

Programação Orientada a Objetos

Professor: Anderson Elias



Estácio

1

Sobrescrita e Sobrecarga de Métodos

Contato:
anderson.elias@estacio.br



Estácio

2

*"Quando penso que cheguei ao meu limite,
descubro que tenho forças para ir além."*
Ayrton Senna

Contato:
anderson.elias@estacio.br



Estácio

3

Sobrescrita / Sobercarga (Overriding / Overloading)

Sobrescrever ou Sobrecarregar:

É quando a Subclasse pode redefinir o método da Superclasse com uma implementação mais adequada para ela.

Podendo fazer uma **Sobrescrita**: (**@OVERRIDE**)

Ou

Podendo fazer uma **Sobrecarga**: (**@OVERLOAD**)

4

Sobrescrita / Sobercarga (Overriding / Overloading)

1. Métodos sobrescritos

Sempre que herdamos um método da superclasse, podemos sobrescrevê-lo.

Fazendo isto temos a oportunidade de determinar um comportamento específico do método para a subclasse.

ATENÇÃO

Métodos marcados como *final*, *private* ou *static* não são herdados e por este motivo não podem ser sobrescritos.

5

Sobrescrita / Sobercarga (Overriding / Overloading)

```
//Superclasse
public class Animal {
    void comer() {
        System.out.println("Animal comendo.");
    }
}
```

```
//Subclasse
public class Cachorro extends Animal {
    void rolar() {
        System.out.println("Cachorro rolando.");
    }
    //Cachorro sobescreveu o método comer.
    void comer() {
        System.out.println("Cachorro comendo.");
    }
}
```

6

Sobrescrita / Sobercarga (Overriding / Overloading)

```
public class Teste {
    public static void main(String[] args) {
        new Cachorro().comer();
    }
}
```

A classe **Cachorro** herdou o método **comer** da classe **Animal** mas por algum motivo teve que escrever seu próprio método comer. Então se num determinado momento quisermos chamar:

```
new Cachorro().comer();
```

Resultado:

Cachorro comendo.

7

Sobrescrita / Sobercarga (Overriding / Overloading)

Quando convertemos pra cima da árvore de herança, os métodos do subtipo são perdidos mas neste caso a versão executada será a do subtipo.

Quando sobrescrevemos um método e depois fazemos o **upcast**:

```
Animal animal = new Cachorro();
animal.comer();
```



Cachorro comendo.

O compilador só verifica o tipo da variável e não o da instância. **Somente métodos de Animal poderão ser executados**. Mas em tempo de execução, o objeto real que está sendo executado é **Cachorro**, então o método de **Cachorro** será chamado.

8

Sobrescrita / Sobercarga (Overriding / Overloading)

2. Métodos sobrecarregados

Métodos sobrecarregados tem uma diferença básica dos métodos sobrescritos: **Sua lista de argumentos DEVE ser alterada.**

Métodos sobrecarregados permitem que se utilize o mesmo nome do método numa mesma classe ou subclasse, dando mais opções a quem for chamar o método.

9

Sobrescrita / Sobercarga (Overriding / Overloading)

2. Métodos sobrecarregados **na mesma classe.**

```
//Superclasse
public class Animal {
    void comer() {
        System.out.println("Animal comendo.");
    }
    //Sobrecarregado na mesma classe.
    void comer(int x, String s) {
        System.out.println("Animal comendo 2...");
    }
}
```

10

Sobrescrita / Sobrecarga (Overriding / Overloading)

2. Métodos sobrecarregados **na mesma classe**.

```
public class AreaComSobrecarga {
    public static void main(String[] args) {
        System.out.println("Área em quadrado: " + cauculaArea(3) );
        System.out.println("Área em Retangulo: " + cauculaArea(3, 2) );
        System.out.println("Área em Cubo: " + cauculaArea(3, 2, 5) );
    }
    public static double cauculaArea(int x) {
        return x * x;
    }
    public static double cauculaArea(int x, int y) {
        return x * y;
    }
    public static double cauculaArea(int x, int y, int z) {
        return x * y * z;
    }
}
```

11

Sobrescrita / Sobrecarga (Overriding / Overloading)

Os métodos sobrecarregados podem ser declarados na mesma classe ou **na subclasse**.

```
public class Cachorro extends Animal {
    void rolar() {
        System.out.println("Cachorro rolando.");
    }
    //Cachorro sobescreveu o método comer.
    void comer() {
        System.out.println("Cachorro comendo.");
    }
    //Sobrecarregou o método da superclasse.
    void comer(int x, float s) {
        System.out.println("Cachorro comendo 2...");
    }
}
```

Observe que está sobrecarregando pois houve mudança na lista de argumentos.

Para saber qual método será chamado, vai depender exclusivamente dos argumentos passados.

12

Sobrescrita / Sobrecarga (Overriding / Overloading)

```
public class Teste {
    public static void main(String[] args) {
        new Cachorro().comer();

        Animal animal = new Cachorro();
        animal.comer();

        Cachorro cachorro = new Cachorro();
        cachorro.comer(1, 9.5f);
    }
}
```

Vamos testar a sobrecarga com o uso do **polimorfismo**:

- Mude os parâmetros do método comer passando o que segue:

`cachorro.comer(1, "Mike");`

- Observe que o método faz chamada agora para a **superclasse**.

Cachorro comendo.

Animal comendo.

13

Sobrescrita / Sobrecarga (Overriding / Overloading)

E se os argumentos forem objetos ao invés de tipos primitivos?

```
public class A {
    void teste(A a) {
        System.out.println("Classe A");
    }
}
```

A saída será normal, executou o método de acordo com o tipo passado:

Classe A
Classe B
Classe A

```
public class B extends A {
    void teste(B b) {
        System.out.println("Classe B");
    }
    public static void main(String[] args) {
        //Chamou passando um objeto A
        new A().teste(new A());
        //Chamou passando um objeto B
        new B().teste(new B());
        //Chamou passando um objeto A
        new B().teste(new A());
    }
}
```

14

Sobrescrita / Sobercarga (Overriding / Overloading)

Mas e se a referência usada fosse **polimórfica**:

```
A a = new B();
a.teste(a);
```

A saída seria **A** pois mesmo sabendo que em tempo de execução o objeto utilizado será um **B** e não um **A**, quando o método é sobrecarregado isto não é decidido em tempo de execução.

Quando o método é sobrecarregado, apenas o tipo da referência importa para saber qual método será chamado.

15

Sobrescrita / Sobercarga (Overriding / Overloading)

Atenção **aos métodos que parecem ser sobrescritos** mas estão sendo **sobrecarregados**:

```
public class C {
    void teste() {
        System.out.println("Classe C");
    }
}
```

```
public class D extends C{
    void teste(String s) {
        System.out.println("Classe D");
    }
}
```

A classe **D** tem dois métodos:

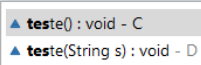
A versão teste sem argumentos que herdou de **C** e a versão sobrecarregada.

16

Sobrescrita / Sobrecarga (Overriding / Overloading)

Testando a sobrecarga das classes C e D:

```
public class TesteCD {
    public static void main(String[] args) {
        D d = new D();
        d.teste()
    }
}
```



Um código que faz referência à classe **C** pode chamar somente teste sem argumentos mas uma referência a **D** pode chamar qualquer um dos dois métodos, tanto da subclasse quanto da superclasse.

17

Sobrescrita / Sobrecarga (Overriding / Overloading)

Se você herdou métodos abstratos de uma classe abstrata, então é obrigatório sobrescrevê-los.

Mas se sua classe também for abstrata não precisa, você pode simplesmente "*passar a vez*" para a última classe concreta da árvore.

Porém em algum momento você será obrigado a sobrescrever os métodos *abstract*.

18

Dúvidas?



Estácio

