Programação Orientada a Objetos

Educação a Distância
Cruzeiro do Sul Educacional
Campus Virtual

Material Teórico

Construtores, Sobrecarga e Herança

Responsável pelo Conteúdo:
Prof. Esp. Alexander Gobbato Albuquerque

Revisão Textual:
Profa. Dra. Patrícia S. Leite Di Iório

UNIDADE Construtores, Sobrecarga e Herança

- Construtores
- Sobrecarga
- Herança

Objetivo de APRENDIZADO

Iremos enriquecer nosso conhecimento com novos conceitos, veremos qual a importância dos construtores, sobrecarga e herança. Veremos que isso é de fácil implementação e adaptação.

Hoje, veremos alguns assuntos introdutórios na nossa disciplina e aproveito para apresentar-lhes alguns conceitos que utilizaremos na estrutura de todas as nossas unidades.

Para obter um bom aproveitamento, vamos conferir a estrutura desta unidade?

Conteúdo Teórico: neste link você encontrará o material principal de estudos na forma de texto escrito.

Atividade de Sistematização: os exercícios disponibilizados são de autocorreção e visam que você pratique o que aprendeu na disciplina e que identifique os pontos em que precisa prestar mais atenção, ou pedir esclarecimentos a seu tutor. Além disso, as notas atribuídas aos exercícios serão parte de sua média final na disciplina.

Atividade de Aprofundamento: é uma atividade dissertativa.

Material Complementar e Referências Bibliográficas: nestes links você poderá ampliar seus conhecimentos.

Vídeoaula: nestes links serão apresentadas algumas ferramentas na prática e também a resolução de alguns exercícios de forma prática.

Atenção

Lembramos-lhe da importância de realizar todas as atividades propostas dentro do prazo estabelecido para cada Unidade, dessa forma, você evitará que o conteúdo se acumule e que você tenha problemas ao final do semestre.

Uma última recomendação, caso tenha problemas para acessar algum item da disciplina, ou dúvidas com relação ao conteúdo, não deixe de entrar em contato com seu professor-tutor por meio do item mensagens.

Contextualização

Já estudamos, nos capítulos anteriores, o encapsulamento de informações, a reutilização de códigos. Vimos, também, como criar os métodos, usar encapsulamento e chamar os métodos criados através de objetos.

Com os conhecimentos adquiridos, podemos utilizar as técnicas de criação de construtores, que são os métodos invocados na criação dos objetos. Veremos a sobrecarga de métodos, passando por novas assinaturas e também por herança, em que as características são passadas para as classes que desejam utilizar a codificação já desenvolvida.

Construtores

Construtores são métodos que são invocados automaticamente quando um objeto é instanciado. São métodos que nunca retornam nada e não possuem tipo. Em todas as instanciações foram utilizadas a palavra reservada, conforme o exemplo abaixo:

Carro c1 = new Carro();

Conforme comentado nas unidades anteriores, a palavra new é a responsável por inicializar o objeto. O método construtor é o método que é chamado toda vez que o objeto é criado, ou seja, quando se utiliza o operador new, o primeiro método que é chamado é o construtor do objeto. Ele é responsável por disponibilizar ou alocar o espaço em memória para a utilização do novo objeto.

Obrigatoriamente, o método construtor deve possuir o mesmo nome da classe, se ele não for declarado, por *default* é criado automaticamente.

Logo abaixo demonstrado-se a codificação de dois arquivos:

Arquivo classe:

```
class Carro{
    private codigo;
    private nome;
    private marca;

    Carro(){ // MÉTODO CONSTRUTOR, MESMO NOME DA CLASSE
        marca = "VALOR PADRÃO"; //INICIALIZA A MARCA COM UM VALOR PADRÃO
    }

    public void MostraMarca(){
        System.out.println("Marca: " + marca);
    }
}
```

Arquivo para instanciar a classe e invocar o método construtor:

```
class UsaCarro{

    public static void main(String args[]){
        Carro c1 = new Carro(); //INVOCA O MÉTODO CONSTRUTOR
        c1.MostraMarca();
    }
}
```

O método construtor funciona como qualquer método criado em um programa Java, pode receber argumentos no momento da criação, veja os exemplos abaixo.

Suponhamos a necessidade de criar dois objetos da classe carro, com diferentes conteúdos para a variável nome.

Classe carro com recebimento de argumentos

```
class Carro{
    private codigo;
    private nome;
    private marca;

    Carro(String n){ // MÉTODO CONSTRUTOR, MESMO NOME DA CLASSE
        nome = n; //INICIALIZA O NOME COM UM VALOR INFORMADO NO PROGRAMA
        marca = "VALOR PADRÃO"; //INICIALIZA A MARCA COM UM VALOR PADRÃO
    }

    public void ExibeInf(){
        System.out.println("Nome: " + nome);
        System.out.println("Marca: " + marca);
    }
}
```

Classe UsaCarro com passagem de valores:

```
class UsaCarro{

    public static void main(String args[]){
        Carro c1 = new Carro("Fiesta"); //INVOCA O MÉTODO CONSTRUTOR
        Carro c2 = new Carro("EcoSport"); //INVOCA O MÉTODO CONSTRUTOR
        c1.ExibeInf();
        c2.ExibeInf();
    }
}
```

Em Java, existe uma referência para a própria classe através da palavra-chave "this". Assim, no construtor ao invés de usar "nome = n", podemos usar "this.nome = nome", nesse caso as mesmas variáveis no construtor podem ser usadas nos atributos.

```
class Carro{
    private codigo;
    private nome;
    private marca;

    Carro(String nome, String marca){ // MÉTODO CONSTRUTOR, MESMO NOME DA CLASSE
        this.nome = nome; //INICIALIZA O NOME COM UM VALOR INFORMADO NO PROGRAMA
        this.marca = marca; //INICIALIZA A MARCA COM UM VALOR PADRÃO
    }

    public void ExibeInf(){
        System.out.println("Nome: " + nome);
        System.out.println("Marca: " + marca);
    }
}
```

Agora, quando instanciamos um objeto da classe Carro, veja como fica:

```
class UsaCarro{

    public static void main(String args[]){
        Carro c1 = new Carro("Fiesta","Ford"); //INVOCA O MÉTODO CONSTRUTOR
        Carro c2 = new Carro("EcoSport","Ford"); //INVOCA O MÉTODO CONSTRUTOR
        c1.ExibeInf();
        c2.ExibeInf();
    }
}
```

Sobrecarga

Sobrecarregar (do inglês *overload*) trata-se de um método para criar mais métodos com o mesmo nome, porém com assinaturas diferentes. Os parâmetros podem se diferenciar em tipo e/ou em quantidade, assim como o tipo de retorno. Ficará a cargo do compilador escolher de acordo com a lista de argumentos enviados os métodos a serem executados. Vejamos os exemplos abaixo:

```
public class Operadores{
    public void multiplicar(float num1, float num2){
        System.out.println("Multiplicação: " + num1 * num2);
    }
}
```

O método recebe dois parâmetros e o retorno é o resultado impresso na tela, veja o método implementado.

```
public class UsaOperadores{
    public static void main(String args[]){
        Operadores op = new Operadores();
        op.multiplicar(1.5, 3.85);
    }
}
```

Agora, imagine que precisamos fazer um método que multiplique números inteiros e não números reais. Para o usuário, é transparente, o programador deve realizar uma sobrecarga de métodos e desenvolveria da seguinte forma:

```
public class Operadores{
    public void multiplicar (float num1, float num2){
        System.out.println("Multiplicação: " + (num1*num2));
    }

    public void multiplicar (int num1, int num2){
        System.out.println("Multiplicação: " + (num1*num2));
    }
}
```

As possibilidades são ilimitadas e a sobrecarga vai depender das necessidades de cada projeto de classe. Em Java, muitos métodos já são sobrecarregados. Por exemplo, o próprio comando de imprimir que sempre usamos "System.out.println" possui várias sobrecargas. Você pode passar como parâmetro um número inteiro, um número real ou mesmo uma String que ele consegue imprimir na tela o que passou. Isso quer dizer que existe um método println que recebe um int, outro que recebe um double, outro que recebe uma String, além de outros tipos.

Herança

Como o nome sugere, na orientação a objeto, se refere a algo que será herdado. Em Java ocorre quando uma classe herda todas as características, métodos e atributos de outra classe, essa técnica possibilita o compartilhamento e reaproveitamento de recursos definidos anteriormente. A classe principal que irá disponibilizar todos os recursos recebe o nome de superclasse e a classe que herda recebe o nome de subclasse.

Outro recurso da herança, permite que a classe que está herdando, ou subclasse possa aproveitar toda a característica da superclasse, como também implementar novos atributos e métodos.

Essa técnica de herança é muito utilizada em Java, para melhor entendimento sobre herança, observe os exemplos abaixo:

Classe veículo:

```
1  class Veiculo{
2      private String nome;
3      private float velocidade;
4
5      public void setNome(String nome){
6          this.nome = nome;
7      }
8
9      public String getNome(){
10         return this.nome;
11     }
12
13     public void setVelocidade(float velocidade){
14         this.velocidade = velocidade;
15     }
16
17     public float getVelocidade(){
18         return velocidade;
19     }
20
21     public void acelera(){
22         if(velocidade<=10){
23             velocidade++;
24         }
25     }
26
27     public void frea(){
28         if(velocidade>0)
29             velocidade--;
30     }
31 }
```

A classe Veículo possui duas variáveis de instância (nome e velocidade) e seis métodos. Existe a necessidade de outra classe utilizar as mesmas características da classe Veiculo, mas também implementará métodos que não foram desenvolvidos na superclasse, como por exemplo, os métodos de ligar e desligar. Com a técnica da herança, as características da classe serão herdadas na nova classe e, para realizar a herança, utiliza-se a palavra *extends*, ou seja a subclasse *extends* superclasse, veja o exemplo abaixo:

```
1  class Veiculo1 extends Veiculo{
2      private boolean ligado;
3
4      public void liga(){
5          ligado = true;
6      }
7
8      public void desliga(){
9          ligado = false;
10     }
11 }
```

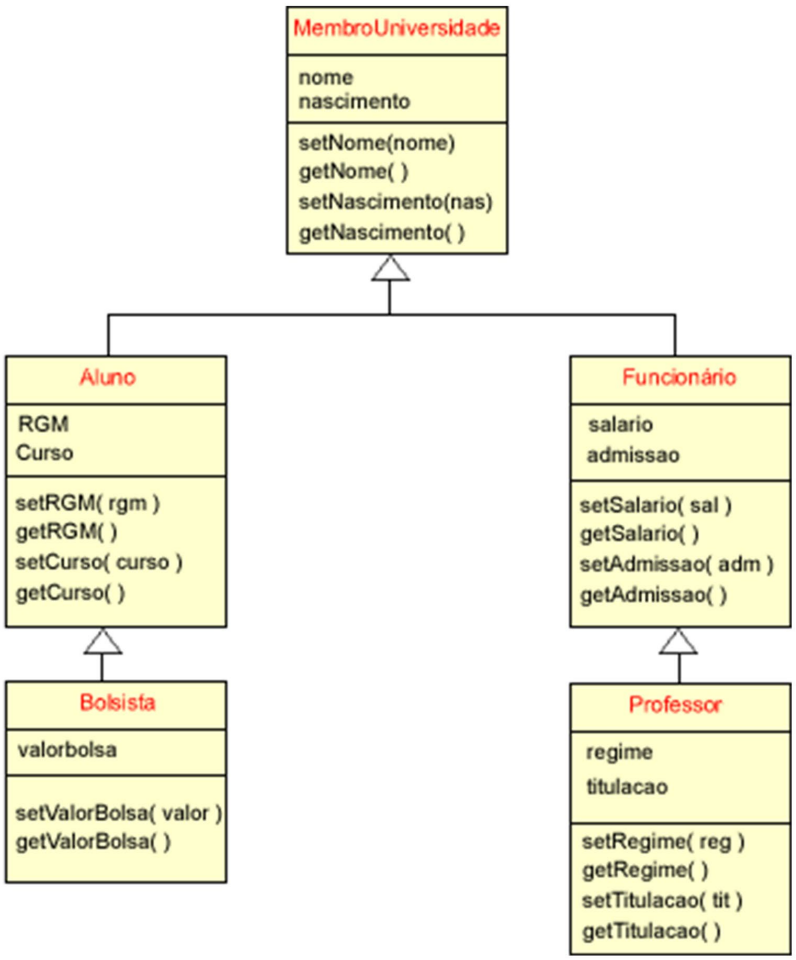
A classe Veículo1 estende (*extends*) as funcionalidades da classe Veículo, ou seja, a classe Veículo1 está herdando os métodos *set e get* e também os métodos acelera() e frea().

Veja a implementação do código:

```
1  import javax.swing.*;
2  public class UsaVeiculo
3  {
4      public static void main(String args[])
5      {
6
7          Veiculo1 v = new Veiculo1();
8
9          v.liga();
10
11         v.setNome(JOptionPane.showInputDialog("Digite o nome:"));
12
13         v.setVelocidade(Integer.parseInt(JOptionPane.showInputDialog("Digite a Velocidade:")));
14
15         JOptionPane.showMessageDialog(null,"Velocidade Atual: " + v.getVelocidade());
16
17         v.acelera();
18
19         JOptionPane.showMessageDialog(null,"Velocidade Atual: "+ v.getVelocidade() );
20
21         v.frea();
22
23         JOptionPane.showMessageDialog(null,"Velocidade Atual: "+ v.getVelocidade() );
24
25         v.desliga();
26     }
27 }
```

Ao instanciar o objeto v, o mesmo receberá as variáveis de instância e todos os métodos presentes nas classes Veículo e Veículo1, desse modo o objeto v terá acesso as variáveis nome, velocidade, ligado e os métodos frea, acelera, liga e desliga.

Com base na modelagem UML abaixo, veremos como ficará o desenvolvimento das classes MembroUniversidade, Aluno, Bolsista, Funcionário e Professor.



Logicamente, começaremos pela classe principal (superclasse).

```
public class MembroUniversidade {
    private String nome;
    private String nascimento;

    public MembroUniversidade(String nnome, String nnascimento) {
        nome = nnome;
        nascimento = nnascimento;
    }
    public MembroUniversidade() {
        nome = "";
        nascimento = "";
    }
    public void setNome(String nnome) {
        nome = nnome;
    }
    public String getNome() {
        return nome;
    }
    public void setNascimento(String nnascimento) {
        nascimento = nnascimento;
    }
    public String getNascimento() {
        return nascimento;
    }
}
```

Vamos, agora, implementar a classe Aluno que herda todas as características de MembroUniversidade. Observe a palavra *extends*, essa palavra indica que a classe Aluno herda as características da classe MembroUniversidade. Pode-se observar que a classe aluno tem seus próprios atributos como RGM e curso. A palavra chave "super" refere-se à superclasse da classe. O comando "super()" está chamando o construtor da classe pai, nesse exemplo a classe pai é a MembroUniversidade.

```
public class Aluno extends MembroUniversidade {
    private String RGM;
    private String curso;

    public Aluno(String no, String na, String r, String cur) {
        super(no, na);
        RGM = r;
        curso = cur;
    }

    public Aluno()  {
        super( );
        RGM = "";
        curso = "";
    }

    public void setRGM(String r) { ... }

    public String getRGM( ) { ... }

    public void setCurso(String cur) { ... }

    public String getCurso( ) { ... }
}
```

Veja, agora, como fica a classe Bolsista que será subclasse de Aluno.

```
public class Bolsista extends Aluno {
    private float valorbolsa;

    public Bolsista(String no, String na, String r, String cur, float v) {
        super(no, na, r, cur);
        valorbolsa = v;
    }

    public Bolsista()  {
        super( );
        valorbolsa = 0;
    }

    public void setValorBolsa(float v) { ... }

    public float getValorBolsa( ) { ... }

}
```

Material Complementar

Livros Disponíveis na Biblioteca Virtual Universitária da Pearson Education:

SINTES, T. **Aprenda Programação Orientada a Objetos em 21 dias.** 1 ed. São Paulo: Pearson Education do Brasil, 2002, v. 1.

DEITEL, P.; DEITEL, H. **Java Como Programar,** 8 ed. São Paulo: Pearson Education do Brasil, 2010.

HORSTMANN, C.S.; CORNELL, G. **Core Java.** 8 ed. São Paulo: Pearson Education do Brasil, 2010, v. 1.

Explore

Conceitos de POO em inglês. Disponível em:

- http://docs.oracle.com/javase/tutorial/java/concepts.

Referências

DEITEL, P.; DEITEL, H. **Java Como Programar,** 8 ed. São Paulo: Pearson Education do Brasil, 2010.

FURGERI, S. **Java 2: Ensino Didático: Desenvolvendo e Implementando Aplicações.** São Paulo: Érica, 2002.

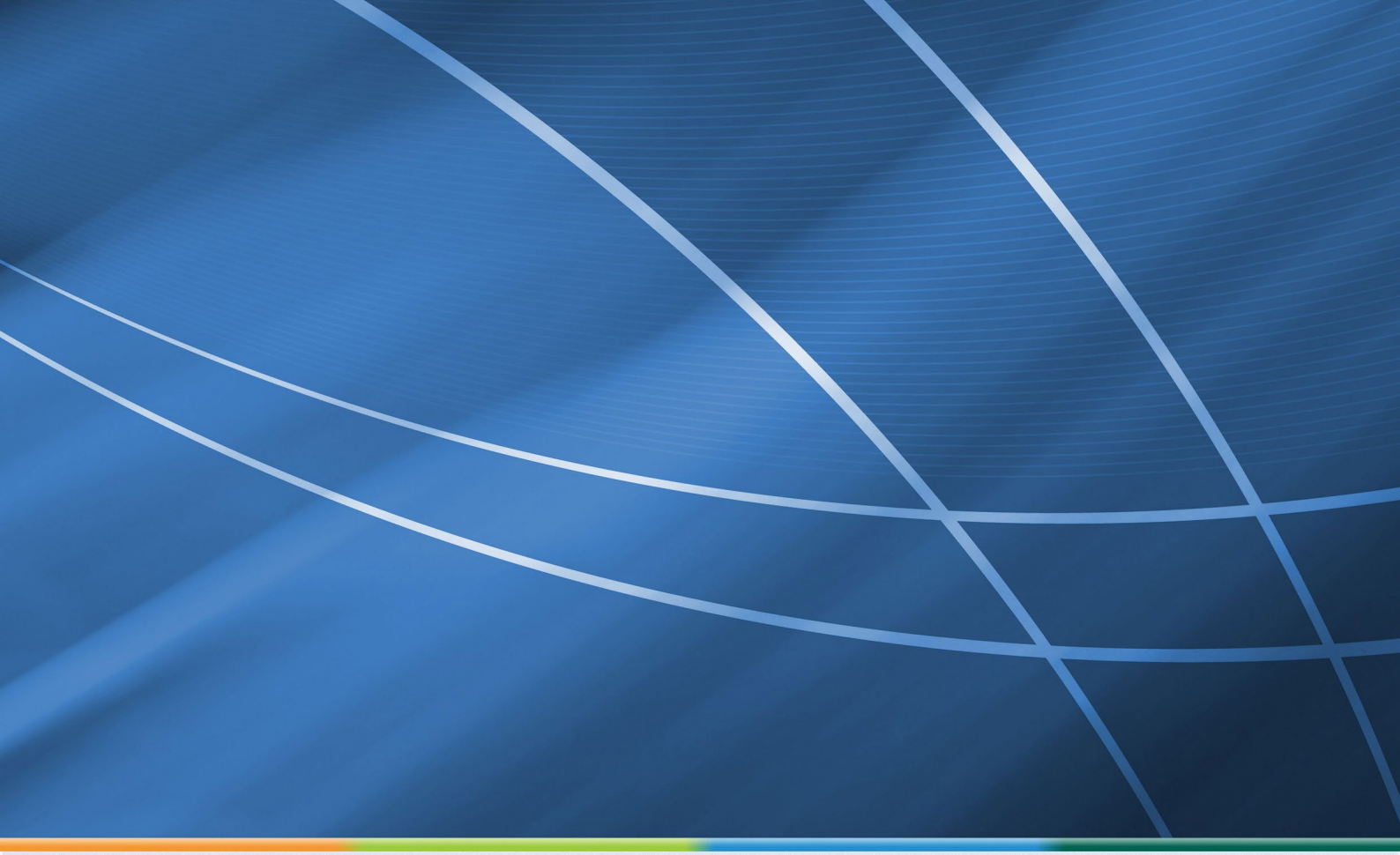
HORSTMANN, C.S.; CORNELL, G. **Core Java.** 8 ed. São Paulo: Pearson Education do Brasil, 2010, v. 1.

SINTES, T. Aprenda **Programação Orientada a Objetos em 21 dias.** 1 ed. São Paulo: Pearson Education do Brasil, 2002, v. 1.

Anotações


Educação a Distância
Cruzeiro do Sul Educacional
Campus Virtual

www.cruzeirodosulvirtual.com.br
Campus Liberdade
Rua Galvão Bueno, 868
CEP 01506-000
São Paulo SP Brasil
Tel: (55 11) 3385-3000



Universidade Cruzeiro do Sul

UNICID Universidade Cidade de S. Paulo

UNIFRAN Universidade de Franca

UDF Centro Universitário

Módulo Centro Universitário