Aula: Polimorfismo e Interface

Objetivo da Aula

Entender o conceito de **polimorfismo** na programação orientada a objetos (POO), como ele funciona em Java, e como utilizá-lo para criar sistemas mais flexíveis e reutilizáveis.

O Que é Polimorfismo?

O **polimorfismo** significa "muitas formas". Em POO, é a habilidade de um objeto assumir diferentes formas, dependendo do contexto em que é utilizado.

- **Polimorfismo em ação**: Métodos podem se comportar de formas diferentes, mesmo que compartilhem o mesmo nome.
- **Benefício**: Permite que você escreva código mais genérico e flexível.

Tipos de Polimorfismo

1. **Polimorfismo de Sobrecarga** (*Compile-time*):
    - Ocorre quando há mais de um método com o mesmo nome, mas parâmetros diferentes na mesma classe.
2. **Polimorfismo de Substituição** (*Runtime*):
    - Ocorre quando uma classe filha substitui um método da classe pai (usando herança e sobrescrita).

Polimorfismo de Sobrecarga

Exemplo:

```java
class Calculadora {
    // Método para somar dois números inteiros
    public int somar(int a, int b) {
        return a + b;
    }

    // Método para somar três números inteiros
    public int somar(int a, int b, int c) {
        return a + b + c;
    }

    // Método para somar dois números double
    public double somar(double a, double b) {
        return a + b;
    }
}

public class TesteSobrecarga {
    public static void main(String[] args) {
        Calculadora calc = new Calculadora();

        System.out.println(calc.somar(5, 10));       // Resultado: 15
        System.out.println(calc.somar(5, 10, 20));  // Resultado: 35
        System.out.println(calc.somar(5.5, 3.3));   // Resultado: 8.8
    }
}
```

Explicação:

- O método somar tem **múltiplas formas**, dependendo dos argumentos fornecidos.
- Isso é possível porque o compilador escolhe qual método chamar com base nos parâmetros.

Polimorfismo de Substituição (Sobrescrita)

Requisitos:

- Usar **herança**.
- O método da classe filha deve ter **mesma assinatura** (nome e parâmetros) do método da classe pai.
- É necessário usar a anotação @Override para indicar que um método está sendo sobrescrito.

Exemplo:

```java
// Classe pai
class Animal {
    public void emitirSom() {
        System.out.println("Animal fazendo barulho");
    }
}

// Classe filha
class Cachorro extends Animal {
    @Override
    public void emitirSom() {
        System.out.println("Cachorro latindo: Au au!");
    }
}

class Gato extends Animal {
    @Override
    public void emitirSom() {
        System.out.println("Gato miando: Miau!");
    }
}

public class TestePolimorfismo {
    public static void main(String[] args) {
        Animal meuAnimal;

        meuAnimal = new Cachorro();
        meuAnimal.emitirSom();  // Resultado: Cachorro latindo: Au au!

        meuAnimal = new Gato();
        meuAnimal.emitirSom();  // Resultado: Gato miando: Miau!
    }
}
```

Explicação:

- A variável meuAnimal pode referenciar diferentes objetos (Cachorro e Gato).
- O método emitirSom é executado com base no tipo do objeto que está sendo referenciado, **não** pelo tipo da variável.

Polimorfismo com Interfaces

O que é uma Interface?

Uma **interface** é um tipo especial em Java que define um **contrato** para as classes que a implementam. Ela contém apenas declarações de métodos (sem implementação) e, opcionalmente, constantes.

Características de uma Interface:

1. Não pode ter métodos implementados (exceto default ou static a partir do Java 8).
2. Não contém atributos comuns, apenas constantes (static final).
3. Classes que implementam a interface devem implementar todos os métodos.

Exemplo:

```java
interface Pagamento {
    void pagar();
}

class CartaoCredito implements Pagamento {
    @Override
    public void pagar() {
        System.out.println("Pagamento realizado com cartão de crédito.");
    }
}

class Pix implements Pagamento {
    @Override
    public void pagar() {
        System.out.println("Pagamento realizado via Pix.");
    }
}

public class TesteInterface {
    public static void main(String[] args) {
        Pagamento meuPagamento;

        meuPagamento = new CartaoCredito();
        meuPagamento.pagar();  // Resultado: Pagamento realizado com cartão de crédito

        meuPagamento = new Pix();
        meuPagamento.pagar();  // Resultado: Pagamento realizado via Pix.
    }
}
```

Explicação:

- A interface Pagamento define um contrato para diferentes formas de pagamento.
- Cada classe que implementa a interface fornece sua própria implementação do método pagar.
- Permite que o sistema lide de forma uniforme com diferentes tipos de pagamentos.

Benefícios do Polimorfismo

1. **Flexibilidade**: Permite usar objetos de forma genérica.
2. **Reutilização de Código**: Evita duplicação, pois métodos genéricos podem ser aplicados a múltiplas classes.
3. **Extensibilidade**: Facilita a adição de novas funcionalidades sem modificar o código existente.

Exercício Prático

Crie um programa que simule um sistema de notificações, com diferentes tipos de envio:

- **Email**
- **SMS**
- **Push Notification**

Cada tipo deve implementar uma interface Notificacao com um método enviar(). No método principal, utilize polimorfismo para enviar notificações de diferentes tipos.

```java
interface Notificacao {
    void enviar(String mensagem);
}

class Email implements Notificacao {
    @Override
    public void enviar(String mensagem) {
        System.out.println("Email enviado: " + mensagem);
    }
}

class SMS implements Notificacao {
    @Override
    public void enviar(String mensagem) {
        System.out.println("SMS enviado: " + mensagem);
    }
}

class PushNotification implements Notificacao {
    @Override
    public void enviar(String mensagem) {
        System.out.println("Push Notification enviada: " + mensagem);
    }
}

public class TesteNotificacao {
    public static void main(String[] args) {
        Notificacao notificacao;

        notificacao = new Email();
        notificacao.enviar("Olá via Email!");

        notificacao = new SMS();
        notificacao.enviar("Olá via SMS!");

        notificacao = new PushNotification();
        notificacao.enviar("Olá via Push Notification!");
    }
}
```

Conclusão

O polimorfismo é uma das características mais poderosas da POO, permitindo que você escreva código mais genérico, reutilizável e extensível. Ele é amplamente usado em frameworks e APIs, como no Spring Framework, para garantir flexibilidade e escalabilidade no desenvolvimento de sistemas.

Pratique bastante com diferentes cenários para dominar esse conceito! 🚀