Flutter 2.2 and Dart.
Packt Publishing Ltd, 2021.

Conteúdo



https://gustavotcustodio.github.io/usabilidade.html

Obrigado

gustavo.custodio@anhembi.br