POLIMORFISMO

Polimorfismo

- No dicionário, *polimorfismo* significa **formas múltiplas** e se refere a um princípio da biologia em que um organismo ou espécie pode ter várias formas ou estágios diferentes.

- Este princípio pode ser aplicado na Programação Orientada a Objetos e em linguagens como Java.

- As subclasses de uma superclasse podem definir comportamentos próprios e, ainda assim, compartilhar um pouco das mesmas funcionalidades da classe pai.

Polimorfismo

- No Java, em particular, o polimorfismo se manifesta claramente na **chamada de métodos**, quando objetos de tipos diferentes, mas relacionados por uma hierarquia de classes, são capazes de acionar **um mesmo método**, mas produzindo resultados diferentes.

- É também o polimorfismo que entra em ação na resolução da chamada de **métodos sobrecarregados**, quando o método acionado é selecionado com base na lista dos tipos dos argumentos selecionados.

Simplificando...

- Na Orientação a Objeto, o polimorfismo que permite que um mesmo método possa ser executado de formas diferentes.

- Os dois tipos de Polimorfismo mais utilizados são o de **sobrecarga** e de **sobreposição**.

Sobreposição: Mesma assinatura, diferentes classes

Sobrecarga: Diferentes assinaturas, mesma classe

Sobrecarga de Métodos

- No contexto do Java, a assinatura de método é o nome de um método mais sua lista de parâmetros.

- É possível, manter a mesma semântica (nome) para operações semelhantes que podem ser realizadas de modos diferentes, simplificando o uso da classe.

- A sobrecarga de métodos permite a existência de dois ou mais métodos com o mesmo nome, desde que sua declarações de parâmetros sejam diferentes.

Sobrecarga de Métodos

Para sobrecarregar um método, em geral, só temos que declarar versões diferentes dele. O compilador do Java se encarrega do resto.

O tipo e/ou a quantidade dos parâmetros deve diferir. Se não houver parâmetros, a assinatura será de nenhum parâmetro e sem tipo.

- **public void calcularMedia (float n1, float n2)**
  Dois parâmetros do tipo float
- **public void calcularMedia (int a1, int a2)**
  Dois parâmetros do tipo int
- **public String calcularMedia (String bim, int x1, int x2)**
  Três parâmetros, um do tipo String e dois do tipo inteiro
- **public String calcularMedia (float n1; float n2; float n3)**
  Três parâmetros do tipo float

Assinatura do Método

- Em linguagens onde vários métodos podem ter o mesmo nome, você precisa ter uma outra forma de evitar a ambiguidade. O compilador precisa saber qual dos métodos com mesmo nome você está chamando. Então você precisa se valer de informações extras disponíveis no método para decidir.

- O mais comum é analisar os parâmetros. Se todos os parâmetros forem iguais, você tem o mesmo método, se apenas um desses parâmetros for diferente, você tem um método diferente. É possível que o retorno e outras informações possam ser analisadas também, mas isto não é comum já que podem trazer alguns problemas.

Adaptado de https://pt.stackoverflow.com/questions/39870/o-que-%C3%A9-a-assinatura-de-um-m%C3%A9todo/39871>. Acesso em 15 de Junho de 2020

Polimorfismo de Sobrecarga em Java - Exemplo 1

```
public class Pagamento {

    private double valor;

    public void pagar(String cartao) {

        if(cartao == "Débito"){
            System.out.println("Você pagou "+ this.valor + “no Débito");
        }
        else {
            this.valor = this.valor + 5;
            System.out.println("Você pagou "+ this.valor + “no Crédito");
        }
    }

    public void pagar(double dinheiro) {
        System.out.println("Você pagou " + dinheiro+ "em dinheiro");
}

    public void pagar(boolean credito, int parcelas) {
        if(credito){
            System.out.println("Crediário em” + parcelas +”x");
        }
        else{
            System.out.println(“Crediário Não Aprovado");
        }
    }
```

Na classe Pagamento ocorre a sobrecarga do método pagar(). Instânciada a classe Pagamento, dependendo do parâmetro passado, o método é o mesmo, mas o resultado é diferente, caracterizando Polimorfismo.

```
Pagamento p = new Pagamento();

p.setValor(300);

p.pagar(300);
Você pagou 300.0 em dinheiro

p.pagar("Crédito");
Você pagou 305.0 no Crédito

p.pagar(true,5);
Crediário em 5x
```

Polimorfismo de Sobrecarga em Java - Exemplo 2

```
public class Aluno {

    public double media;

    public void mediaFinal(double n1, double n2) {
        this.media = n1 + n2 / 2;
        System.out.println("Curso Modular");
        System.out.println("Sua média final é " + media);
    }

    public void mediaFinal(double n1, double n2, double n3, double n4) {
        this.media = (n1 + n2 + n3 + n4) / 4;
        System.out.println("Curso Integrado");
        System.out.println("Sua média final é " + media);
    }

    public void mediaFinal(double n1, double n2, boolean posGraduado) {
        System.out.println("Concurso Público");
        if(posGraduado){
            this.media = (n1 + n2 + 2) / 3;
        }
        else{
            this.media = (n1 + n2 + 1) / 3;
        }
        System.out.println("Sua média final é " + media);
    }
}
```

Na classe Aluno ocorre a sobrecarga do método médiaFinal(). Instânciada a classe Aluno, dependendo do parâmetro passado, o método é o mesmo, mas o resultado é diferente, caracterizando Polimorfismo.

```
Aluno a = new Aluno();

a.mediaFinal(5.0, 8.5);
// Curso Modular
// Sua média final é 9.25

a.mediaFinal(7.0, 9.0, 8.0, 10);
// Curso Integrado
Sua média final é 8.5

a.mediaFinal(7.0,10.0,false);
// Concurso Público
// Sua média final é 6.0
```

Classes e Métodos Abstratos

- Existem situações em que é necessário definir uma superclasse que declare a estrutura de uma determinada abstração **sem fornecer uma implementação completa** de cada método.

- Ou seja: Criar uma superclasse definida de uma forma generalizada, que será compartilhada por todas as suas subclasses, **cabendo a cada subclasse o preenchimento dos detalhes**. Tal superclasse determina a natureza dos métodos que devem ser implementados pelas subclasses.

Classes e Métodos Abstratos

- As vezes, uma superclasse não é capaz de criar uma implementação significativa de um método.

- Neste caso, é preciso garantir que uma subclasse sobrescreva, de fato, todos os métodos necessários. Para isso, existem os **métodos abstratos**.

- É possível **exigir** que um método seja sobrescrito pela subclasse, especificando o modificador de tipo abstract. Para declarar um método abstrato, usamos a seguinte estrutura: **abstract tipo nome(parâmetros);** Não há corpo no método.

Classes e Métodos Abstratos

- Qualquer classe que contenha um ou mais métodos abstratos também deve ser declarada como abstrata.

- Para declarar uma classe abstrata, simplesmente use a palavra-chave **abstract** na frente da palavra-chave *class*, no início da declaração da classe.

- A classe abstrata tem uma importante característica: Classes abstratas não podem ser instanciadas, e só podem servir como superclasse, ou seja: **ela não pode gerar objetos**, somente ser herdada.

Classes e Métodos Abstratos

- Uma classe abstrata serve apenas como modelo para outra classe. No entanto, quem herda dessa classe não se torna abstrato.
- Classes abstratas podem conter um ou todos os **métodos abstratos**. Um método abstrato só pode ser colocado numa classe abstrata (ou em uma interface).
- Uma classe que tenha **pelo menos um método** abstrato deve ser obrigatoriamente abstrata. Mas ela pode não ter métodos abstrato.

```
public abstract class Eletrodomestico {

    private boolean ligado;
    private int voltagem;

    public abstract void ligar();
    public abstract void desligar();
```

```
public abstract class Funcionario {

    protected double salario;

    public double Bonificacao() {
        return this.salario * 1.2;
    }
```

Classes e Métodos Abstratos

```
public abstract class Eletrodomestico {

    private boolean ligado;
    private int voltagem;

    public abstract void ligar();
    public abstract void desligar();
```

Eletrodométicо é uma classe abstrata. As subclasses que estenderem dela (conceito de herança), deverão implementar os métodos ligar() e desligar();

Funcionário é uma classe abstrata. As subclasses que estenderem dela poderão herdar e reescrever o método bonificação();

```
public abstract class Funcionario {

    protected double salario;

    public double bonificacao() {
       return this.salario * 1.5;
    }
```

Sobreposição de Métodos

- Acontece quando substituimos um método de uma superclasse na sua subclasse, usando a mesma assinatura.

- A classe herda o método de outra e utiliza com o mesmo nome e assinatura, mas com outro comportamento.

- **Importante:** Não é possível fazer polimorfismo de sobreposição sem herança.

Sobreposição de Métodos

- Permite reescrever um método, ou seja, podemos reescrever nas subclasses os métodos criados inicialmente na superclasse.

- Os métodos que serão sobrepostos, diferentemente dos sobrecarregados, devem possuir o mesmo nome, tipo de retorno e quantidade de parâmetros do método inicial.

- Porém o mesmo será implementado com especificações da classe atual, podendo adicionar um algo a mais ou não.

Sobreposição de Métodos

- A habilidade de uma subclasse de sobrepor um método, permite a classe herdar de uma superclasse, cujo comportamento seja “suficientemente próximo” e então, modificar seu comportamento conforme necessário.

- Ao sobrepor um método é necessário usar a anotação @Override, que diz ao compilador que você pretende sobrepor um método na superclasse.

- Os métodos sobrescristos (outra forma de se referir) permitem ao Java suportar o polimorfismo na execução.

Sobreposição de Métodos

- A decisão sobre qual o método que deve ser selecionado, de acordo com o tipo da classe derivada, é tomada em tempo de execução.

- O polimorfismo é essencial para a programação Orientada a Objeto pois permite que uma classe genérica especifique métodos que serão comuns a todas as classes derivadas.

- Ao mesmo tempo, o polimorfismo permite que as subclasses definam a implementação específica de alguns ou todos os métodos.

Polimorfismo de Sobreposição em Java

```
public abstract class Forma {

    protected double dim1;
    protected double dim2;

    public abstract double area();
}
```

```
public class Retangulo extends Forma{

    @Override
    public double area() {
        System.out.print("Área do Retângulo:);
        return dim1 * dim2;
    }
}
```

@Override é uma anotação que deixa explícito no código que determinado método é a reescrita de um método da superclasse.

```
public class Triangulo extends Forma{

    @Override
    public double area() {
        System.out.print("Área do Triângulo: );
        return dim1 * dim2 / 2 ;
    }
}
```

A superclasse abstrata Forma possui o método area() que é herdado e implementado pelas subclasses com a mesma assinatura. Ocorre então uma sobrescrita de método: area() é reescrito nas subclasses Retângulo e Triângulo.

Polimorfismo de Sobreposição em Java

```
public class Principal {

    public static void main(String[] args) {

        Retângulo r = new Retângulo();
        Triangulo t = new Triangulo();

        r.dim1 = 9;
        r.dim2 = 3;
        System.out.println(r.area());
        // Área do Retângulo: 27.0

        t.dim1 = 7;
        t.dim2 = 6;
        System.out.println(t.area());
        // Área do Triângulo: 21.0
    }
}
```

Foram instanciadas as classes Retângulo e Triângulo, que são subclasses de Forma.

No código, o método area(), da classe Forma, que foi sobrescrito nas subclasses é chamado por cada uma das subclasses.

O mesmo método, herdado de Forma terá um comportamento diferente dependendo da classe (Triângulo ou Retângulo) que o chama.

Portanto, verificamos várias formas de comportamento para um mesmo método, em classes diferentes, o que caracteriza POLIMORFISMO.

Métodos Final

- Há momentos em que a sobrescrita de um método não deve ocorrer (por exemplo, quando uma implementação não deve ser mudada para não comprometer o sistema).

- Para impedir que a sobreposição de um método, especifique a palavra-chave **final** no início da declaração.

```
public final void mensagem(){
        System.out.print("Este método não pode ser sobrescrito")
}
```

- Ao tentar sobrescrever um método final, haverá um erro de compilação. É possível declarar um ou todos os métodos de uma classe como **final.**

Classe Final

- Também é possível impedir que uma classe seja herdada, utilizando novamente a palavra-chave **final**.

```
public final class Exemplo(){
        // Corpo da Classe
}
```

- Implicitamente, todos os métodos que a classe final venha a ter, serão também métodos final.

- Uma classe não pode ser abstract e final. Classes Abstratas são incompletas sozinhas e dependem das subclasses para fornecer implementações completas.

Referências Bibliográficas

JUNIOR, Peter Jandl. **Java: guia do programador.** 3 ed. São Paulo: Novatec, 2015.

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

GALLARDO, R.; KANNAN, S.; ZACKHOUR, S. B. **Tutorial Java.** 5 ed. Rio de Janeiro: Alta Books, 2015.

SCHILDT, Herbert. **Java: a referência completa.** 1 ed. Porto Alegre: Bookman, 2014.

Sites Consultados

https://www.profissionaisti.com.br/2010/10/paradigmas-de-programacao/
https://imasters.com.br/devsecops/paradigmas-de-programacao-sao-importantes
https://www.devmedia.com.br/os-4-pilares-da-programacao-orientada-a-objetos/9264
https://www.devmedia.com.br/conceitos-da-linguagem-java/5341
https://www.oficinadanet.com.br/post/14614-programacao-orientada-a-objetos
https://www.ramon.pro.br/o-paradigma-orientado-a-objetos/
http://www.dca.fee.unicamp.br/cursos/PooJava/desenvolvimento/umlclass.html
https://pt.slideshare.net/profDanielBrandao/encapsulamento-em-orientao-a-objetos
https://www.cursoemvideo.com/course/curso-de-poo-php/
https://www.cursoemvideo.com/course/curso-de-poo-java/
https://www.caelum.com.br/download/caelum-java-objetos-fj11.pdf
https://www.caelum.com.br/download/caelum-csharp-dotnet-fn13.pdf
http://www.tiexpert.net/programacao/java/criacao-de-classe.php
https://www.devmedia.com.br/java-declaracao-e-utilizacao-de-classes/38374
https://www.cursoemvideo.com
https://www.w3schools.com/java/java_methods.asp
https://www.w3schools.com/java/java_class_methods.asp
https://www.w3schools.com/java/java_modifiers.asp
http://www.tiexpert.net/programacao/java/this.php
http://www.tiexpert.net/programacao/java/new.php
https://www.w3schools.com/java/java_classes.asp
http://www.tiexpert.net/programacao/java/metodo-construtor.php
https://www.devmedia.com.br/java-declaracao-e-utilizacao-de-classes/38374
https://www.caelum.com.br/apostila-java-orientacao-objetos/heranca-reescrita-e-polimorfismo/#invocando-o-mtodo-reescrito
http://www.dca.fee.unicamp.br/cursos/PooJava/polimorf/polijex.html
http://www.dca.fee.unicamp.br/cursos/PooJava/polimorf/index.html#:~:text=Polimorfismo%20%C3%A9%20o%20princ%C3%ADpio%20pelo,objeto%20do%20tipo%20da%20superclasse.
https://www.devmedia.com.br/sobrecarga-e-sobreposicao-de-metodos-em-orientacao-a-objetos/33066
https://www.caelum.com.br/apostila-java-orientacao-objetos/classes-abstratas/#classe-abstrata
https://www.caelum.com.br/apostila-java-orientacao-objetos/heranca-reescrita-e-polimorfismo/#invocando-o-mtodo-reescrito

Material desenvolvido pelo
Prof. Rafael da Silva Polato
rafael.polato@etec.sp.gov.br

Divisão de Turma
Prof. Leandro Bordignon
leandro.bordignon01@etec.sp.gov.br