

## Apresentação

A abordagem principal na programação orientada a objetos (POO) diz respeito ao *design* de um sistema e seus componentes, que incluem entidades, objetos e relacionamentos. Diferentes métodos são utilizados para determinar o comportamento dos objetos, abrangendo aspectos como estado, funções e procedimentos da programação estruturada.

Um dos métodos utilizados na POO é a sobrescrita, que está diretamente ligada à herança. Esse método envolve a existência de um método na classe-mãe, que é herdado pela classe-filha, mas implementado de forma diferente, mantendo a mesma assinatura. Sobrescrever um método significa substituir a implementação da superclasse daquele método com a sua própria, mantendo a mesma assinatura, mas alterando o comportamento. O tipo de retorno pode variar de maneira particular.

Nesta Unidade de Aprendizagem, você vai explorar as características da sobrescrita e da assinatura de método, bem como identificar a herança e compreender conceitos básicos sobre a construção de um aplicativo. Com isso, é possível construir um aplicativo com uma estrutura de herança fazendo uso da sobrescrita de métodos.

Bons estudos.

**Ao final desta Unidade de Aprendizagem, você deve apresentar os seguintes aprendizados:**

- Definir o que é sobrescrita e assinatura de método.
- Identificar herança.
- Construir um aplicativo com uma estrutura de herança fazendo uso de sobrescrita de métodos.

# Desafio

No paradigma da POO, a aplicação de cálculos deve ser implementada com base em boas práticas, considerando os pilares da POO, entre eles a herança e a sobrescrita. Existem diferentes maneiras de implementar o cálculo de figuras geométricas. A herança permite a reutilização do código e define métodos abstratos da classe-pai, já a sobrescrita de método permite realizar o cálculo correto da área de cada figura.

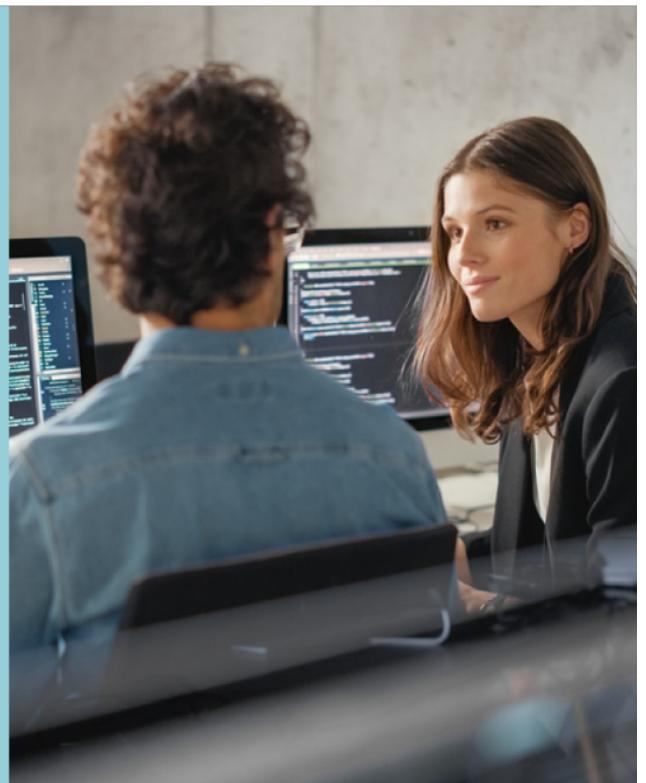
Para implementar o cálculo com base em uma figura geométrica, é necessário utilizar a forma adequada e saber o tipo de figura. O cálculo da área também exige a definição de métodos abstratos na classe abstrata e a definição de herança para efetivar métodos.

Analise a seguinte situação:

Você é analista/programador em uma fábrica de *software* e foi encarregado de criar uma aplicação para realizar o cálculo da área de figuras geométricas, especificamente retângulo e triângulo retângulo.

Sua tarefa é criar as classes-modelo utilizando técnicas de herança e sobrescrita de método e uma classe para executar a aplicação.

Para construir o programa, o gestor solicitou que você utilize uma linguagem de programação orientada a objeto com um IDE para auxiliar na organização do projeto e recomendou que a linguagem implementada seja Java com IDE NetBeans.



Para comprovar a entrega do projeto, considere os seguintes entregáveis:

- a) Salve o projeto com o nome “CalculoArea” e encaminhe o arquivo zipado como anexo na plataforma.
- b) Descreva brevemente os passos que foram seguidos para a construção do programa. Fale sobre estrutura do aplicativo, superclasses, métodos, classes de chamadas e valores inseridos.

# Infográfico

A sobrescrita é uma característica permanente da linguagem orientada a objetos. Por isso, quando a versão de um método é executada, a ação será determinada pelo objeto usado para invocá-la. Se um objeto de uma classe-pai é usado para chamar o método, a versão na classe-pai é executada; no entanto, se um objeto da subclasse é utilizado para chamar o método, a versão na classe-filha é executada.

Algumas linguagens deixam que os programadores escolham entre utilizar e evitar que métodos sejam sobrepostos. A principal propriedade da substituição de método é que a classe pode prover sua própria implementação específica para um método herdado sem modificar o código da classe-pai.

Veja, neste Infográfico, os conceitos de sobrescrita.

## Conceitos importantes sobreescrita

A sobreescrita de métodos permite criar métodos na classe-filha com base na mesma assinatura e tipo de retorno do método sobreescrito. Existem conceitos importantes que devem ser considerados para garantir a implementação correta.

Conheça cada um desses conceitos.

### A sobreescrita de métodos

#### HERANÇA



- Permite a criação de classes com base nas características de uma classe existente.
- A classe existente é denominada "superclasse".
- A classe criada a partir dela é a subclasse.
- Ocorre o reúso de software.

#### SOBRESCRITA E ASSINATURA

##### O que é assinatura?

- Garante que os métodos vinculados entre si tenham o mesmo nome, quantidade e tipos de parâmetros.

##### Outras características da sobreescrita (override) em Java

- Altera o comportamento das subclasses por um mais específico.
- Quando um método abstrato herda as características de uma classe abstrata, estamos praticando a sobreescrita.



### Exemplo

#### O conceito de sobreescrita (override) considera algumas regras.

- Que tal aplicar a um cenário real para esclarecer sua funcionalidade?



A onça é um mamífero carnívoro que tem comportamento em comum com outros animais, como a atividade da caça. Animais de outras espécies, como os répteis, também praticam a caça. Porém, outras atividades exclusivas também caracterizam o seu comportamento.

**Na sobreescrita, os dados seriam definidos assim:**

```
Animal: interface  
Método: cacar ()  
Classe: mamifero_carnivoro
```

**O código implementado seria:**

```
package br.com.exemplojm.animal;  
  
public interface Animal {  
  
    public String cacar();  
  
}
```

E o diagrama seria construído da seguinte forma:



A sobreescrita é um método muito utilizado no cotidiano de programadores e desenvolvedores que utilizam a linguagem de programação Java. Ter conhecimento da abordagem de sobreescrita e de suas regras é essencial para tornar-se um bom profissional.



Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

# Conteúdo do Livro

---

A programação orientada a objetos é um paradigma de programação que visa a organizar o código em torno de objetos e suas interações. A ideia central é modelar o mundo real por meio de classes, abstrações que representam entidades do mundo real. Com isso, é possível criar programas mais modularizados e de fácil manutenção.

Um dos recursos mais importantes da programação orientada a objetos é a herança, que permite criar classes a partir de outras já existentes, herdando seus atributos e métodos. A sobrescrita de métodos é um recurso da herança que permite a um objeto da subclasse (classe-filha) alterar a implementação de um método herdado da superclasse (classe-pai). Com isso, podem ser criados programas mais flexíveis e adaptáveis, capazes de lidar com diferentes situações de forma eficiente.

No capítulo **Sobrescrita**, você vai ver os conceitos de sobrecarga de métodos e sobrescrita de métodos e a diferença entre eles. Também vai estudar a implementação de herança em Java e, por fim, como construir uma aplicação com uma estrutura de herança usando a sobrescrita de métodos. Esses recursos são amplamente utilizados na programação orientada a objetos. Dominando-os, você poderá escrever códigos mais eficientes e organizados, reduzindo as redundâncias e melhorando a manutenção e a evolução do seu *software*.

Boa leitura.

# PROGRAMAÇÃO ORIENTADA A OBJETOS



sagah<sup>+</sup>

# Sobrescrita

*Juliana Padilha*

## OBJETIVOS DE APRENDIZAGEM

- > Definir o que é sobrescrita e assinatura de método.
- > Identificar herança.
- > Construir um aplicativo com uma estrutura de herança fazendo uso de sobrescrita de métodos.

## Introdução

A programação orientada a objetos é um paradigma importante e flexível que permite aos desenvolvedores criar aplicações escaláveis e modulares. Uma das principais características da programação orientada a objetos é a sua capacidade de abstração, que possibilita modelar objetos do mundo real em código de forma mais precisa e intuitiva. Dentro desse paradigma, a sobrescrita e a assinatura de método são conceitos fundamentais e avançados, para personalização de classes e objetos de acordo com as necessidades específicas de uma aplicação.

Neste capítulo, você vai estudar os conceitos de sobrescrita e assinatura de método e como podem ser aplicados no mundo real, além de ver como identificar a implementação da herança na programação orientada a objetos. Você vai também construir uma aplicação utilizando a estrutura de herança e a sobrescrita de métodos, para estudar o tema de forma prática. Esse conhecimento é importante para desenvolver aplicações mais flexíveis e personalizadas, bem como criar hierarquias de classes com comportamentos distintos e adequados às necessidades específicas da sua aplicação.

## Sobrescrita e sobrecarga de métodos

Programação orientada a objetos (POO) é um paradigma que se baseia em classes e objetos que têm propriedades (atributos) e comportamentos definidos por métodos. Em Java, existem duas técnicas para trabalhar com métodos de mesmo nome, mas implementações diferentes: a sobrescrita e a sobrecarga de métodos. A sobrescrita de métodos (*override*, em inglês) é um recurso fundamental da orientação a objetos e diretamente relacionado à herança. Com a sobrescrita de métodos, é possível personalizar e modificar o comportamento do método herdado de uma superclasse na subclasse, tornando-o mais adequado e personalizado para as necessidades específicas da subclasse (DEITEL; DEITEL, 2008).

A sobrescrita ocorre quando uma subclasse implementa um método com o mesmo nome, tipo de retorno e lista de parâmetros de uma classe-pai. Quando o método é chamado em um objeto da subclasse, a implementação da subclasse é executada, ignorando a implementação da classe-pai. Usamos essa técnica, principalmente, para modificar ou estender o comportamento de um método existente em uma classe-pai, personalizando, assim, o comportamento da classe conforme as necessidades do programa.

Já com a sobrecarga de métodos (*overload*, em inglês), é possível definir vários métodos com o mesmo nome, desde que eles tenham diferentes combinações de parâmetros, variando em quantidade, tipo e/ou ordem. Quando um método é sobre carregado, o compilador é acionado para selecionar o método apropriado com base na análise dos argumentos passados na chamada, considerando número, tipos e ordem dos parâmetros. Usamos a sobrecarga de métodos, portanto, para criar vários métodos com o mesmo nome, que realizam tarefas semelhantes, mas operam em tipos de dados diferentes (DEITEL; DEITEL, 2008).



### Fique atento

Ao sobre carregar métodos que executam tarefas relacionadas, é possível aumentar a legibilidade e a compreensibilidade dos programas (DEITEL; DEITEL, 2008).

Os métodos sobreescritos são identificados por sua assinatura, que é uma combinação do nome do método e seus tipos de parâmetros. Isto é, a assinatura de método se refere ao conjunto de parâmetros que um método recebe e ao tipo de retorno que ele produz. Como essa assinatura é única, dois métodos com o mesmo nome, mas com assinaturas diferentes são considerados distintos em POO. Isso é importante para a sobreescrita de método, que permite a uma classe ter vários métodos com o mesmo nome e diferentes assinaturas (DEITEL; DEITEL, 2008).

Essa assinatura é essencial para ajudar o compilador a compreender corretamente os métodos de um programa. Durante a compilação, se o compilador considerasse apenas os nomes dos métodos, poderia haver ambiguidade, como no caso em que há dois métodos com o mesmo nome. Por exemplo, considere que há dois métodos denominados `quadrado()`. Para resolver, o compilador utiliza nomes decorados com mais dados, que incluem o nome original do método, o tipo de cada parâmetro e a ordem exata dos parâmetros, para determinar se os métodos em uma classe são únicos ou não. O Exemplo 1 ilustra o conceito de assinatura por meio da implementação do método `quadrado()`.

## Exemplo 1

```
public int quadrado(int x) {
    return x * x;
}

public double quadrado(double x) {
    return x * x;
}
```

No Exemplo 1, é possível observar que existem duas versões do método `quadrado()`: uma que recebe um inteiro (`int`) como parâmetro e retorna um inteiro, e outra que recebe um número em ponto flutuante (`double`) como parâmetro e retorna um número em ponto flutuante. Dessa forma, quando utilizamos o método `quadrado()` em um objeto da classe `Calculadora`, por exemplo, o compilador é capaz de distinguir qual versão do método deve ser utilizada com base no tipo de parâmetro que é passado para ele.

## Implementação do conceito de sobrecarga de método

A seguir, será apresentado um exemplo prático de como implementar o conceito de sobrecarga de métodos em uma aplicação real. Com esse exemplo, será possível compreender como essa técnica pode tornar o código mais adaptável e versátil, pois permite que os métodos sejam utilizados com diferentes tipos de parâmetros, facilitando o reuso de código em diferentes partes da aplicação.

Considere o código exibido no Exemplo 2. A classe `Calculadora` tem três métodos com o mesmo nome: `somar()`. No entanto, esses métodos têm diferentes assinaturas. Por isso, foi utilizado o conceito de sobrecarga de método, que permite que um método tenha o mesmo nome, mas diferentes parâmetros, retornos ou tipos de exceção.

### Exemplo 2

```
class Calculadora {  
    public int somar(int a, int b) {  
        return a + b;  
    }  
  
    public double somar(double a, double b) {  
        return a + b;  
    }  
  
    public int somar(int a, int b, int c) {  
        return a + b + c;  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {  
        Calculadora calc = new Calculadora();  
  
        int resultadoInt1 = calc.somar(3, 4);  
        double resultadoDouble = calc.somar(3.5d, 4.5d);  
        int resultadoInt2 = calc.somar(3, 4, 5);  
    }  
}
```

```
System.out.println("Resultado da soma de 2 números inteiros: " + resultadoInt1);
System.out.println("Resultado da soma de 3 números ponto flutuante: " + resultadoDouble);
System.out.println("Resultado da soma de 3 inteiros: " + resultadoInt2);
}
```

Observe que no código apresentado no Exemplo 2, o primeiro método `somar()` recebe dois argumentos do tipo `int` e retorna um valor do mesmo tipo, representando a soma desses dois números inteiros. Já o segundo método `somar()` recebe dois argumentos do tipo `double` e retorna um valor do mesmo tipo, representando a soma desses dois números de ponto flutuante. Por fim, o terceiro método `somar()` recebe três argumentos do tipo `int` e retorna um valor do mesmo tipo, representando a soma desses três números inteiros.

Desse modo, ao chamar `somar()` em um objeto `Calculadora`, o compilador escolherá o método a ser executado de acordo com o número e tipos dos argumentos passados na chamada. Por exemplo, se a chamada for `calculadora.somar(3, 4)`, o primeiro método será executado, retornando o valor 7. Se a chamada for `calculadora.somar(3.5, 4.5)`, o segundo método será executado, retornando o valor 8.0. E se a chamada for `calculadora.somar(3, 4, 5)`, o terceiro método será executado, retornando o valor 12.



### Fique atento

Quando tentamos criar um método numa classe que já tem outro método com a mesma assinatura (nome e parâmetros), há um erro de sintaxe, por se tratar de uma sobrecarga incorreta de método (DEITEL; DEITEL, 2008).

Em resumo, a sobrescrita e a sobrecarga são recursos fundamentais da POO para adaptar o comportamento dos métodos às necessidades específicas de cada classe. Como vimos, a sobrescrita permite modificar o comportamento de um método herdado da superclasse na subclasse, enquanto a sobrecarga permite criar métodos com o mesmo nome, mas com parâmetros diferentes. Se utilizadas de maneira adequada, a sobrescrita e a sobrecarga podem tornar o código mais organizado, eficiente e fácil de entender.

Na próxima seção, será abordado outro conceito fundamental da POO, a herança, que permite reutilizar o código de uma classe existente em uma nova classe, estendendo seu comportamento de acordo com as necessidades da nova classe.

## Herança em programação orientada a objetos

Na POO, a herança é um conceito crucial, que permite a criação de classes que compartilham características e comportamentos com outras classes. Possibilita a reutilização de código, fazendo com que novas classes possam ser criadas a partir de classes existentes, herdando seus atributos e comportamentos e expandindo-as com funcionalidades específicas que as novas classes demandam (DEITEL; DEITEL, 2008).

A capacidade de reutilização de código é uma grande vantagem do mecanismo de herança, pois otimiza o tempo de desenvolvimento de programas. Além disso, esse mecanismo encoraja a utilização de software previamente testado e depurado, reduzindo a ocorrência de problemas após a implementação de um sistema.

A classe existente é conhecida como “superclasse”, “classe-base” ou “classe-pai”, e a nova classe é chamada de “classe-subclasse”, “classe-derivada” ou “classe-filha”. A superclasse contém propriedades (atributos) e métodos que são compartilhados com suas subclasses. Por sua vez, as subclasses podem adicionar novas propriedades e métodos, bem como substituir ou sobrepor os métodos que já existem na superclasse. Essa descrição refere-se à **herança simples** (SEBESTA, 2018).

Vale ressaltar que, além da herança simples, também existe a **herança múltipla**. No entanto, a linguagem Java não suporta a herança múltipla, que é a capacidade de uma classe herdar diretamente atributos e métodos de duas ou mais classes diferentes. Isso porque, quando uma classe é derivada de duas superclasses que definem um método público com os mesmos nome e protocolo, a herança múltipla pode levar a conflitos de nomes de método. Essa situação, portanto, pode resultar em ambiguidade e dificultar a leitura e a manutenção do código (SEBESTA, 2018).

Para contornar esses problemas, então, a linguagem Java oferece outras maneiras de reuso de código, como a **implementação de interfaces**, com a

qual é possível obter uma forma limitada de herança múltipla. Por exemplo, considere que temos duas classes, Carro e Caminhão, com métodos semelhantes, como `acelerar()` e `frear()`. Podemos definir uma interface Veiculo, que declare esses métodos e, em seguida, fazer com que as classes Carro e Caminhão implementem essa interface. Isso permitirá que as classes herdem o comportamento dos métodos da interface, mas sem os conflitos de nomes que podem ocorrer com a herança múltipla. Além disso, os conflitos de nomes de variáveis são completamente evitados, uma vez que as interfaces não podem definir variáveis (ORACLE, 2022; SEBESTA, 2018).



### Saiba mais

Uma interface é uma coleção de métodos abstratos que uma classe pode implementar para definir seu comportamento, sendo possível compartilhá-lo. Como a classe herda comportamentos de várias interfaces, podemos usar uma interface para simular a herança múltipla (SEBESTA, 2018).

A seguir, você vai acompanhar um exemplo prático de implementação de herança na linguagem Java. Vai ver como as subclasses podem estender as funcionalidades da superclasse, além de como acessar e modificar as variáveis de instância e métodos herdados. Esse exemplo será útil para compreender a importância e a utilidade da herança na POO.

## Implementação do conceito de herança

Em Java, a herança pode ser identificada por meio da palavra-chave `extends` na declaração da subclasse. Essa palavra-chave indica que a classe está herdando as propriedades e os métodos de outra classe, especificada logo após a palavra-chave `extends`. Para facilitar o entendimento, considere o trecho de código apresentado no Exemplo 3, que tem o objetivo de exemplificar a sintaxe necessária à criação de uma herança na linguagem Java.

## Exemplo 3

```
public class NomeSubclasse extends NomeSuperClasse {
    //código da classe NomeSubClasse
}
```

Com base na sintaxe apresentada no Exemplo 3, foi implementado um código em Java para ilustrar um exemplo de herança. O código resultante é exibido no Exemplo 4 e inclui as classes `Animal`, `Cachorro`, `Gato` e `Ave`. A classe `Animal` é a superclasse, que contém as propriedades compartilhadas pelas subclasses `Cachorro`, `Gato` e `Ave`. Cada uma dessas subclasses herda da classe `Animal` e adiciona propriedades específicas dessas classes.

A classe `Cachorro` tem a propriedade de raça; a classe `Gato` tem a propriedade de cor da pelagem; e a classe `Ave` tem a propriedade de voar. Podemos usar essas classes para criar objetos que representam animais específicos. Por fim, a classe `Main()` cria instâncias das classes `Cachorro`, `Gato` e `Ave`, com parâmetros passados pelos construtores de cada classe. Em seguida, são exibidos os atributos de cada instância usando os métodos `getters` correspondentes. No caso, são impressos o nome, a idade e a raça do cachorro; o nome, a idade e a cor do gato; e, por fim, o nome, a idade e a espécie da ave.

É importante notar que uma subclass deve invocar um método construtor `public/protected` definido na sua superclasse, fornecendo os parâmetros necessários. Essa invocação é feita utilizando-se a palavra “super”, conforme exibido no Exemplo 4.

## Exemplo 4

```
class Animal {
    private String nome;
    private int idade;

    public Animal(String nome, int idade) {
        this.nome = nome;
        this.idade = idade;
    }
}
```

```
public String getNome() {
    return nome;
}

public void setNome(String nome) {
    this.nome = nome;
}

public int getIdade() {
    return idade;
}

public void setIdade(int idade) {
    this.idade = idade;
}

class Cachorro extends Animal {
    private String raca;

    public Cachorro(String nome, int idade, String raca) {
        super(nome, idade);
        this.raca = raca;
    }

    public String getRaca() {
        return raca;
    }

    public void setRaca(String raca) {
        this.raca = raca;
    }
}

class Gato extends Animal {
    private String cor;
```

## 10 Sobrescrita

```
public Gato(String nome, int idade, String cor) {
    super(nome, idade);
    this.cor = cor;
}

public String getCor() {
    return cor;
}

public void setCor(String cor) {
    this.cor = cor;
}

class Ave extends Animal {
    private String especie;

    public Ave(String nome, int idade, String especie) {
        super(nome, idade);
        this.especie = especie;
    }

    public String getEspecie() {
        return especie;
    }

    public void setEspecie(String especie) {
        this.especie = especie;
    }
}

public class Main {
    public static void main(String[] args) {
        Cachorro cachorro = new Cachorro("Mel", 3, "Labrador");
        Gato gato = new Gato("Nina", 5, "Preto");
        Ave ave = new Ave("Pituquinha", 2, "Calopsita");

        System.out.println(cachorro.getNome() + " - " + ca-
chorro.getIdade() + " - " + cachorro.getRaca());
    }
}
```

```
System.out.println(gato.getNome() + " - " + gato.  
getIdade() + " - " + gato.getCor());  
System.out.println(ave.getNome() + " - " + ave.getI-  
dade() + " - " + ave.getEspecie());  
}  
}
```



### Saiba mais

---

A palavra-chave “super” tem duas funcionalidades em Java. A primeira é permitir a chamada ao construtor da superclasse. Quando utilizada, deve ser a primeira instrução no construtor da subclasse, com a notação `super(parâmetros)`. Já a segunda funcionalidade é permitir a chamada explícita a um método da superclasse, utilizando a notação `super.nomeDoMetodo`. Isso permite chamar um método da superclasse mesmo que sobreescrito pela subclasse (HORSTMANN, 2009).

---

A herança é um conceito importante em POO, para que as classes sejam organizadas em uma hierarquia de classes e para promover a reutilização de código. Dessa forma, a herança permite que as subclasses compartilhem atributos e comportamentos com a superclasse, ao mesmo tempo em que adicionam novos comportamentos e atributos, específicos para cada subclasse. Isso ajuda a criar um código mais organizado, mais fácil de entender e de manter, além de aumentar a eficiência na programação.



### Fique atento

---

É importante salientar que a herança deve ser usada com cuidado, pois pode levar a uma hierarquia de classes muito complexa e difícil de entender. Além disso, a herança pode criar dependências entre as classes, dificultando a manutenção do código (DEITEL; DEITEL, 2008).

---

Na próxima seção, serão apresentados códigos para ilustrar a construção de uma aplicação com a estrutura de herança e a sobreescrita de métodos. Com esse exemplo prático, você pode compreender melhor como utilizar esses conceitos para criar uma estrutura de classes mais eficiente e modular, resultando em inúmeros benefícios para o desenvolvimento de software, como

uma organização melhor do código, facilidade de entendimento e leitura do código, redução de códigos duplicados, reutilização de códigos em outras partes do sistema, etc.

## Construção de aplicação com estrutura de herança e sobrescrita de métodos

A estrutura de herança e a sobrescrita de método são técnicas poderosas da POO, que, em conjunto, resultam em aplicações eficientes e personalizadas. Como vimos, a herança permite que as classes herdem métodos e atributos de uma superclasse, enquanto adicionam seus próprios elementos, o que é notadamente útil quando há classes que compartilham um grande código em comum (HORSTMANN, 2009).

Já a sobrescrita de método, que é uma técnica importante no contexto da herança, permite que uma subclasse substitua a implementação de um método de sua superclasse, personalizando o comportamento do método de acordo com suas próprias necessidades. Combinadas, essas técnicas possibilitam uma reutilização de código mais eficiente e a criação de aplicações personalizadas (DEITEL; DEITEL, 2008).

Considere a classe `Animal` e suas subclasses `Cachorro` e `Gato` no Exemplo 5.

### Exemplo 5

```
class Animal {  
    public String emitirSom() {  
        return "som genérico do animal";  
    }  
}  
  
class Cachorro extends Animal {  
    @Override  
    public String emitirSom() {  
        return "au au";  
    }  
}
```

```

class Gato extends Animal {
    @Override
    public String emitirSom() {
        return "miau";
    }
}

public class Main {
    public static void main(String[] args) {
        Animal animal = new Animal();
        Cachorro cachorro = new Cachorro();
        Gato gato = new Gato();

        System.out.println(animal.emitirSom()); //exibe a
frase: som genérico do animal
        System.out.println(cachorro.emitirSom()); //exibe
a frase: au au
        System.out.println(gato.emitirSom()); //exibe a
frase: miau
    }
}

```

O código do Exemplo 5 define três classes: Animal, Cachorro e Gato. A classe Animal tem um método público chamado `emitirSom()`, que retorna uma string com o som genérico de um animal. Já Cachorro e Gato herdam da classe Animal, o que significa que elas têm todos os métodos e propriedades desta, além dos seus próprios métodos e propriedades. Cachorro e Gato sobrescrevem o método `emitirSom()` da classe Animal, retornando os sons característicos de um cachorro e de um gato, respectivamente.

Para utilizar essas classes, criou-se uma classe Main, que tem um método estático `main()`. No método `main()`, foram criados objetos das classes Cachorro e Gato e chamado o método `emitirSom()` em cada objeto. O resultado é a impressão dos sons característicos de cada animal. Por exemplo, a classe Cachorro pode sobrepor o método `emitirSom()` para retornar Au Au, enquanto a classe Gato pode sobrepor o mesmo método para retornar Miau. Isso permite que cada objeto de Cachorro e Gato emita o som correto, mesmo que ambos os objetos sejam do tipo Animal.

É importante salientar que, ao utilizar a sobrescrita de método, deve-se atentar para o fato de que o método da subclasse deve ter o mesmo nome e os mesmos parâmetros que o método da superclasse. Além disso, é preciso utilizar a anotação `@Override` para indicar que o método está sobreescrivendo um método da superclasse. Dessa forma, o compilador Java saberá que deve usar a implementação da subclasse em vez da implementação da superclasse (DEITEL; DEITEL, 2008).

A combinação de herança e sobrescrita, portanto, é uma ferramenta útil para a construção de aplicações Java, já que, com ela, as classes compartilham um código em comum e, ao mesmo tempo, adicionam comportamentos específicos para atender às necessidades de cada caso. No entanto, é importante usar essa técnica com cuidado e garantir que a estrutura de herança esteja bem projetada e organizada.



### ***Fique atento***

---

Os métodos privados não são sobreescritos, porque são acessíveis somente dentro da própria classe, e não pelas subclasses. A sobreescrita de métodos só é aplicável a métodos herdados pela subclasse e acessíveis a partir da subclasse. Como os métodos com acesso *private* não podem ser acessados pelas subclasses, não podem ser sobreescritos.

---

Neste capítulo, você estudou os conceitos fundamentais de sobreescrita de método e herança e como implementar a sobreescrita na linguagem Java. A sobreescrita é uma técnica poderosa, que permite que uma subclasse substitua a implementação de um método de sua superclasse. Assim, a subclasse personaliza o comportamento do método de acordo com suas necessidades. Além disso, as classes podem herdar os métodos e atributos de suas superclasses, o que é útil para compartilhar código e criar hierarquias de classes mais simples e organizadas. Por fim, você viu como a implementação de sobreescrita de método pode ser usada com a herança para personalizar o comportamento de um método numa subclasse.

## Referências

- DEITEL, H. M.; DEITEL, P. J. *Java: como programar*. 6. ed. Porto Alegre: Bookman, 2008.
- HORSTMANN, C. *Conceitos de computação com Java*. 5. ed. Porto Alegre: Bookman, 2009.
- ORACLE. Java tutorials: multiple inheritance of state, implementation, and type. *Oracle*, 2022. Disponível em: <https://docs.oracle.com/javase/tutorial/java/iandl/multipleinheritance.html>. Acesso em: 22 maio 2023.
- SEBESTA, R. W. *Conceitos de linguagens de programação*. 11. ed. Porto Alegre: Bookman, 2018.

## Leituras recomendadas

CARVALHO, T. L. *Orientação a objetos: aprenda seus conceitos e suas aplicabilidades de forma efetiva*. São Paulo: Casa do Código, 2016.

SCHILD, H. *Java para iniciantes: crie, compile e execute programas Java rapidamente*. 6. ed. Porto Alegre: Bookman, 2015.



### Fique atento

Os *links* para sites da web fornecidos neste capítulo foram todos testados, e seu funcionamento foi comprovado no momento da publicação do material. No entanto, a rede é extremamente dinâmica; suas páginas estão constantemente mudando de local e conteúdo. Assim, os editores declararam não ter qualquer responsabilidade sobre qualidade, precisão ou integralidade das informações referidas em tais *links*.

Conteúdo:

**sagah**<sup>+</sup>



# Dica do Professor

---

A implementação da sobrescrita de método na POO exige a definição prévia de herança e classes, incluindo a superclasse e suas subclasses. Com a sobrescrita é possível alterar o comportamento das subclasses e criar novos códigos com base no código-base do programa.

Nesta Dica do Professor, é possível aprender a como implementar a sobrescrita de método em Java, utilizando os recursos e componentes da linguagem para conduzir esse processo. Compreender como realizar a sobrescrita é fundamental para a construção de programas mais flexíveis e eficientes, que possam ser facilmente adaptados e estendidos de acordo com as necessidades específicas de cada projeto. A sobrescrita de método é uma das principais técnicas em POO e pode ser aplicada em diferentes linguagens de programação.



Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

# Exercícios

- 1) A sobrescrita permite aplicar a modelagem sobre diagramas de classes e suas definições. Além disso, pode ser elaborada considerando a declaração de nome de uma classe com uma ou mais subclasses atribuídas a si.

Um método pode ser sobreescrito apenas quando:

- A) tem o modificador de acesso *private*.
  - B) tem o mesmo nome da classe à qual pertence.
  - C) é um construtor.
  - D) é acessível.
  - E) tem o mesmo tipo de retorno.
- 2) Em alguns casos, existe a necessidade de herdar habilidades em Java, porém é necessário garantir a possibilidade e personalizá-las. A herança deve considerar a mudança de comportamento entre as classes.

Nesse contexto, ao sobreescrivar um método adotado pela linguagem de programação Java e outras orientadas a objeto, é necessário considerar que o ato praticado será:

- A) substituir a implementação do método da superclasse.
  - B) escrever um método herdado com o mesmo tipo de retorno, mas com nome diferente.
  - C) atribuir assinaturas diferentes; todavia, a implementação deve ser igual.
  - D) garantir que as assinaturas e as implementações sejam diferentes daquelas na superclasse.
  - E) escrever o método com mesmo nome, modificando apenas seus parâmetros.
- 3) Métodos representam uma parte essencial na POO, utilizados para que seja possível consultar e alterar atributos elencados aos objetos.

Sobre as características de um método na sobrescrita, é correto afirmar que:

- A) os métodos que sobrescrevem não têm especificadores de acesso.
  - B) uma subclasse não pode alterar o acesso aos métodos da superclasse.
  - C) um método declarado como *protected* na superclasse pode ser declarado *public* na subclasse.
  - D) um método declarado como *protected* na superclasse pode ser declarado *private* na subclasse.
  - E) o método que sobrescreve não pode ser final.
- 4) Em Java, implementar classes é possível por meio de palavras-chaves. Para definir classes públicas, é necessário declarar a classe por meio do comando *public class*; já para modificar o acesso do método utiliza-se *public void*.

Considerando as informações repassadas, analise o código a seguir:

```
public class Animal{  
    public void locomover(){  
        System.out.println("Se locomove");  
    }  
}  
  
public class Peixe extends Animal {  
    public void locomover(){  
        System.out.println("Nada");  
    }  
}
```

Ao analisar o código, está correto afirmar que:

- A) a classe Animal é uma subclasse de Peixe.
- B) a classe Peixe é uma classe genérica.
- C) ao instanciar a classe Peixe e, por meio de uma variável de referência, chamar o método *locomover()*, a saída que se tem é: Se locomove.
- D) ao instanciar a classe Peixe e, por meio de uma variável de referência, chamar o método *locomover()*, a saída que se tem é: Nada.
- E) o método *locomover()*, herdado pela subclasse Animal, foi sobreescrito, pois sua assinatura e seu comportamento foram alterados.

5) A sobrescrita utiliza um conjunto de modificadores para definir o conceito de herança.

Em relação à herança e à sobrescrita, é correto afirmar que:

- A) a sobrescrita de métodos permite estender um código existente.
- B) um método pode ser sobreescrito somente se for acessível.
- C) o método da subclasse não sobrescreve o método *private* da superclasse.
- D) invocações de métodos *private* sempre invocam a implementação do método declarado na classe atual.
- E) uma invocação externa do método da subclasse, acessível fora de sua classe, resulta na invocação da implementação da superclasse.

## Na prática

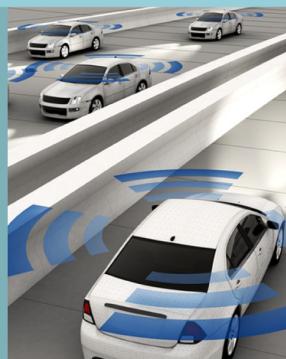
A partir da sobrescrita, é possível especializar os métodos e a forma como eles são herdados entre as classes, alternando o comportamento dos elementos de um código. O método sobrescrito basicamente fornece uma nova versão de determinada propriedade considerando as características da classe-pai, praticando a herança, mantendo a assinatura e garantindo o vínculo entre elas.

Neste Na Prática, você vai ver um exemplo que ilustra como a sobrescrita é utilizada na POO para resolver problemas e desenvolver soluções mais flexíveis e adaptáveis.

# UTILIZANDO A SOBRESCRITA NA CRIAÇÃO DE CARROS AUTÔNOMOS

A tecnologia dos carros autônomos tem se desenvolvido rapidamente nos últimos anos, trazendo novas possibilidades para a indústria automobilística. Nesse contexto, a POO tem um papel fundamental na criação desses veículos.

Neste exemplo prático, pode-se notar como a sobreescrita é utilizada na criação de carros autônomos.



Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

## Desenvolvendo a arquitetura

A empresa X está desenvolvendo um novo modelo de carro autônomo, que deve ser capaz de tomar decisões de forma independente, com base em sensores e informações coletadas em tempo real. Para isso, a equipe de desenvolvimento decidiu utilizar a linguagem Java aplicando POO para criar uma arquitetura de software robusta e escalável.

```
public void frear() {  
    // implementação padrão de frear  
}  
  
public void virar() {  
    // implementação padrão de virar  
}  
  
public void parar() {  
    // implementação padrão de parar  
}
```

A classe principal do sistema é a "**CarroAutonomo**", que conta com diversos métodos para controlar o veículo, como "acelerar", "frear", "virar" e "parar". No entanto, **cada um desses métodos precisa de uma implementação específica para o modo autônomo**, levando em conta as informações coletadas pelos sensores do carro.

## Classe-filha

Para resolver o problema encontrado, a equipe de desenvolvimento utiliza a sobreescrita na criação das classes-filhas. Por exemplo, a classe "**CarroAutonomoAcelerar**" herda da classe "**CarroAutonomo**" e sobre escreve o método "acelerar" com uma nova implementação, levando em conta a velocidade máxima do veículo e as condições da pista.

```
public class CarroAutonomoAcelerar extends CarroAutonomo {  
    @Override  
    public void acelerar() {  
        // implementação específica para o modo autônomo de acelerar  
        // leva em conta a velocidade máxima do veículo e as  
        // condições da pista  
    }  
}
```

Outra classe-filha, "**CarroAutonomoVirar**", sobre escreve o método "**virar**" com uma nova implementação. Ela leva em conta a direção do veículo e as informações coletadas pelos sensores de navegação.

```
public class CarroAutonomoVirar extends CarroAutonomo {  
    @Override  
    public void virar() {  
        // implementação específica para o modo autônomo de virar  
        // leva em conta a direção do veículo e as informações  
        // coletadas pelos sensores de navegação  
    }  
}
```

Com a utilização da sobreescrita, a equipe de desenvolvimento da empresa X consegue criar um software mais flexível e adaptável, capaz de lidar com as diversas situações que podem surgir em um ambiente de condução autônoma. Essa abordagem também permite uma melhor organização do código, com cada classe-filha responsável por uma funcionalidade específica.

# Saiba mais

Para ampliar o seu conhecimento a respeito desse assunto, veja abaixo as sugestões do professor:

## Definições e termos de Java e orientação a objetos

Neste artigo, você pode revisar os principais contextos e termos utilizados na POO e na linguagem Java. Você vai ver conceitos sobre sobrescrita, assinatura, classes e métodos, além de códigos que aplicam os conceitos na prática.



Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

## Sobrecarga e sobreposição de métodos em orientação a objetos

Veja, neste vídeo, como funcionam a sobrecarga e a sobreposição de métodos em orientação a objetos e como esses dois conceitos podem ajudar você na hora de programar.



Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

## Polimorfismo com sobrescrita

Acompanhe no Capítulo 8, **Polimorfismo e coerções** (p. 302), da obra a seguir. Você pode aprofundar seus conhecimentos sobre o uso do polimorfismo, incluindo a sobrescrita e tipo objeto *versus* tipo referência.

Conceúdo interativo disponível na plataforma de ensino!