Orientação a objetos – Polimorfismo

O polimorfismo é a possibilidade de utilizar um objeto "como se fosse" um outro. Embora o conceito seja esse, algumas publicações relacionadas ao JAVA e à orientação de objetos fazem abordagens diferentes. Então, podemos considerar basicamente três tipos de polimorfismo:

- Métodos;
- Classe;
- Interface.

1. **De métodos**

   Muitos autores consideram polimorfismo a possibilidade de utilizar dois ou mais métodos com a mesma assinatura, mas com comportamentos (codificação) diferentes. Basta lembrar que já abordamos um recurso da linguagem Java que permite exatamente isso: a sobrescrita.

   A questão não é avaliar se essa visão está correta ou não. Mesmo porque, de certa forma, a sobrescrita possibilita o uso de um método que pode assumir várias formas (executar tarefas diferentes, de acordo com sua codificação) a partir da mesma chamada, sendo diferenciado pela classe que deu origem ao objeto, Isso justificaria chamar esse recurso de polimorfismo, mas acreditamos que é melhor definido como sobrescrita.

2. **De Classe**

   Considerando uma hierarquia de classes, temos, em uma superclasse, a generalização de um tipo e, em suas subclasses, a especialização do mesmo tipo, imagine a seguinte situação:

   - Se colocarmos uma cesta à disposição dos clientes da livraria e nela estiver escrito "Coloque aqui seus produtos", estes "produtos" podem ser livros, cd's ou dvd's. Ou ainda, todos eles juntos.

   Outro exemplo:

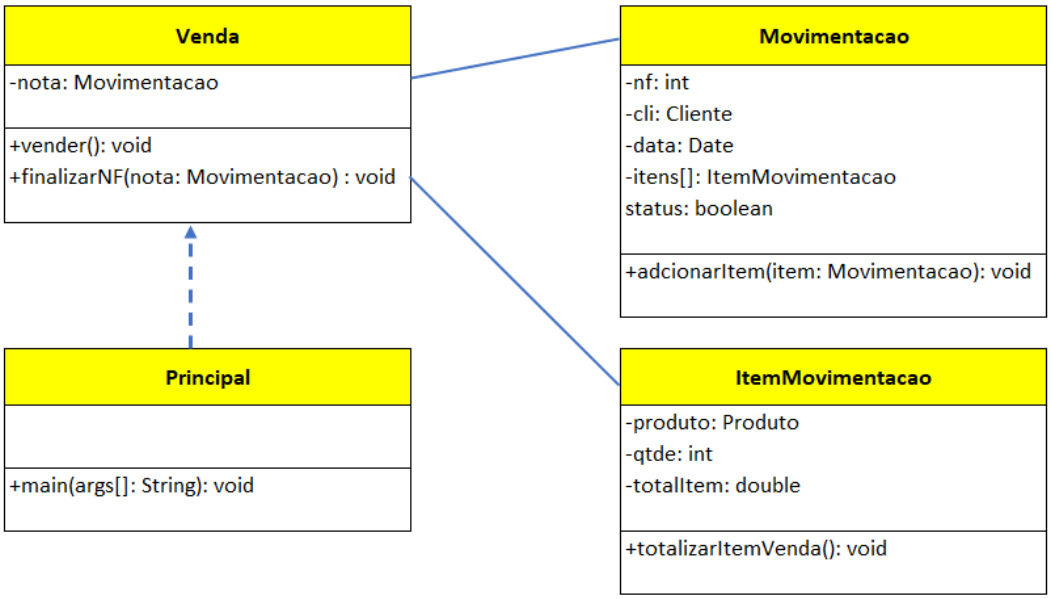
   - Na livraria, existe uma porta de acesso ao estoque e nela há uma placa com o aviso "Entrada permitida somente para funcionários"

   Tanto pode entrar um vendedor como um gerente, porque ambos são funcionários.

Esse mesmo princípio se aplica aos programas em Java. Se definirmos um método que recebe um objeto do tipo Produto por parâmetro, podemos passar qualquer objeto instanciado a partir de uma subclasse de Produto que ele será aceito. Da mesma forma, se um método espera um objeto do tipo Funcionário, é possível passar um objeto instanciado a partir da classe Vendedor, por exemplo, que será aceito sem problemas.

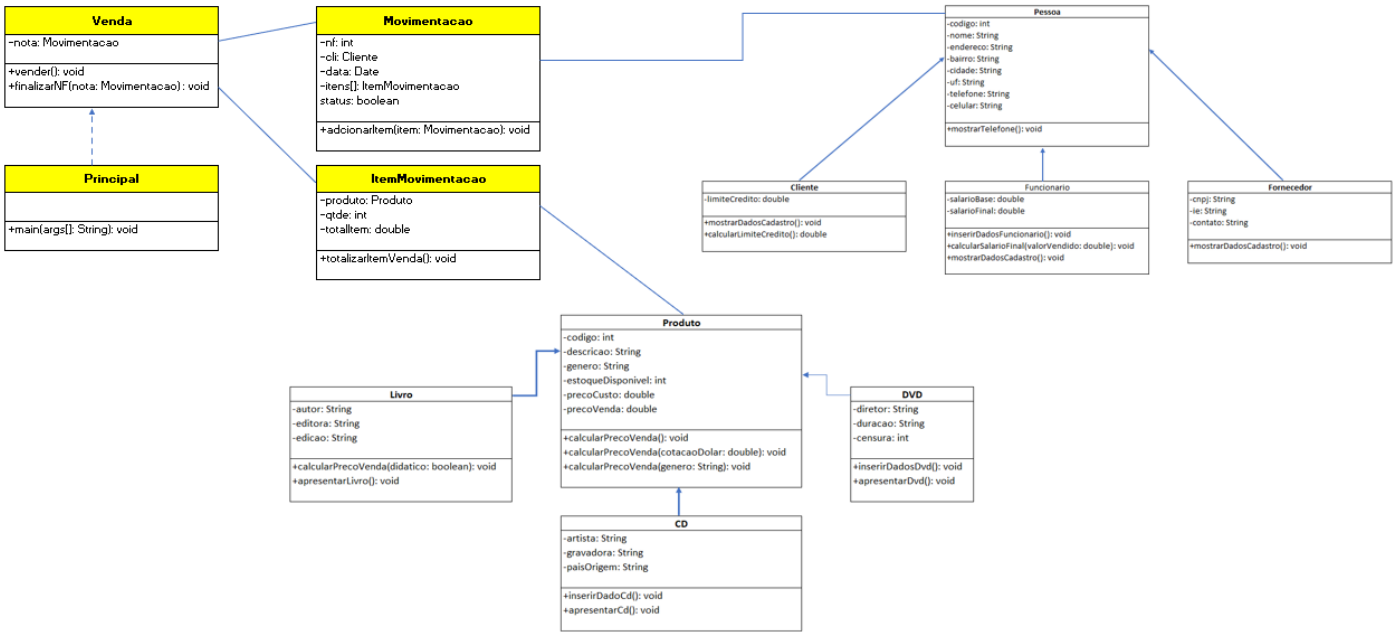
O raciocínio é o seguinte: "Um vendedor é um funcionário", assim como, "Um livro é um produto". A diferença é que vendedor foi definido a partir de uma especialização da classe Funcionário. Assim, pode ter atributos e métodos específicos referentes a vendedores, porém, não deixa de ser um Funcionário.

Apesar de um livro poder ser utilizado "como se fosse" um Produto, não deixa de "ser" um livro e de ter as características especificas de livro. O polimorfismo, portanto, é a possibilidade de utilizar um objeto como se fosse outro e não o transformar em outro objeto. O uso do polimorfismo de classe (ou de tipo) é mostrado na figura abaixo, que mostra o processo de venda da livraria.



Orientação a objetos - Polimorfismo

Diagrama completo do sistema da livraria



Resumo: Polimorfismo é a capacidade de uma referência de um tipo genérico referenciar um objeto de um tipo mais específico.

Exemplo 1:

Veja que o tipo da variável é Carro, mas o objeto colocado nela não é Carro e sim Ferrari e Fusca.

```
package com.mycompany.polimorfismoexemplo1;

public interface Carro {

    public void acelerar();

}
```

```
package com.mycompany.polimorfismoexemplo1;

public class Ferrari implements Carro {

    @Override
    public void acelerar() {
        System.out.println(x: "Ferrari acelerando…");
    }

}
```

```
package com.mycompany.polimorfismoexemplo1;

public class Fusca implements Carro {

    @Override
    public void acelerar() {
        System.out.println(x: "Fusca tentando acelerar…");
    }

}
```

```
package com.mycompany.polimorfismoexemplo1;

public class PolimorfismoExemplo1 {

    public static void main(String[] args) {
        Carro c = new Ferrari();
        c.acelerar();

        c = new Fusca();
        c.acelerar();

    }
}
```

Exemplo 2:

Novamente, o tipo da variável é Conexão, mas não referência um objeto do tipo Conexão, mas sim um objeto DialUp ou Adsl, essa variável com pode referenciar qualquer objeto que implemente a interface Conexão ou estenda a classe, se for uma.

```java
package com.mycompany.polimorfismoexemplo2;

public interface Conexao {

    public void conectar();

}
```

```java
package com.mycompany.polimorfismoexemplo2;

public class DialUp implements Conexao {

    @Override
    public void conectar() {
        System.out.println(x: "Modem discando…");
    }

}
```

```java
package com.mycompany.polimorfismoexemplo2;

public class AdSi implements Conexao {

    @Override
    public void conectar() {
        System.out.println(x: "Adsl conectado…");
    }

}
```

```java
package com.mycompany.polimorfismoexemplo2;

public class PolimorfismoExemplo2 {

    public static void main(String[] args) {
        Conexao con = new DialUp();
        con.conectar();

        con = new AdSi();
        con.conectar();

    }
}
```

Exemplo 3:

Vamos para a nossa loja de carros, onde você é o programador Java de lá.

Lembra que aprendeu, através de exemplos, a criar uma classe bem genérica “interface” chamada "Carro" e depois criamos várias subclasses, de fuscas, ferraris etc. Imagine que, todo ano, todos na empresa tem um aumento, a Ferrari teve aumento de 5%. o fusca terá aumento de 3% e o gol terá de 1%.

Note que, embora todos sejam "Carro", cada objeto terá que calcular seu aumento de forma diferente, pois terão diferentes valores de aumento.

Como criar, então, um método na superclasse que atenda todas essas necessidades diferentes?

Não é na superclasse que se resolve, mas nas subclasses, criando o método “AumentoAnual()” em cada uma. Ou seja, vai criar vários métodos, e para fazer o aumento realmente ocorrer de maneira correta, é só invocar o método do objeto específico.

Portanto: objetoFerrari.aumento() é diferente de objetoFusca.aumento().

Note que usamos o mesmo nome do método para todas as subclasses, porém cada método é diferente um do outro. Isso é o polimorfismo em ação: embora todos os objetos sejam "Carro", eles terão uma forma diferente de agir, pois implementamos os métodos de maneira diferente. Apenas invocamos, e todo objeto sabe exatamente o que fazer.

Por isso o nome polimorfismo, pois cada objeto terá sua forma própria de como rodar, pois os métodos 'AumentoAnual()' dos objetos são diferentes.

```
package com.mycompany.concessionaria;

public interface Carro {

    public double AumentoAnual();

    public void ApresentaCarro();

}
```

```
package com.mycompany.concessionaria;

public class Fusca implements Carro {
    public double valorCarro;

    public Fusca() {
        this.valorCarro = 50000;
    }

    @Override
    public double AumentoAnual() {
        this.valorCarro *= 1.10;
        return this.valorCarro;
    }

    @Override
    public void ApresentaCarro() {
        System.out.println("Valor do Fusca: R$ " + String.format("%.2f", this.valorCarro));
    }

}
```

```
package com.mycompany.concessionaria;

public class Golf implements Carro {
    public double valorCarro;

    public Golf() {
        this.valorCarro = 180000;
    }

    @Override
    public double AumentoAnual() {
        this.valorCarro *= 1.20;
        return this.valorCarro;
    }

    @Override
    public void ApresentaCarro() {
        System.out.println("Valor do Golf: R$ " + String.format("%.2f", this.valorCarro));
    }

}
```

```
package com.mycompany.concessionaria;

public class Renault implements Carro {
    public double valorCarro;


    public Renault() {
        this.valorCarro = 100000;
    }

    @Override
    public double AumentoAnual() {
        this.valorCarro *= 1.07;
        return this.valorCarro;
    }

    @Override
    public void ApresentaCarro() {
        System.out.println("Valor do Renault: R$ " + String.format("%.2f", this.valorCarro));
    }

}
```

```
package com.mycompany.concessionaria;

public class Concessionaria {

    public static void main(String[] args) {
        Carro objeto;

        objeto = new Fusca();
        objeto.AumentoAnual();
        objeto.ApresentaCarro();

        objeto = new Golf();
        objeto.AumentoAnual();
        objeto.ApresentaCarro();
```

```
        objeto = new Renault();
        objeto.AumentoAnual();
        objeto.ApresentaCarro();

    }
}
```

Outra vantagem do polimorfismo: você já viu que, criando o método “AumentoAnual()” em toda as subclasses, ela agirão de maneira independente da superclasse e diferente de outros objetos. Agora, quando chegar outro carro na sua loja você de adicionar o método “AumentoAnual()”, e terá um novo tipo de objeto, sem grandes alterações no código.