Polimorfismo

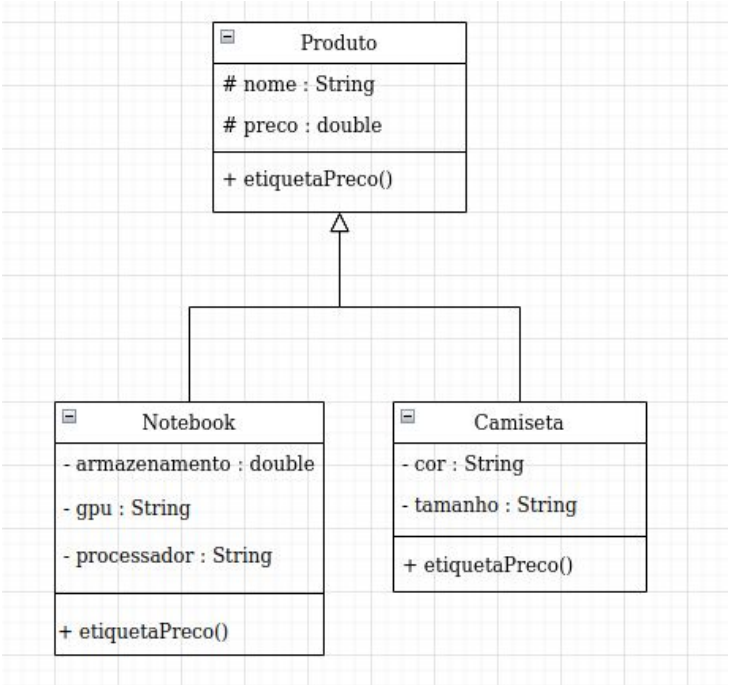
Várias formas de se fazer algo

Relembrando o conceito de Herança

- Recurso do Paradigma orientado a objetos;
- Possibilita **herdar estados** e **comportamentos** de uma classe (**Superclasse ou classe mãe**). A partir de tal conceito, permite-se economizar linhas de código, desde que classes distintas originam de uma mesma abstração.

Relembrando o conceito de Herança

- Iremos utilizar o seguinte UML para as explicações:



Modelando as classes utilizando Herança - Produto

```java
public class Produto {
    protected String nome;
    protected double preco;

    public Produto(String nome, double preco) {
        this.nome = nome;
        this.preco = preco;
    }

    public void etiquetaPreco(){
        System.out.println("Etiqueta padrão com valor: " + this.preco);
    }
}
```

Modelando as classes utilizando Herança - Camiseta

```java
public class Camiseta extends Produto{
    private String cor;
    private String tamanho;

    public Camiseta(String nome, double preco, String cor, String tamanho) {
        super(nome, preco);
        this.cor = cor;
        this.tamanho = tamanho;
    }

    @Override
    public void etiquetaPreco() {
        System.out.println("Etiqueta personalizada para Camiseta. Valor: " + this.preco);
    }
}
```

Modelando as classes utilizando Herança - Notebook

```java
public class Notebook extends Produto{
    private double armazenamento;
    private String gpu;
    private String processador;

    public Notebook(String nome, double preco, double armazenamento, String gpu, String processador) {
        super(nome, preco);
        this.armazenamento = armazenamento;
        this.gpu = gpu;
        this.processador = processador;
    }

    @Override
    public void etiquetaPreco() {
        System.out.println("Etiqueta personalizada para Notebook. Valor: " + this.preco);
    }
}
```

Compreendendo o Polimorfismo

- Polimorfismo (Poli - várias e Morfo - formas);
- Da sua tradução, o polimorfismo **permite fazer alguma coisa de múltiplas formas**. Dessa forma, um mesmo nome (**assinatura**) pode representar **vários comportamentos**;
- É considerado um dos pilares do paradigma orientado a objetos;

Exemplificando o Polimorfismo

- Como podemos observar, a classe Produto possui um método, chamado **“etiquetaPreco”**.

```
public void etiquetaPreco(){
    System.out.println("Etiqueta padrão com valor: " + this.preco);
}
```

- Dessa forma, cada produto específico pode implementar a sua própria forma de etiquetar o produto;
- Em sua essência o polimorfismo se aplica a essa situação, em que **um mesmo nome (método com a mesma assinatura) possui diferentes comportamentos dado a classe que o implementa**;

Sobrescrita de métodos

- Como é possível implementar um comportamento específico, que já foi escrito na classe mãe?
- Para isso, utiliza-se a anotação ***@Override***, a qual indica que há uma sobrescrita do método da superclasse. Logo, há uma **especialização** do **método** na **subclasse**.

Sobrescrita na classe Camiseta

Sobrescrita do método:

```java
@Override
public void etiquetaPreco() {
    System.out.println("Etiqueta personalizada para Camiseta. Valor: " + this.preco);
}
```

Instanciando uma camiseta e chamando o método **etiquetaPreco**:

```java
Camiseta camiseta = new Camiseta( nome: "Nike", preco: 50, cor: "Vermelho", tamanho: "M");
camiseta.etiquetaPreco();
```

Resultado:

```
Etiqueta personalizada para Camiseta. Valor: 50.0
```

Sobrescrita na classe Notebook

Sobrescrita do método:

```java
@Override
public void etiquetaPreco() {
    System.out.println("Etiqueta personalizada para Notebook. Valor: " + this.preco);
}
```

Instanciando um notebook e chamando o método **etiquetaPreco**:

```java
Notebook notebook = new Notebook( nome: "Acer-e15", preco: 1000, armazenamento: 256, gpu: "1050ti", processador: "i5");
notebook.etiquetaPreco();
```

Resultado:

```
Etiqueta personalizada para Notebook. Valor: 1000.0
```

Continuando sobre o Polimorfismo

Agora, vamos criar uma abstração da etiquetadora:

```
public class Etiquetadora {
    private String partNumber;
    private String nome;

    public Etiquetadora(String partNumber, String nome) {
        this.partNumber = partNumber;
        this.nome = nome;
    }

    public void etiquetar(Produto[] produtos){
        for (int i = 0; i < produtos.length; i++) {
            if(produtos[i] != null){
                produtos[i].etiquetaPreco();
            }
        }
    }
}
```

Referência através da superclasse

- Como notebook e camiseta herdam de produto, estes são um **Produtos**. Dessa forma, uma instância de **Notebook** pode ser referenciado por uma variável do tipo **Produto**. Ainda assim, uma instância de **Camiseta** também pode ser referenciada por uma variável do tipo **Produto.**
- Isso amplia nossas possibilidades, uma vez que objetos que possuam parentesco podem ser **referenciados** pelas suas **superclasses**. Isso quer dizer que podemos agrupá-los em um **Array** de **Produtos**?

Continuando sobre Polimorfismo

Criando as referências a adicionando ao nosso ArrayList:

```
Produto[] produtos = new Produto[2];
Notebook notebook = new Notebook( nome: "Acer-e15", preco: 1000, armazenamento: 256, gpu: "1050ti", processador: "i5");
Camiseta camiseta = new Camiseta( nome: "Nike",  preco: 50,  cor: "Vermelho",  tamanho: "M");
produtos[0] = notebook;
produtos[1] = camiseta;
```

Agora podemos etiquetar vários produtos, mas para isso iremos utilizar a nossa Etiquetadora. Primeiro, criaremos uma instância desta classe e invocaremos o método etiquetar. Dessa forma, iremos passar nosso Array como parâmetro para ele, **o qual irá se encarregar de etiquetar cada produto da maneira que este necessita ser etiquetado (Através do método etiquetaPreco implementado em cada subclasse)!**

```
Etiquetadora etiquetadora = new Etiquetadora( partNumber: "PRZFV",  nome: "Etiq-01");
etiquetadora.etiquetar(produtos);
```

Vamos aos resultados

- **Método invocado:**

```
public void etiquetar(Produto[] produtos){
    for (int i = 0; i < produtos.length; i++) {
        if(produtos[i] != null){
            produtos[i].etiquetaPreco();
        }
    }
}
```

- **Resultado:**

```
Etiqueta personalizada para Notebook. Valor: 1000.0
Etiqueta personalizada para Camiseta. Valor: 50.0
```

- Como notebook e camiseta são produtos, o método invocado por cada produto será o que foi sobrescrito nas classes filhas e não o da classe mãe. Como uma referência do tipo produto “conhece” o método **etiquetaPreco**, ela consegue invocá-lo sem problemas.

Conclusão

- A partir disso, é possível deixar o software com uma maior possibilidade de evolução sem afetar múltiplas partes deste.
- Como é possível assumir diferentes comportamentos (com o método etiquetaPreco), utilizando uma referência do tipo produto, não devemos nos preocupar com o tipo de produto que está sendo “etiquetado”;
- **OBS.: CASO DESEJA-SE INVOCAR COMPORTAMENTOS ESPECÍFICOS DE CADA CLASSE, QUE NÃO TENHAM NAS DEMAIS, DEVE-SE UTILIZAR VARIÁVEIS DE REFERÊNCIA IGUAIS A INSTÂNCIA OU FAZER UM DOWNCASTING;**

Muito obrigado!