Programação orientada a objeto
Aula 1

Fernando Duwe

v 1.0

Monólito significa “obra construída em uma só pedra” por isso é utilizado para definir a arquitetura de alguns sistemas, refere-se a forma de desenvolver um sistema, programa ou aplicação onde todas as funcionalidades e códigos estejam em um único processo. Essas diversas funcionalidades estão em um mesmo código fonte e em sua execução compartilham recursos da mesma máquina, seja processamento, memória, bancos de dados e arquivos.

Como o sistema está inteiro em um único bloco, seu desenvolvimento é mais ágil, se comparado com outras arquiteturas, sendo possível desenvolver uma aplicação em menos tempo e com menor complexidade inicial, reparem na palavra inicial.

A programação orientada a objetos é um modelo de programação onde diversas classes possuem características que definem um objeto na vida real. Cada classe determina o comportamento do objeto definido por métodos e seus estados possíveis definidos por atributos. São exemplos de linguagens de programação orientadas a objetos: C++, Java, C#, Object Pascal, entre outras.

Este modelo foi criado com o intuito de aproximar o mundo real do mundo virtual. Para dar suporte à definição de Objeto, foi criada uma estrutura chamada Classe, que reúne objetos com características em comum, descreve todos os serviços disponíveis por seus objetos e quais informações podem ser armazenadas.

Classes

Uma classe é uma forma de definir um tipo de dado em uma linguagem orientada a objeto. Ela é formada por dados e comportamentos.

Para definir os dados são utilizados os atributos, e para definir o comportamento são utilizados métodos. Depois que uma classe é definida podem ser criados diferentes objetos que utilizam a classe.

Exemplo de uma classe em Dart

```
// Declarando a classe Pessoa
class Pessoa {
  // Atríbutos
  String nome;
  int idade;

  // Método Constutor
  Pessoa({required this.nome, required this.idade});

  // métodos
  String meApresentar() {
    return "Olá, meu nome é ${this.nome}, e tenho ${this.idade} anos";
  }
}
```

Abstração

A abstração consiste em um dos pontos mais importantes dentro de qualquer linguagem Orientada a Objetos. Como estamos lidando com uma representação de um objeto real (o que dá nome ao paradigma), temos que imaginar o que esse objeto irá realizar dentro de nosso sistema.

São três pontos que devem ser levados em consideração nessa abstração:

Abstração

O primeiro ponto é darmos uma identidade ao objeto que iremos criar. Essa identidade deve ser única dentro do sistema para que não haja conflito.

```
class Pessoa {
}
```

Abstração

A segunda parte diz respeito a características do objeto. Como sabemos, no mundo real qualquer objeto possui elementos que o definem. Dentro da programação orientada a objetos, essas características são nomeadas propriedades. Por exemplo, as propriedades de um objeto “Cachorro” poderiam ser “Tamanho”, “Raça” e “Idade”.

```
class Pessoa {

  String nome;
  int idade;
```

Abstração

Por fim, a terceira parte é definirmos as ações que o objeto irá executar. Essas ações, ou eventos, são chamados métodos. Esses métodos podem ser extremamente variáveis, desde “Acender()” em um objeto lâmpada até “Latir()” em um objeto cachorro.

```
  String meApresentar() {
    return "Olá, meu nome é ${this.nome}, e tenho
${this.idade} anos";
  }

  void fazerAniversario() {
    this.idade++;
  }
```

Encapsulamento

O conceito do encapsulamento consiste em “esconder” os atributos da classe de quem for utilizá-la. Isso se deve por dois motivos principais:

Um é para que alguém que for usar a classe não a use de forma errada como, por exemplo, em uma classe que tem um método de divisão entre dois atributos da classe.

O outro motivo é de manter todo o código de uma determinada classe encapsulada dentro dela mesmo como, por exemplo, se existe uma classe Conta, talvez seja melhor não permitir que um programador acesse o atributo saldo diretamente, nem mesmo com os métodos get e set, mas somente por operações, como saque, depósito e saldo.

Exemplo de encapsulamento

```
class Calculadora {
  double valor1 = 0; // atributo público
  double _valor2 = 0; // atributo privado
  CalculadoraOperacao operacao = CalculadoraOperacao.soma;

  double calcular() { // método público
    return valor1 + _valor2;
  }

  double _calcularOperacao() { // método privado
    return 0;
  }
}
```

Exercício 1

Crie uma classe de ContaCorrente, a classe deve implementar as seguintes funcionalidades:

- Armazenar o saldo atual, que deve ser privado;
- Possuir os métodos de depósito e saque;
- Os métodos de saque e depósito devem receber o valor a ser sacado ou a ser depositado;
- Criar um método também para exibir o saldo atual;

Exercício 2

Crie uma classe para representar uma nota específica, guardando os seguintes atributos:

- Descrição da prova;
- Nota;

Crie uma lista de Notas e calcule sua média geral.

Getters e Setters

Getters e setters são usados para proteger seus dados, especialmente na criação de classes.

Para cada instância de variável, um método getter retorna seu valor, enquanto um método setter o define ou atualiza. Com isso em mente, getters e setters também são conhecidos como métodos de acesso e de modificação, respectivamente.

Getters e Setters

Exemplo de utilização de getters e setters

```
double get right => left + width;

set right(double value) => left = value - width;

double get bottom => top + height;

set bottom(double value) => top = value - height;
```

Herança

A herança é um tipo de relacionamento que define que uma classe "é um" de outra classe como, por exemplo, a classe Funcionario que é uma Pessoa, assim um Funcionário tem um relacionamento de herança com a classe Pessoa.

Ver mais

```
class Television {
  void turnOn() {
    _illuminateDisplay();
    _activateIrSensor();
  }
  // ...
}

class SmartTelevision extends Television {
  void turnOn() {
    super.turnOn();
    _bootNetworkInterface();
    _initializeMemory();
    _upgradeApps();
  }
  // ...
}
```

Exercício 3

Vamos criar uma classe chamada Carro. Essa classe contará com as seguintes informações:

- Quilometragem atual;
- Litros de gasolina no tanque;
- Número da placa;
- Deve possuir também os seguintes métodos: ligar, desligar acelerar e frear;

Após criar essa classe, crie uma nova classe chamada Jipe, que herda as funções de carro, mas possui dois novos métodos: ativar4x4 e desativar4x4

Polimorfismo

O Polimorfismo é a possibilidade de em uma hierarquia de classes implementar métodos com a mesma assinatura e, assim, implementar um mesmo código que funcione para qualquer classe dessa hierarquia sem a necessidade de implementações específicas para cada classe. O principal objetivo do polimorfismo é diminuir a quantidade de código escrito, aumentando a clareza e a facilidade de manutenção.

Um exemplo de polimorfismo

```
class Television {
  // ...
  set contrast(int value) {...}
}

class SmartTelevision extends Television {
  @override
  set contrast(num value) {...}
  // ...
}
```

Construtores e destrutores

Construtores são basicamente funções de inicialização de uma classe, as quais são invocadas no momento em que objetos desta classe são criadas. Eles permitem inicializar campos internos da classe e alocar recursos que um objeto da classe possa demandar, tais como memória, arquivos, semáforos, soquetes, etc.

Destrutores realizam a função inversa: são funções invocadas quando um objeto está para ``morrer''. Caso um objeto tenha recursos alocados, destrutores devem liberar tais recursos. Por exemplo, se o construtor de uma classe alocou uma variável dinamicamente com new, o destrutor correspondente deve liberar o espaço ocupado por esta variável com o operador delete.

Ver mais

```
class Point {
  double x = 0;
  double y = 0;

  Point(double x, double y) {
    // See initializing formal parameters for a better way
    // to initialize instance variables.
    this.x = x;
    this.y = y;
  }

  Point.origin()
      : x = xOrigin,
        y = yOrigin;
}
```

Exercício 4

Crie uma nova classe chamada FolhaSalarial. Essa classe deve possuir as seguintes características:

- Guardar o valor recebido por hora;
- Guardar a quantidade de horas trabalhadas durante o mês;
- Possuir um método calcular, que calculará o valor total a receber;

Crie uma classe chamada FolhaSalarialVendedor, que além de calcular o valor total a receber por mês, guardará também a quantidade total de vendas e um valor de bonificação por cada venda.

Ao calcular a folha, levar esses valores em consideração.

Exercício 5

Crie uma classe chamada País, devendo possuir as seguintes propriedades:

- Nome do país;
- População;
- Área;

Crie uma classe chamada Continente. A classe de continente deve possuir as seguintes funcionalidades:

- Guardar o nome do continente;
- Uma lista dos países, inicializada no método construtor. Essa lista deve ser privada;
- Um método para adicionar um novo país;
- Um Getter para saber a população total do continente;
- Um Getter para saber a área total do continente;