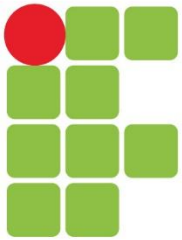


INSTITUTO FEDERAL
SÃO PAULO
Câmpus Araraquara

Disciplina de Programação Orientada a Objetos
Sobrecarga de métodos e construtores

3º Semestre - Curso Superior em ADS

Prof. Msc. Dênis L. Zaniro
deniszaniro@ifsp.edu.br

Sumário

- Objetivos
- Sobrecarga de métodos
- Sobrecarga de construtores
- Exercícios de fixação
- Referências bibliográficas

Objetivos

- Entender o que é sobrecarga de métodos e construtores, como aplicar esse conceito e quais são as implicações decorrentes de sua utilização.

Sobrecarga de métodos

- Problema: É necessário escrever uma classe em Java que permita realizar cálculos aritméticos. Esses cálculos devem ser baseados sempre em dois operandos - ou ambos operandos são valores inteiros ou são valores decimais. Considere inicialmente a operação de soma. Suponha que outras classes necessitem calcular e obter a soma de dois números inteiros ou decimais.
- Como resolver esse problema?

Sobrecarga de métodos

- Hipótese 1: A classe responsável pelos cálculos deve possuir um único método para calcular a soma. Os parâmetros desse método devem ser do tipo decimal.
  - Problemas?
- Hipótese 2: A classe responsável deve definir dois métodos distintos para calcular a soma. Como cada método será baseado em tipos diferentes, cada um deverá ter seu próprio nome.
  - Problemas?

Sobrecarga de métodos

- Hipótese 3: A classe responsável pelos cálculos deve definir dois métodos distintos (com base em tipos diferentes), mas com nomes iguais.
- Esse é o conceito de **sobrecarga de métodos**.
- O que são métodos sobrecarregados?
  - Métodos sobrecarregados são métodos distintos, mas possuem o mesmo nome.
  - Métodos sobrecarregados podem ser definidos na mesma classe ou em uma hierarquia de herança.

Sobrecarga de métodos

- Por que criar métodos sobrecarregados?
  - A sobrecarga de métodos permite fornecer, de forma conveniente, um serviço que seja baseado em tipos diferentes.
- Como o compilador Java distingue um método do outro na sobrecarga?
  - O compilador realiza a distinção com base nos parâmetros dos métodos.
  - São verificados o **tipo** e a **ordem** dos parâmetros.

Sobrecarga de métodos

- Ilustração:

```
public class Calculadora
{
   ...
   public double somar(double op1, double op2) {
     return op1 + op2;
   }
   public int somar(int op1, int op2) {
     return op1 + op2;
   }
}
```

Sobrecarga de métodos

- Ilustração:

```
public class Calculadora
{
  ...
  public double somar(double op1,
                      double op2) {
    return op1 + op2;
  }
  public int somar(int op1,
                   int op2) {
    return op1 + op2;
  }
}
```

```
public class Principal
{
  public static void main(String args[]) {
    Calculadora c = new Calculadora();
    c.somar(2, 4);
    c.somar(2.5, 5);
  }
}
```

Sobrecarga de métodos

- Exemplos:

```
public void x(double a) { ... }
public void x(double a, double b) { ... }
```
OK

```
public void x(double a) { ... }
public void x(int a) { ... }
```
OK

```
public void x(double a, int b) { ... }
public void x(int b, double a) { ... }
```
OK

```
public void x(String a) { ... }
public void x() { ... }
```
OK

Todos esses pares de métodos são exemplos de métodos que possuem o mesmo nome, mas suas listas de parâmetros são diferentes entre si. Em outras palavras, são métodos com **assinaturas** diferentes entre si.

Sobrecarga de métodos

- Há algum problema no código dado a seguir?

```
public class Calculadora
{
  ...
  public double somar(double op1,
                      double op2) {
    return op1 + op2;
  }
  public int somar(double a,
                   double b) {
    return (int)(a + b);
  }
}
```

Existe um erro de compilação nesse código. O motivo é que há dois métodos idênticos que foram definidos na mesma classe. Essa situação permite concluir que o tipo de retorno não faz parte da assinatura de um método.

Sobrecarga de métodos

- Exercício:

```
...
public double somar(double op1,
                    double op2) {
  return op1 + op2;
}                                  1
public int somar(int op1,
                 int op2) {
  return op1 + op2;
}                                  2
public double somar(double op1,
                    int op2) {
  return op1 + op2;
}                                  3
...
```

```
...
public static void main(String args[]) {
  Calculadora c = new Calculadora();
  byte b = 20; short s = 30; int i = 1000;
  long l = 5000l; float f = 3.14f; double d = 200;
  c.somar(b, s);   2
  c.somar(f, s);   3
  c.somar(b, d);   1
  c.somar(l, b);   3
  c.somar(d, l);   1
  c.somar(i, l);   1
  c.somar(i, s);   2
  c.somar(f, f);   1
}
...
```

Sobrecarga de construtores

- Novo problema: É necessário escrever uma classe que represente pessoas. Sabe-se que toda pessoa possui nome e endereço. Para que uma pessoa seja criada por outras classes, deve-se definir, pelo menos, seu nome. Além disso, embora o endereço não seja requisito obrigatório para que uma pessoa exista, algumas pessoas podem ser criadas a partir de seu nome e seu endereço.
- Como resolver esse problema?

Sobrecarga de construtores

- Hipótese 1: A classe de pessoas deve definir um único construtor que possua dois parâmetros. Um parâmetro deve representar o nome e o outro deve representar o endereço.
  - Problemas?
- Hipótese 2: A classe de pessoas deve definir um único construtor cujo parâmetro represente o nome da pessoa que está sendo instanciada.
  - É a melhor solução?

Sobrecarga de construtores

- Hipótese 3: A classe de pessoas deve definir dois construtores. Um construtor deve possuir um único parâmetro e o outro construtor deve possuir dois parâmetros.
- Esse é o conceito de **sobrecarga de construtores**.
- O que significa criar mais de um construtor?
  - Quando uma classe qualquer define mais de um construtor, significa que existe mais de uma forma para que se instanciem objetos dessa classe.

Sobrecarga de construtores

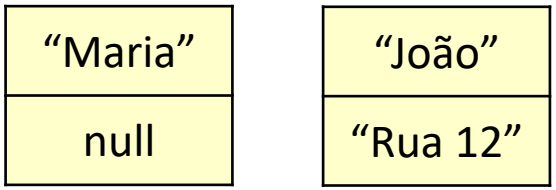
- Quando é necessário definir mais de um construtor em uma classe?
  - Quando é necessário permitir que as classes clientes escolham uma opção dentre um conjunto predefinido de opções para instanciar um objeto.
  - Para o problema anterior, considere que nem todas as classes capazes de instanciar pessoas saberão qual é o endereço da pessoa no momento de sua instanciação.

Sobrecarga de construtores

- Ilustração:

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

```
public class Principal
{
  public static void main(String args[]) {
    Pessoa p1, p2;
    p1 = new Pessoa("Maria");
    p2 = new Pessoa("João", "Rua 12");
  }
}
```



Sobrecarga de construtores

- Questão: Se houvesse a possibilidade de se instanciar uma pessoa de forma completamente *default*, qual alteração seria necessária?
- Seria necessário criar um terceiro construtor, sem parâmetros.
- A existência desse construtor implicaria três formas diferentes de se instanciarem pessoas.

Sobrecarga de construtores

- Ilustração:

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

Sobrecarga de construtores

- Novo requisito: Suponha que, caso uma pessoa seja instanciada de forma incompleta (isto é, sem algum dado), a classe de pessoas deverá atribuir um valor qualquer diferente de null para o atributo em questão.
- Por exemplo, se uma pessoa for instanciada de forma que seu nome ou endereço não tenham sido definidos, o valor atribuído automaticamente deverá ser “”.

Sobrecarga de construtores

- Ilustração:

```
...
  public Pessoa() {
    this.nome = “”;
    this.endereco = “”;
  }
  public Pessoa(String nome) {
    this.nome = nome;
    this.endereco = “”;
  }
  public Pessoa(String nome, String endereco) {
    this.nome = nome;
    this.endereco = endereco;
  }
...
```

Sobrecarga de construtores

- Reflexão:
- Observando o código anterior, há possibilidade de melhoria?
  - A resposta é sim.
  - Existem instruções repetidas entre os construtores.
- E qual é a solução?
  - Especificar instruções que permitam que um construtor chame outro.
  - Isso será possível por meio da instrução **this()**.

Sobrecarga de construtores

- Mas this não é usada para armazenar a referência do objeto corrente?
  - Exatamente.
  - Entretanto, a instrução **this()** pode ser usada para que um construtor chame outro na mesma classe.
- Então, qualquer método de uma classe será capaz de chamar um construtor usando-se this()?
  - A resposta é não.
  - A instrução this() é usada apenas no contexto de construtores.

Sobrecarga de construtores

- Novo código:

```
...
  public Pessoa() {
    this("", "");
  }
  public Pessoa(String nome) {
    this(nome, "");
  }
  public Pessoa(String nome,
                String endereco) {
    this.nome = nome;
    this.endereco = endereco;
  }
...
```

```
public class Principal
{
  public static void main(String args[]) {
    Pessoa p1, p2;
    p1 = new Pessoa("Maria");
    p2 = new Pessoa();
  }
}
```

Regra fundamental:
Quando a instrução **this()** é usada, ela deve ser a primeira instrução em um construtor.

Exercícios de fixação

- Escreva uma classe que forneça um serviço de cálculo do maior número. As classes clientes podem utilizar esse serviço sempre que precisarem calcular o maior de dois números, ou o maior de três números ou ainda o maior de quatro números inteiros. Defina uma solução que leve à reutilização de código.

Exercícios de fixação

- Escreva uma classe que represente livros de uma biblioteca. Sabe-se que todo livro possui título, autor(es), editora, ano de publicação e quantidade de páginas. Para um livro, pode haver, no máximo, dois autores (e, no mínimo, um autor). Com exceção do valor de quantidade de páginas (esse valor pode ou não ser informado durante a instanciação de um livro), os demais dados são obrigatórios para que um livro seja instanciado. Portanto, elabore os construtores necessários para que livros sejam instanciados de maneira adequada. Evite repetição de código entre construtores.

Referências bibliográficas

- SANTOS, Rafael. **Introdução à programação orientada a objetos usando Java**. Elsevier, 2003.
- SIERRA, Kathy; BATES, Bert. **Use a cabeça! Java**. 2. ed. Alta Books, 2009.