


05. Sobrecarga de Métodos

Prof. Alexandre Krohn




Roteiro

- Sobrecarga de métodos
- Passagem de Objetos por parâmetros
- Chamada de construtores dentro da chamada do método



Sobrecarga de métodos

- Um método pode assumir formas diferentes das quais foram implementadas inicialmente e agir de modo diferente quando invocado de forma diferente por outra classe
- Isso também é **polimorfismo**



Sobrecarga de métodos

- Em uma classe, um mesmo nome de método pode possuir listas de parâmetros diferentes
- O tipo de retorno e o nome do método devem ser sempre os mesmos.
- Isso configura **sobrecarga de métodos**

Sobrecarga de métodos

- Exemplos:

```
public void defineCores(String camiseta) {  
    this.setCamiseta(camiseta);  
}
```

```
public void defineCores(int camiseta, int calcao) {  
    this.setCamiseta(camiseta);  
    this.setCalcao(calcao);  
}
```

```
public void defineCores(int camiseta, int calcao, int meias) {  
    this.setCamiseta(camiseta);  
    this.setCalcao(calcao);  
    this.setMeias(meias);  
}
```

Sobrecarga de métodos


- Exemplos:

```
public void defineCores(String camiseta) {  
    this.setCamiseta(camiseta);  
}
```

Mesmo nome, listas
de parâmetros diferentes


```
public void defineCores(int camiseta, int calcao) {  
    this.setCamiseta(camiseta);  
    this.setCalcao(calcao);  
}
```

```
public void defineCores(int camiseta, int calcao, int meias) {  
    this.setCamiseta(camiseta);  
    this.setCalcao(calcao);  
    this.setMeias(meias);  
}
```



Sobrecarga de métodos

- Dependendo dos parâmetros passados, será chamada uma implementação específica do método
- Ou seja: chamada diferente, comportamento diferente



Sobrecarga de métodos

- O que faz a diferença entre os métodos é a quantidade de parâmetro e seus tipos, e não o nome das variáveis
- Não há limite para o número de métodos sobrecarregados em uma classe

Sobrecarga de métodos

- `public float soma(int a, int b) {...}` ✓
- `public float soma(float a, int b) {...}` ✓
- `public float soma(int a, float b) {...}` ✓
- `public float soma(float a, float b) {...}` ✓
- `public float soma(String a, float b) {...}` ✓
- `public float soma(String x, float z) {...}` ✗

Sobrecarga de métodos

- `public float soma(int a, int b) {...}` ✓
- `public float soma(float a, int b) {...}` ✓
- `public float soma(int a, float b) {...}` ✓
- `public float soma(float a, float b) {...}` ✓
- `public float soma(String a, float b) {...}` ✓
- `public float soma(String x, float z) {...}` ✗

Não pode pois **só muda o nome** dos parâmetros!



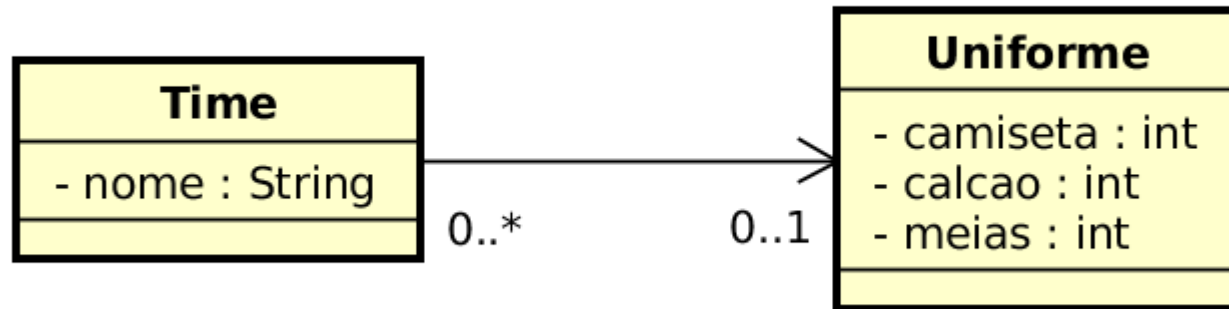
Roteiro

- Sobrecarga de métodos
- Passagem de Objetos por parâmetros
- Chamada de construtores dentro da chamada do método

Passagem de Objetos por Parâmetro

- Objetos podem ser aninhados. Um objeto pode possuir outro objeto dentro de si
- Essa “montagem” é muito comum!

Objetos Aninhados



- O diagrama acima pode ser lido como :
 - Um time possui nenhum ou um uniforme
 - Um uniforme pertence a nenhum ou vários times
- “Dentro do objeto time tem um objeto uniforme”

Objetos Aninhados

Time

nome
uniforme

camiseta
calcao
meias



Passagem de Objetos

- A maneira de definir e acessar os objetos aninhados é através de métodos.
- No nosso exemplo, para definirmos o uniforme de um time, precisamos passar o uniforme por parâmetro para ele

Passagem de Objetos

```
Uniforme un = new Uniforme();
```

```
un.setCamiseta(3);
```

```
un.setCalcao(1);
```

```
un.setMeias(2);
```

Primeiro cria-se
o objeto **un**,
da classe
Uniforme,
E preenche-se
seus dados

```
Time t1 = new Time();
```

```
t1.setNome("Tricolor");
```

```
t1.setUniforme(un);
```

Passagem de Objetos

```
Uniforme un = new Uniforme();
```

```
un.setCamiseta(3);  
un.setCalcao(1);  
un.setMeias(2);
```

```
Time t1 = new Time();
```

```
t1.setNome("Tricolor");  
t1.setUniforme(un);
```

Depois, o objeto **un**,
da classe **Uniforme**
é passado por
parâmetro para o objeto **t1**,
da classe **Time**

Passagem de Objetos

- Declaração do método na classe Time:

```
public void setUniforme(Uniforme uniforme) {  
    this.uniforme = uniforme;  
}
```

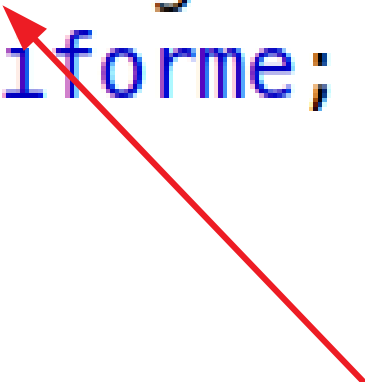
A declaração é igual a dos outros Métodos.

O tipo do **parâmetro** é uma classe

Retorno de Objetos

- Métodos também podem retornar, (ou devolver) objetos quando chamados:

```
public Uniforme getUniforme() {  
    return uniforme;  
}
```



Nesse caso, o tipo do **retorno** é uma classe



E o null?

- A ausência de criação de um objeto leva a uma referência nula, ou *null*
- Em qualquer local onde se pode passar um objeto, pode-se passar null quando não se tem o objeto
- Isso pode gerar erro se o método “acredita” que o objeto virá preenchido.

null

- Invocar método em objeto nulo gera erro ***NullPointerException***

```
Time inter = new Time("Internacional");  
inter.setUniforme(null);
```

```
Uniforme uniformeInter = inter.getUniforme();  
System.out.println(uniformeInter.getCamiseta());
```

```
Exception in thread "main" java.lang.NullPointerException  
    at TestaTimes.main(TestaTimes.java:92)
```



Roteiro

- Sobrecarga de métodos
- Passagem de Objetos por parâmetros
- Chamada de construtores dentro da chamada do método

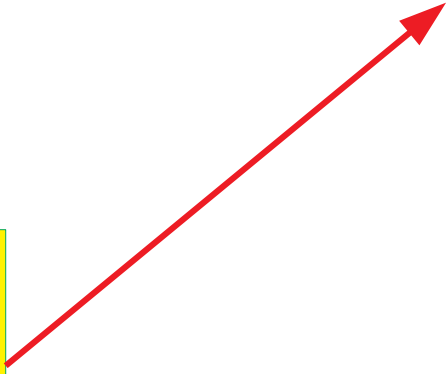
Chamando construtores dentro de métodos

- É possível chamar um construtor dentro de um método
- O retorno de um construtor é um objeto da classe que ele constrói!

Chamando construtores dentro de métodos

```
Time caxias = new Time("Caxias");  
caxias.setUniforme(new Uniforme());
```

O objeto Uniforme está sendo criado na hora



Chamando construtores dentro de construtores

- O mesmo pode ser feito no momento da construção dos objetos:

```
Time juventude = new Time("Juventude", new Uniforme());
```

Definição dos construtores

```
public Uniforme() {  
    this.camiseta = 0;  
    this.calcao = 0;  
    this.meias = 0;  
}
```

Percebam que os **construtores**
também podem ser
sobrecarregados

```
public Uniforme(int camiseta, int calcao, int meias) {  
    this.camiseta = camiseta;  
    this.calcao = calcao;  
    this.meias = meias;  
}
```

Definição dos construtores

```
public Time() {  
}
```

```
public Time(String nome) {  
    this.nome = nome;  
}
```

```
public Time(String nome, Uniforme uniforme) {  
    this.nome = nome;  
    this.uniforme = uniforme;  
}
```

sobrecarga





Dúvidas?





Atividades

- Execute as atividades presentes no documento

05.Lista.de.Exercícios.POO.pdf

Próximos passos

- Strings

