## **O que é Sobrecarga de métodos?**

Quando estudamos o conceito de [polimorfismo](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=polimorfismo), aprendemos que podem existir métodos com o mesmo nome e que são especializados em subclasses, ou seja, métodos que são herdados de uma superclasse e sobrescritos de forma mais adaptada na subclasse. Agora iremos conferir que é possível haver vários métodos com o mesmo nome e dentro de uma mesma classe através do conceito de Sobrecarga.

Sobrecarregar métodos é uma capacidade da Linguagem Java que não está presente em todas as linguagens de programação e consiste simplesmente na ação de criar duas ou mais implementações de métodos com o mesmo nome dentro de uma mesma classe, bastando, para isso, apenas que eles tenham assinaturas únicas (diferentes entre si).

## **Assinatura de um método**

A assinatura de um método é o conjunto de informações que o compilador utiliza para identificar o método dentro de uma classe. Isso é o que o torna único naquela classe. O modelo a seguir representa a assinatura de um método:

nome\_do\_método([lista\_de\_parâmetros])

Ao mencionarmos assinatura de métodos que recebem parâmetros, podemos nos referenciar a eles apenas pelo seu tipo de dado, ignorando seus nomes. Observe, a seguir, um exemplo de método que recebe parâmetros e a sua assinatura.

public String valorPorExtenso(int valor){ //... }

Assinatura: valorPorExtenso(int)

A partir do exemplo anterior, podemos afirmar que não é possível manter, dentro de uma mesma classe, dois métodos chamados valorPorExtenso que recebam apenas um parâmetro do tipo int. Existem duas partes do método que não são relevantes para a sua assinatura, ou seja, que não são levadas em consideração pelo compilador no momento de tentar diferenciar um método de outro. São elas:

* + Modificador de acesso/visibilidade;
  + Tipo de retorno.

Para entender melhor como funciona a análise de assinaturas de métodos, observe o quadro da próxima tela, que mostra pares de métodos. Nesse quadro, perceberemos se eles podem ser considerados sobrecarregados ou não. Para o caso em que eles são considerados sobrecarregados, ou seja, quando possuem a mesma assinatura, ocorrerá um erro se tentarmos compilar uma classe contendo o par de métodos em questão.

| **Métodos** | **São sobrecarregados?** |
| --- | --- |
| public double valorDoSalario() e  private int valorDoSalario() | Não, pois modificadores de acesso e  tipo de retorno não servem para diferenciação de métodos. Ambos têm assinatura valorDoSalario(). |
| public double valorDoSalario(double comissão) e  public double valorDoSalario() | Sim. O primeiro método recebe um  parâmetro do tipo double e o segundo não recebe parâmetros. Suas  assinaturas são valorDoSalario(double)  e valorDoSalario(),  respectivamente. |
| public void imprimir(int a) e  public void imprimir(int b) | Não, pois os nomes dos parâmetros  não são suficientes para diferenciar os métodos. Ambos  têm assinatura imprimir(int). |
| void imprimir(int a, String b) e  void imprimir(String b, int a) | Sim, pois, embora possuam os mesmos  tipos de parâmetros, a lista desses parâmetros está em uma sequência diferente  em cada método. Suas assinaturas são imprimir(int,  String)  e imprimir(String,  int  ), respectivamente. |

## **Diferença de Sobrescrita e Sobrecarga**

É importante compreender que Sobrescrita e Sobrecarga de métodos são duas coisas diferentes. Quando trabalhamos com herança, podemos sobrescrever métodos de superclasses em subclasses, mas apenas se mantivermos exatamente a sua assinatura. Observe, no próximo exemplo, que sobrescrevemos o método valorDoSalario() em FuncionarioComissionado. Isso só é possível porque mantivemos sua assinatura, ou seja, seu nome e a lista de parâmetros que, no caso, é vazia.

Já a Sobrecarga ocorre com métodos definidos na própria classe, como podemos constatar na classe FuncionarioComissionado, que, além de sobrescrever o método double valorDoSalario(), que foi herdado da classe Funcionario, também realiza uma Sobrecarga desse método através da definição do método double valorDoSalario(int gratificacao). Perceba que eles têm o mesmo nome, mas se diferenciam em relação aos parâmetros.

public class Funcionario {

private double salario;

public double valorDoSalario(){

return salario;

}

}

public class FuncionarioComissionado extends Funcionario {

private double comissaoPorPeca;

private int quantidadePecasVendidas;

public double valorDoSalario(){

return super.valorDoSalario() + comissaoPorPeca \* quantidadePecasVendidas;

}

}

Chegamos ao fim do nosso primeiro tópico. Nele, entendemos o que é a Sobrecarga de métodos e percebemos que ela está intimamente ligada às assinaturas dos métodos. Entendemos também que Sobrecarga e Sobrescrita de métodos são coisas diferentes. A seguir, conheceremos diversos exemplos de Sobrecarga para assimilar melhor seu conceito.

## **Exemplos de Sobrecarga de métodos**

Vamos conferir alguns exemplos de Sobrecarga de métodos. Imagine uma classe na qual queiramos criar um método para calcular o triplo de um número. Temos que definir que tipo de número esse método vai receber como parâmetro. Se precisarmos calcular o triplo de inteiros e também de números do tipo double, a solução seria fazer dois métodos sobrecarregados, chamados triplo. Em Java, poderíamos fazer da seguinte maneira:

public int triplo(int valor) {

return (3\*valor);

}

public double triplo(double valor) {

return (3\*valor);

}   
  
 Da forma como acabou de ser apresentado, teríamos dois métodos que têm a mesma função (calcular o triplo de um valor numérico) e que possuem o mesmo nome (triplo). Isso facilita a invocação do método, pois você não precisa lembrar qual método trabalha com int e qual trabalha com double, basta simplesmente invocá-lo pelo nome triplo, passar o parâmetro e deixar o compilador descobrir de qual método estamos precisando.

Vamos conhecer mais um exemplo. Uma funcionalidade muito comum, enquanto estamos desenvolvendo programas, é a ordenação de dados, ou seja, colocar certa quantidade de dados na ordem crescente ou decrescente, dependendo do que iremos necessitar. Às vezes precisamos apenas descobrir qual é o maior ou o menor valor dentro de uma lista de dados, como um array. Digamos que precisamos de um método para descobrir qual valor é o maior de uma lista de valores. Podemos criar um método maiorValor para fazer isso, como apresentado a seguir.  
 1ª Sobrecarga:

public int maiorValor(int v1, int v2) {

if(v1 > v2) return (v1);

else return (v2);

}

2ª Sobrecarga:

public int maiorValor(int v1, int v2, int v3) {

int maiorTemp = v1;

if(v2 > maiorTemp) maiorTemp = v2;

if(v3 > maiorTemp) maiorTemp = v3;

return (maiorTemp);

}  
  
 Nesse exemplo, podemos observar três sobrecargas para o método maiorValor. Na primeira, o método recebe dois inteiros e devolve o valor do maior deles. Na segunda, o método trabalha com três valores inteiros e, por isso, utiliza uma variável auxiliar para manter o valor do maior.

## **A importância da Sobrecarga de métodos**

Ao aplicarmos [polimorfismo](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=polimorfismo), podemos tornar nosso código mais genérico, uma vez que ganhamos a possibilidade de invocar métodos de subclasses a partir de referências da superclasse. E qual seria a vantagem de possuir vários métodos com o mesmo nome na mesma classe, ou seja, sobrecarregar métodos? A vantagem está em podermos criar diferentes métodos que executam ações semelhantes e que possuem o mesmo nome.

Aplicando o conceito de Sobrecarga, os nomes dos nossos métodos passam a representar uma ação genérica, então deixamos para o compilador o trabalho de escolher qual implementação deve ser usada. Perceba que isso simplifica o desenvolvimento de programas de duas formas diferentes: evita que precisemos definir métodos semelhantes com nomes diferentes e facilita a invocação, já que temos apenas um nome para representar todas as sobrecargas.

Algumas linguagens de programação, como a C, não possuem o recurso da Sobrecarga. Nessas linguagens, somos obrigados a definir métodos com nomes diferentes, mesmo que eles tenham um comportamento muito parecido. Se quiséssemos criar um método para calcular o dobro de um valor int e o dobro de um número double em C, teríamos de declará-los como nomes diferentes, por exemplo, dobro(int) e dobroDouble(double).

E como o compilador sabe qual a versão do método que estamos precisando? Ele utiliza a lista de parâmetros que fornecemos na invocação do método para identificar qual das sobrecargas deve ser utilizada. Daí a necessidade de cada método apresentar uma assinatura única. Observe, a seguir, a simulação de como o compilador Java encontraria a referência certa de cada método, baseado nos parâmetros passados no momento da invocação.

public static int dobro(int a) { → dobro(int)

return 2\*a;

}

public static double dobro(double a) { → dobro(double)

return 2\*a;

}

public static void main(String[] args){

int i = 5; double d = 5;

System.out.println(dobro(i)); → Checagem de Assinatura

System.out.println(dobro(d));

}

## **Sobrecarga de métodos construtores**

Assim como sobrecarregamos métodos comuns, podemos sobrecarregar métodos construtores. De fato, esta é uma das principais aplicações de Sobrecarga. Sobrecarregar métodos construtores nos dá flexibilidade para instanciar um objeto da forma mais adequada ou de uma forma mais simples, quando não temos todos os dados necessários no momento. Vamos conferir, na próxima tela, um exemplo para a classe Quadrado. Ela representa um Quadrado para o qual podemos definir sua posição através das coordenadas (x,y) e o tamanho do seu lado.

A classe Quadrado possui quatro métodos construtores sobrecarregados. O primeiro, construtor padrão, não recebe nenhum parâmetro. Para esse caso, os valores de x, y e lado serão inicializados com 0 (zero), pois este é um comportamento padrão da Linguagem Java para inicialização de valores relativos a variáveis numéricas. No segundo construtor, podemos informar os valores de x e y. No terceiro, podemos informar as coordenadas x, y e lado do quadrado. Por fim, temos o quarto construtor, que recebe um objeto do tipo Quadrado como parâmetro e o utiliza como base para o novo quadrado que está sendo construído, copiando todos os seus valores de coordenadas x, y e lado. O resultado do uso do quarto construtor é um objeto idêntico ao objeto definido como parâmetro. Perceba que, nos três últimos construtores, usamos a palavra-chave this para fazer referência às variáveis declaradas no escopo da classe. A seguir, um exemplo de uso destes quatro construtores.

public class Quadrado {

private int x, y, lado;

public Quadrado(){

}

public Quadrado(int x, int y){

this.x = x;

this.y = y;

}

public Quadrado(int x, int y, int lado){

this.x = x;

this.y = y;

this.lado = lado

}

public Quadrado(Quadrado q){

this.x = q.x;

this.y = q.y;

this.lado = q.lado;

}

}

01 public class TesteQ {

02

03

04 public static void main(String[] args){

05 Quadrado q1 = new Quadrado();

06 Quadrado q2 = new Quadrado(10, 15);

07 Quadrado q3 = new Quadrado(20, 30, 50);

08 Quadrado q4 = new Quadrado(q3);

09 }

10 }

11   
  
 No método principal da classe TesteQ, instanciamos quatro objetos do tipo Quadrado. O primeiro, q1, tem valores de x, y e lado iguais a zero. O objeto q2 tem coordenadas (x,y) iguais a 10 e 15, respectivamente, e lado igual a zero. No objeto q3, definimos coordenadas (x, y) iguais a 20 e 30, respectivamente, e lado de tamanho 50. O objeto q4 terá exatamente as mesmas características de q3.