



Características das Linguagens de Programação I

Conteúdo: Construtores e encapsulamento

Prof. Dsc. Giomar Sequeiros
giomar@eng.uerj.br

Construtores e encapsulamento

Construtores

- Sempre que queremos **criar um novo objeto** de uma determinada classe utilizamos a palavra **new** acompanhada por um construtor.
- O construtor de uma classe tem, por definição, o **mesmo nome que a classe**.

```
public class Conta {  
  
    int numero;  
    Cliente titular;  
    double saldo;  
    double limite;  
  
    // construtor  
    public Conta() {  
        System.out.println("Construindo uma conta.");  
    }  
  
    // ..  
}
```

Quando fizermos: `Conta c = new Conta();`

A mensagem "construindo uma conta" aparecerá. É como uma rotina de inicialização que é chamada sempre que um novo objeto é criado. Um **construtor** pode parecer, mas **não** é um **método**.

Construtores

- Quando você não declara nenhum construtor na sua classe, o Java cria um para você. Esse construtor é o **construtor default**, ele não recebe nenhum argumento e o **corpo** dele é **vazio**.
- A partir do momento que você declara um construtor, o construtor default não é mais fornecido.
- Um construtor **não** é um **método**.

Construtores: Exemplo

Um construtor só pode rodar durante a **construção do objeto**, isto é, você nunca conseguirá chamar o construtor em um objeto já construído. Porém, durante a construção de um objeto, você pode fazer com que um **construtor chame outro**.

```
public class Conta {  
    int numero;  
    Cliente titular;  
    double saldo;  
    double limite;  
  
    // construtor  
    public Conta (Cliente titular) {  
        // faz mais uma série de inicializações e configurações  
        this.titular = titular;  
    }  
  
    public Conta (int numero, Cliente titular) {  
        this(titular); // chama o construtor que foi declarado acima  
        this.numero = numero;  
    }  
  
    //..  
}
```


Encapsulamento

- Encapsulamento é uma técnica utilizada para **restringir o acesso** a variáveis (**atributos**), **métodos** ou até à própria **classe**.
- A ideia do encapsulamento na programação orientada a objetos é que **não seja permitido acessarmos diretamente** as propriedades de um objeto. Nesse caso, precisamos **operar sempre por meio dos métodos** pertencentes a ele. A complexidade de um objeto é escondida, portanto, pela abstração de dados que estão “por trás” de suas operações.

Encapsulamento

- Qualificadores de acesso:

public (público): indica que o método ou o atributo são acessíveis por **qualquer classe**, ou seja, que podem ser usados por qualquer classe, independentemente de estarem no mesmo pacote ou estarem na mesma hierarquia;

private (privado): indica que o método ou o atributo são acessíveis apenas pela **própria classe**, ou seja, só podem ser utilizados por métodos da própria classe;

protected (protegido): indica que o atributo ou o método são acessíveis pela **própria classe**, por **classes do mesmo pacote ou classes da mesma hierarquia** (estudaremos hierarquia de classes quando tratarmos de herança).

Quando omitimos o qualificador de acesso, o atributo ou método são considerados **protected por padrão**.

Encapsulamento: Getters e Setters

- Para **permitir** o **acesso** aos **atributos privados** de uma maneira controlada, a prática mais comum é criar dois métodos, um que retorna o valor e outro que muda o valor.
- A **convenção** para esses métodos é de colocar a palavra **get** ou **set** antes do nome do atributo.

Encapsulamento: Exemplo

- Criar um novo projeto, criar a **Circulo** conforme a figura abaixo

```
1 public class Circulo {
2     // Atributos
3     private double radio;
4     private String cor;
5
6     // Construtores
7     public Circulo() { // 1ro construtor
8         this.radio = 1.0;
9         this.cor = "vermelho";
10    }
11    public Circulo(double radio) { // 2do construtor
12        this.radio = radio;
13        this.cor = "vermelho";
14    }
15    public Circulo(double radio, String cor) { // 3ro construtor
16        this.radio = radio;
17        this.cor = cor;
18    }
19 }
```

Encapsulamento: Exemplo

- Continua a classe **Circulo**

```
20 // Getters e setters
21 public double getRadio() {
22     return this.radio;
23 }
24 public void setRadio(double radio) {
25     this.radio = radio;
26 }
27 public String getCor() {
28     return this.cor;
29 }
30 public void setCor(String cor) {
31     this.cor = cor;
32 }
33
34 // Métodos públicos
35 public double getArea() {
36     return this.radio*this.radio*Math.PI;
37 }
38 public String toString() {
39     return "Circulo com radio = " + this.radio + " e cor de " + cor;
40 }
41 }
```

Encapsulamento: Exemplo

- Crie a classe **TesteCirculo**

```
1 public class TesteCirculo {  
2     public static void main(String[] args) {  
3         Circulo c1 = new Circulo(3.0, "azul");  
4         System.out.println(c1.toString());  
5  
6         Circulo c2 = new Circulo(2.0);  
7         System.out.println(c2.toString());  
8  
9         Circulo c3 = new Circulo();  
10        System.out.println(c3.toString());  
11    }  
12 }
```

Referências

BIBLIOGRAFIA BÁSICA:

- DEITEL, Harvery M.. Java : como programar. 10ª ed. São Paulo: Pearson - Prentice Hall, 2017.
- BORATTI, Isaías Camilo. Programação Orientada a Objetos em Java : Conceitos Fundamentais de Programação Orientada a Objetos. 1ª ed. Florianópolis: VisualBooks, 2007.
- SIERRA, Kathy; BATES, Bert. Use a Cabeça! Java. 2ª ed. Rio de Janeiro: Alta Books, 2007.

