

Construtores de Classes

Orientação a Objetos - DCC025

Prof. Edmar Welington Oliveira
edmar.oliveira@ufjf.edu.br

Universidade Federal de Juiz de Fora - UFJF
Departamento de Ciência da Computação - DCC

Exercícios - Revisando

Marque a alternativa CORRETA em relação às visibilidades JAVA:

- a. Visibilidades somente podem ser aplicadas a atributos.
- b. Visibilidades somente podem ser aplicadas a operações/métodos.
- c. A visibilidade pública em atributos deve ser evitada para evitar alterações indevidas nos mesmos.
- d. A visibilidade privada impede que uma classe seja aberta no editor.
- e. A visibilidade *default* em Java é a privada.

Exercícios - Revisando

Um aluno chamado José Carlos do curso de Informática efetua matrícula nas disciplinas de Matemática, Física e Programação. *Quais são os objetos da sentença?*

- a. Aluno, curso, matrícula e disciplina
- b. José Carlos, Informática e disciplinas
- c. José Carlos, Informática, Matemática, Física e Programação
- d. Computador, José Carlos, Informática, Matemática, Física e Programação
- e. Não há objetos, somente classes.

Em relação aos atributos, é correto dizer que:

- a. Uma classe deve ter ao menos um atributo
- b. Atributos e operações são sinônimos
- c. São propriedades de uma classe que definem os valores possíveis para as instâncias das propriedades
- d. Atributos somente são utilizados em tabelas de banco de dados.

Construtores

```
public class Pessoa {  
  
    private String nome;  
  
    public Pessoa(String nomeTeste){  
        nome = nomeTeste;  
    }  
  
    public String getNome(){  
        return nome;  
    }  
  
    public void setNome(String nome){  
        this.nome = nome;  
    }  
}
```

Construtores

```
public class Pessoa {  
  
    private String nome;  
  
    public Pessoa(String nomeTeste) {  
        nome = nomeTeste;  
    }  
  
    public String getNome() {  
        return nome;  
    }  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
}
```

Construtor da classe: permite inicializar os campos declarados

Construtores

Classe Pessoa

```
public class Pessoa {  
  
    private String nome;  
  
    public Pessoa(String nomeTeste){  
        nome = nomeTeste;  
    }  
  
    public String getNome(){  
        return nome;  
    }  
  
    public void setNome(String nome){  
        this.nome = nome;  
    }  
}
```

```
public static void main(String args[]){  
  
    Pessoa p1 = new Pessoa("Edmar Oliveira");  
    System.out.println(p1.getNome());  
}  
}
```

Método main em outra Classe

Construtores

- Construtores de classes
 - Toda classe Java possui um **construtor padrão** ou um **construtor especificado pelo programador**. Todo campo em Java possui um valor inicial padrão (um valor fornecido pelo Java quando o programador não especificar o valor inicial do campo). Ex:
 - Campos do tipo String: possui NULL como valor inicial padrão
 - Tipos inteiros possui "0" como valor padrão.

Construtores

(a)

```
public class Pessoa {  
  
    private String nome;  
  
    public String getNome() {  
        return nome;  
    }  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
}
```

(b)

```
public class Pessoa {  
  
    private String nome;  
  
    public Pessoa(String nomeTeste) {  
        nome = nomeTeste;  
    }  
  
    public String getNome() {  
        return nome;  
    }  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
}
```


Construtores

- Continuação

- **Não se exige** que campos sejam explicitamente inicializados antes de serem utilizados em um programa. Isso quer dizer que classes podem ser instanciadas sem se especificar os valores iniciais dos objetos criados.
 - A classe Pessoa (mostrada anteriormente) pode ser instanciada sem definir valores para os campos Nome, Telefone e Endereço.

Construtores

- Construtores padrão
 - Cada classe deve ter pelo menos um construtor. Se não for fornecido um construtor na classe, o compilador cria um construtor padrão que não recebe nenhum argumento quando é invocado.

Classe "Sem" Construtor

```
public class Pessoa {  
  
    private String nome;  
  
    public String getNome(){  
        return nome;  
    }  
  
    public void setNome(String nome){  
        this.nome = nome;  
    }  
}
```

O que será impresso?

```
3 public class Principal {  
4  
5     public static void main(String args[]){  
6  
7         Pessoa p1 = new Pessoa();  
8         System.out.println(p1.getNome());  
9     }  
10 }
```

Construtores

- Construtores padrão

- Se uma classe declara , pelo menos, um construtor, o compilador não criará um construtor padrão para ela. Se o programador desejar, ele pode especificar uma inicialização padrão de objetos de sua classe.
 - **Vantagem:** Todos os objetos são inicializados com os mesmos valores
Qualquer objeto instanciado possuirá os mesmos valores iniciais
 - **Como se faz isso:** declarando um construtor sem argumentos

Classe Com Construtor

```
public class Pessoa {  
  
    private String nome;  
  
    public Pessoa(String nomeTeste){  
        nome = nomeTeste;  
    }  
  
    public String getNome(){  
        return nome;  
    }  
  
    public void setNome(String nome){  
        this.nome = nome;  
    }  
}
```

A classe Pessoa possui um construtor. Logo, o compilador do Java não irá criar um construtor para ela

Prática

Prática 01

■ Exercício

- Escreva uma classe chamada classe “Comparavel” que tem como atributo um valor inteiro, um construtor para inicializar o atributo e o método comparaValor, que recebe como atributo um valor inteiro “valor” e verifica se ele é maior, igual ou menor que aquele do objeto instanciado.
 - Faça a verificação e imprima o resultado

Solução

```
3 public class Comparavel {
4
5     private int x;
6
7     public Comparavel(int valor){
8         x = valor;
9     }
10
11     public void comparaValor(int valor){
12         if(x > valor){
13             System.out.println("Valor " + valor + " é menor que " + x);
14         }
15         else if(x < valor){
16             System.out.println("Valor " + valor + " é maior que " + x);
17         }
18         else{
19             System.out.println("Valor " + valor + " é igual a " + x);
20         }
21     }
22 }
```


Principal - Exemplo

```
2
3 public class Principal {
4
5     public static void main(String args[]) {
6
7         Comparavel c1 = new Comparavel(10);
8         c1.comparaValor(10);
9     }
10 }
```

Problems Javadoc Declaration Console Progress

<terminated> Principal (14) [Java Application] C:\Program Files\Java\jdk1.6.0_21\jre\bin
Valor 10 é igual a 10

Exemplos e Implicações

Construtores

```
public class Pessoa {  
  
    private String nome;  
  
    public Pessoa(String nome) {  
        this.nome = nome;  
    }  
}
```

Classe Pessoa. Observe a presença de um construtor (com argumento)

```
public class Executa {  
  
    public static void main(String args[]){  
        Pessoa p1 = new Pessoa("NomeTeste");  
    }  
}
```

```
public class Executa {  
  
    public static void main(String args[]){  
        Pessoa p1 = new Pessoa();  
    }  
}
```

The constructor Pessoa() is undefined

3 quick fixes available:

- + Add argument to match 'Pessoa(String)'
- Change constructor 'Pessoa(String)': Remove parameter 'String'
- Create constructor 'Pessoa()'

Construtores

A cada objeto instanciado, um valor para o campo nome deve ser, obrigatoriamente, fornecido (caso contrário gera-se um erro)

```
public class Executa {  
  
    public static void main(String args[]){  
        Pessoa p1 = new Pessoa("NomeTeste1");  
        Pessoa p2 = new Pessoa("NomeTeste2");  
        Pessoa p3 = new Pessoa("NomeTeste3");  
        Pessoa p4 = new Pessoa("NomeTeste4");  
  
    }  
}
```

```
public class Executa {  
  
    public static void main(String args[]){  
        Pessoa p1 = new Pessoa("NomeTeste1");  
        Pessoa p2 = new Pessoa("NomeTeste2");  
        Pessoa p3 = new Pessoa("NomeTeste3");  
        Pessoa p4 = new Pessoa();  
  
    }  
}
```

The constructor Pessoa() is undefined

3 quick fixes available:

- + Add argument to match 'Pessoa(String)'
- Change constructor 'Pessoa(String)': Remove parameter 'String'
- Create constructor 'Pessoa()'

Construtores

```
public class Pessoa {  
  
    private String nome;  
  
    public Pessoa() {  
        this.nome = "Nome Teste";  
    }  
}
```

Observe a presença de um construtor, mas sem argumentos (parâmetros)

```
public class Pessoa {  
  
    private String nome;  
  
    public Pessoa(String nome) {  
        this.nome = nome;  
    }  
}
```

Observe a presença de um construtor, mas com argumentos (parâmetros)

Construtores

1ª possibilidade - Sem Construtor

O compilador cria um construtor padrão para a classe
As variáveis de instância são inicializadas com seus valores padrão

```
public class Pessoa {  
  
    private String nome;  
  
}
```

```
public class Executa {  
  
    public static void main(String args[]){  
        Pessoa p1 = new Pessoa();  
    }  
  
}
```

Resumo Construtores

Construtor Inexistente

1ª possibilidade - Sem Construtor

O compilador cria um construtor padrão para a classe
As variáveis de instância são inicializadas com seus valores padrão

```
public class Pessoa {  
  
    private String nome;  
  
}
```

```
public class Executa {  
  
    public static void main(String args[]){  
        Pessoa p1 = new Pessoa();  
    }  
  
}
```


Construtor Sem Argumentos

2ª possibilidade - Construtor sem Argumentos

O programador pode definir um valor padrão para as variáveis

```
public class Pessoa {  
  
    private String nome;  
  
    public Pessoa() {  
        this.nome = "Nome Teste";  
    }  
}
```

```
public class Executa {  
  
    public static void main(String args[]) {  
        Pessoa p1 = new Pessoa();  
    }  
}
```

Construtor Com Argumentos

3ª possibilidade - Com Construtor (Com Argumentos)

O compilador não cria um construtor padrão para a classe

As variáveis de instância são inicializadas os valores passados no construtor

```
public class Pessoa {  
  
    private String nome;  
  
    public Pessoa(String nome){  
        this.nome = nome;  
    }  
  
}
```

```
public class Executa {  
  
    public static void main(String args[]){  
        Pessoa p1 = new Pessoa("NomeTeste1");  
    }  
  
}
```

Construtor Com Alguns Argumentos

4ª possibilidade - Com Construtor (Com Alguns Argumentos de Inicialização)

O compilador não cria um construtor padrão para a classe

As variáveis de instância são inicializadas os valores passados no construtor

As variáveis não inicializadas pelo construtor são inicializadas com seus valores padrão

```
public class Pessoa {  
  
    private String nome;  
    private String sobrenome;  
  
    public Pessoa(String nome) {  
        this.nome = nome;  
    }  
  
    public String getNome() {  
        return nome;  
    }  
  
    public String getSobrenome() {  
        return sobrenome;  
    }  
}
```

```
public class Executa {  
  
    public static void main(String args[]) {  
        Pessoa p1 = new Pessoa("NomeTeste1");  
    }  
}
```

Construtores Sobrecarregados

Construtores Sobrecarregados

```
public class Pessoa {  
  
    private String nome;  
    private String sobrenome;  
  
    public Pessoa(String nome) {  
        this.nome = nome;  
    }  
  
    public Pessoa(String nome, String sobrenome) {  
        this.nome = nome;  
        this.sobrenome = sobrenome;  
    }  
}
```

Sobrecarga de construtores

Sobrecarga de métodos

Métodos de nomes iguais, mas assinaturas diferentes
São métodos diferentes, que apenas possuem o mesmo nome


Construtores Sobrecarregados

```
class Pessoa {  
    String rg;  
    int cpf;  
  
    Pessoa(String rg) {  
        this.rg = rg;  
    }  
  
    Pessoa(int cpf) {  
        this.cpf = cpf;  
    }  
}
```

```
// Chamando o primeiro construtor  
Pessoa p1 = new Pessoa("123456X");  
// Chamando o segundo construtor  
Pessoa p2 = new Pessoa(123456789);
```

Construtores Sobrecarregados

```
public class Pessoa {  
  
    private String nome;  
    private String sobrenome;  
  
    public Pessoa() {  
        this(null, null);  
    }  
  
    public Pessoa(String nome) {  
        this(nome, null);  
    }  
  
    public Pessoa(String sobrenome) {  
        this(null, sobrenome);  
    }  
  
    public Pessoa(String nome, String sobrenome) {  
        setName(nome);  
        setSobrenome(sobrenome);  
    }  
}
```

 Duplicate method Pessoa(String) in type Pessoa

1 quick fix available:

 Rename method 'Pessoa'



Mescla de Construtores

```
public class Pessoa {  
  
    private String nome;  
  
    public Pessoa() {  
        this("teste");  
    }  
  
    public Pessoa(String nome) {  
        setNome(nome);  
    }  
  
    public String getNome() {  
        return nome;  
    }  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
}
```

```
public class Executa {  
  
    public static void main(String args[]) {  
        Pessoa p1 = new Pessoa("NomeTeste1");  
        Pessoa p2 = new Pessoa();  
  
        System.out.println(p1.getNome());  
        System.out.println(p2.getNome());  
    }  
}
```

- 1 - Construtor sem argumentos
invoca construtor com argumento
- 2 - Construtor com argumentos
invoca método setNome
- 3 - Método getNome
- 4 - Método setNome

Construtor Chamando Construtor

```
1 // arquivo: Conta.java
2 class Conta {
3     int numero;
4     double limite;
5
6     Conta(int numero) {
7         this.numero = numero;
8     }
9
10    Conta(int numero, double limite) {
11        this(numero);
12        this.limite = limite;
13    }
14 }
```

Na linha 11, o `this(numero)` evita que o programador escreva novamente tudo que foi escrito no outro construtor. Neste caso, evita-se que seja escrito `this.numero = numero`.

Permite reusar construtores já escritos para inicializar atributos

Exercícios

Exercício 01

■ Exercício 01

- Escreva uma classe em Java que simule uma calculadora bem simples. Essa classe deve ter como atributos duas variáveis Double e uma String. Deve possuir um construtor que recebe como parâmetro dois números e um caracter, correspondente a uma das operações básicas (+, -, *, /). Deve ter um método para calcular a operação desejada e um para imprimir o resultado. O programa deve considerar divisões por zero como sendo erros, e imprimir uma mensagem adequada.
- Excreva uma classe de Teste para testar seu programa.
 - Teste as quatro operações
 - Teste uma divisão por 0

Solução

```
3 public class Calculadora {
4
5     private double valor01;
6     private double valor02;
7     private String oper;
8
9     public Calculadora(int v1, int v2, String op){
10         this.valor01 = v1;
11         this.valor02 = v2;
12         this.oper = op;
13     }
14
15     public void calcularOperacao(){
16         double resultado = 0;
17
18         if(oper == "*"){
19             resultado = valor01 * valor02;
20         }
21         else if(oper == "+"){
22             resultado = valor01 + valor02;
23         }
24         else if(oper == "-"){
25             resultado = valor01 - valor02;
26         }
```

Solução - Continuação

```
27         else if(oper == "/"){
28             if (valor02 == 0){
29                 System.out.println("Divisão por 0 - Inválida");
30             }
31             else{
32                 resultado = valor01 / valor02;
33             }
34         }
35
36         imprimirresultado(resultado);
37     }
38
39     public void imprimirresultado(double resultado){
40         System.out.println("Resultado da operação: " + resultado);
41     }
42
43 }
```

Solução - Classe Principal

```
3 public class Principal {  
4  
5     public static void main(String args[]) {  
6  
7         Calculadora c1 = new Calculadora(100,10,"/");  
8         c1.calcularOperacao();  
9     }  
10 }
```

Construtores de Classes

Orientação a Objetos - DCC025

Prof. Edmar Welington Oliveira
edmar.oliveira@ufjf.edu.br

Universidade Federal de Juiz de Fora - UFJF
Departamento de Ciência da Computação - DCC