PROGRAMAÇÃO I - CCT0827

Semana Aula: 4

Unidade 2 - Conceitos de orientação a objetos

Tema

Métodos Construtores e Polimorfismo de Sobrecarga

Palavras-chave

Método Construtor, Polimorfismo, Polimorfismo de Sobrecarga

Objetivos

O aluno deverá ser capaz de:

·Compreender e aplicar o conceito de métodos construtores.

· Compreender e aplicar os conceitos de polimorfismo de sobrecarga.

Estrutura de Conteúdo

Métodos Construtores

A programação orientada a objetos permite que possamos controlar a criação de um objeto, através dos chamados métodos construtores.

São métodos especiais que são executados apenas uma vez por cada objeto criado, pois só é executado no momento da instanciação / criação do objeto e é responsável por realizar as ações necessárias para a sua criação (controlar a criação do objeto).

Características:

•São sempre públicos (public, característica de encapsulamento – será visto mais adiante em nosso conteúdo), não podendo ter nenhum tipo de restrição;

• Não existe definição de tipo de retorno, pois métodos construtores não podem retornar valores com a instrução "return", são sem tipo;

• Devem ser identificados sempre com o mesmo nome da classe;

•São executados exclusivamente durante o processo de criação / instanciação do objeto, não podendo ser usados pelo objeto após a sua criação.

Exemplo:

Classe **Pessoa**

```
public class Pessoa {
String nome, identidade;
int idade;
public Pessoa(String nome, String identidade, int idade) {
setNome( nome );
setIdentidade( identidade );
setIdade( idade );
}
public String getNome() {
return nome;
}
public void setNome( String no ) {
if (!no.isEmpty()){
nome = no;
}
}
public String getIdentidade() {
return identidade;
}
public void setIdentidade( String id ) {
if (!id.isEmpty()){
identidade = id;
}
}
public int getIdade() {
```

```
return idade;

}

public void setIdade( int id ) {

if (id > 0){

idade = id;

}

}

public void imprimir() {

System.out.println( "Pessoa:");

System.out.println("Nome = " + nome);

System.out.println("Identidade = " + identidade);

System.out.println("Idade = " + idade);

}

}
```

Aplicação: **AppPessoa**

```
public class AppPessoa {

public static void main(String[] args) {

// TODO Auto-generated method stub

//Pessoa p1 = new Pessoa();

Pessoa p2 = new Pessoa("Marcela", "0123456-7", 23);

p2.imprimir();

}

}
```

Observação: Quando não temos um construtor em uma classe, um construtor VAZIO é criado no processo de compilação.

Polimorfismo de Sobrecarga.

Polimorfismo: quer dizer “muitas formas”, é permite o emprego de operadores e identificadores de várias formas, sendo então necessária uma contextualização para que seja realizada a operação adequada. Este contexto está ligado ao emprego do operador, dom método, etc, de acordo com uma situação.

1. Polimorfismo de Sobrecarga de Operadores

O operador + é um dos mais usados e por isso é um bom exemplo de Sobrecarga deOperadores, ele pode ser usado de várias e diferentes formas em função do contexto:

1. Concatenação: String nome = "João" + " da " + "Silva";
2. Soma inteira: int a = 3 + 4;
3. Soma real: double b = 1.3 + 2.7;
4. Incremento: x++; ou ++x;
5. Concatenação entre textos e valores: System.out.println("Idade" + p2.getIdade());

Exemplo:

System.out.println("Valor=" + ((3 + 4) + (1.3 + 2.7) + (++x)));

2. Polimorfismo de Sobrecarga de Métodos

A Sobrecarga de métodos permite que possamos ter mais de um método com o mesmo identificador em uma mesma classe. Isso só é possível em razão da avaliação do contexto no momento da execução.

Com o uso da Sobrecarga de métodos você poderá criar quantos métodos com o mesmo identificador (nome) você quiser em uma mesma classe, desde que eles não possuam a mesma assinatura de método.

Exemplos de Polimorfismo de Sobrecarga válidos para uma mesma classe:

```
int meuMetodo( int a, double b, String c ) {

return 1;

}
```

```
int meuMetodo ( double b, String c, int a ){
return 2;
}
int meuMetodo ( String c, int a, double b ) {
return 3;
}
int meuMetodo ( String c, double b, int a ) {
return 4;
}
```

As assinaturas são respectivamente:

```
meuMetodo ( int , double , String )
meuMetodo ( double , String , int )
meuMetodo ( String , int , double )
meuMetodo ( String , double , int )
```

Todos os métodos acima, apesar de possuírem a mesma quantidade de parâmetros, suas assinaturas são diferentes e serão executadas em função de contexto diferentes, respectivamente:

```
int g = meuMetodo( 2, 2.25, "Casa" );
int h = meuMetodo( 2.25, "Casa", 2 );
int i = meuMetodo( "Casa", 2, 2.25 );
int j = meuMetodo( "Casa", 2.25, 2 );
```

Os valores armazenados em g, h, i e j serão respectivamente: 1, 2, 3 e 4.

Sobrecarga de Métodos Construtores

Métodos construtores são métodos e sendo assim, também podem ser sobrecarregados.

Uma classe que possui mais de um método construtor é uma classe que oferece diferentes formas de criação para os seus objetos.

Estratégias de Aprendizagem

Implementar programas em Java, explorando os conceitos de polimorfismo de sobrecarga.

Exemplo de Sobrecarga de métodos de construtores

Classe: **Carro**

```
import java.util.Scanner;

public class Carro {

// use as regras da boa prática em programação Java

// para os identificadores da classe, dos atributos e dos métodos

String fabricante, modelo, cor, placa;

double valor;

int numeroPortas, anoFabricacao, anoModelo;

public Carro() { }

public Carro(String placa, double valor) {

this.placa = placa;

this.valor = valor;

}

public Carro(String modelo, String cor, String placa, double valor) {

this.modelo = modelo;

this.cor = cor;

this.placa = placa;

this.valor = valor;

}

public Carro(String fabricante, String modelo, String cor,
```

```
String placa, double valor) {
this.fabricante = fabricante;
this.modelo = modelo;
this.cor = cor;
this.placa = placa;
this.valor = valor;
}
public Carro(String fabricante, String modelo, String cor,
String placa, double valor, int numeroPortas,
int anoFabricacao, int anoModelo) {
this.fabricante = fabricante;
this.modelo = modelo;
this.cor = cor;
this.placa = placa;
this.valor = valor;
this.numeroPortas = numeroPortas;
this.anoFabricacao = anoFabricacao;
this.anoModelo = anoModelo;
}
public String getFabricante () {
return fabricante;
}
public void setFabricante (String fab) {
if(!fab.isEmpty()) {
fabricante = fab;
```

```
}
}
public String getModelo () {
return modelo;
}
public void setModelo (String mod) {
if(!mod.isEmpty()) {
modelo = mod;
}
}
public String getCor () {
return cor;
}
public void setCor (String co) {
if(!co.isEmpty()) {
cor = co;
}
}
public String getPlaca () {
return placa;
}
public void setPlaca (String pla) {
if(!pla.isEmpty()) {
placa = pla;
}
```

```
}
public double getValor () {
return valor;
}
public void setValor (double val) {
if(val > 0) {
valor = val;
}
}
public int getNumeroPortas () {
return numeroPortas;
}
public void setNumeroPortas (int nump) {
if(nump > 0) {
numeroPortas = nump;
}
}
public int getAnoFabricacao () {
return anoFabricacao;
}
public void setAnoFabricacao (int anof) {
if(anof > 0) {
anoFabricacao = anof;
}
}
```

```
public int getAnoModelo () {
return anoModelo;
}
public void setAnoModelo (int anom) {
if(anom > 0) {
anoModelo = anom;
}
}
public void imprimir (){
System.out.println( "Fabricante : " + getFabricante() );
System.out.println( "Modelo : " + getModelo() );
System.out.println( "Cor : " + getCor() );
System.out.println( "Placa : " + getPlaca() );
System.out.println( "Valor : " + getValor() );
System.out.println( "Número de Portas : " + getNumeroPortas() );
System.out.println( "Ano de fabricação: " + getAnoFabricacao() );
System.out.println( "Ano do Modelo : " + getAnoModelo() );
}
public void entradaDados () {
Scanner entrada = new Scanner( System.in );
// O objeto Scanner deve ficar local ao método
// o objeto Scanner para entrada de dados não é um atributo do carro
// é apenas um objeto auxiliar a entrada de dados
System.out.println("Digite o Fabricante do carro :");
setFabricante( entrada.nextLine() );
```

```
System.out.println("Digite o Modelo do carro :");
setModelo( entrada.nextLine() );
System.out.println("Digite a Cor do carro :");
setCor( entrada.nextLine() );
System.out.println("Digite a Placa do carro :");
setPlaca( entrada.nextLine() );
System.out.println("Digite o Valor do carro :");
setValor( Double.parseDouble( entrada.nextLine()) );
System.out.println("Digite o Número de Portas do carro :");
setNumeroPortas( Integer.parseInt( entrada.nextLine()) );
System.out.println("Digite o Ano de fabricação do carro :");
setAnoFabricacao( Integer.parseInt( entrada.nextLine()) );
System.out.println("Digite o Ano do Modelo do carro :");
setAnoModelo( Integer.parseInt( entrada.nextLine()) );
}
}
```

Aplicação antiga **AppCarro**

```
public class AppCarroNovo {
public static void main(String[] args) {
// TODO Auto-generated method stub
Carro car1 = new Carro();
car1.entradaDados();
car1.imprimir();
Carro car2 = new Carro("AAA1A00", 25000);
car2.imprimir();
```

```
Carro car3 = new Carro("Logan", "Azul", "ABC1E00", 32000);

car3.imprimir();

Carro car4 = new Carro("Audi", "A5", "Prata", "AUD0I00", 123000);

car4.imprimir();

Carro car5 = new Carro("Fiat", "Argo", "Verde", "ABB1I00", 42000, 5,

2018, 2019);

car5.imprimir();

}

}
```

Indicação de Leitura Específica

Polimorfismo de Sobrecarga de Métodos:

Sobrecarga e sobreposição de métodos em orientação a objetos - https://www.devmedia.com.br/sobrecarga-e-sobreposicao-de-metodos-em-orientacao-a-objetos/33066

Métodos Construtores:

FURGERI, Sérgio. Java 8 – ensino didático: desenvolvimento e implementação de aplicações. São Paulo: Érica, 2015. [Páginas: 115 – 116]

Sobrecarga:

FURGERI, Sérgio. Java 8 – ensino didático: desenvolvimento e implementação de aplicações. São Paulo: Érica, 2015. [Páginas: 96 – 97]

Aplicação: articulação teoria e prática

Exercício:

A classe Computador possui os atributos e métodos a seguir, crie a classe Computador e a aplicação AppComputador. A aplicação deverá criar 10 diferentes objetos Computador, utilizando diferentes construtores para a criação de cada um.

Classe: **Computador**

Atributos	**Métodos**

Fabricante : texto
Modelo : texto
Peso : real
Preço : real
QuantidadeMemória : inteiro
Processador : texto
CapacidadeArmazenamento: inteiro
	- Setters para todos os atributos
	- Getters para todos os atributos
	- Pelo menos 12 métodos construtores
	- Imprimir () // imprime todos os dados do carro
	- entradaDados () // entrada de dados pelo teclado

Considerações Adicionais