C206 - POO

Arthur Openheimer

Inatel

Informações Gerais

- Atendimento → Terça-feira 17:30-19:10, prédio 1, sala 19
- Email → arthur.openheimer@ges.inatel.br
- Github →https://github.com/ArthurOpenheimer/C206-Monitoria

Inatel

Herança

Herança é o segundo dos 4 pilares de POO, com ela podemos reutilizar e reaproveitar os códigos desenvolvidos. Fazemos isso criando novas Classes baseadas em Classes já criadas anteriormente

Herança

```java
public class Professor {
    private String nome;
    private int idade;
    private double salario;

    @Override
    public String toString() {
        return "idade=" + idade + ", nome="
        + nome + ", salario=" + salario;
    }

    public void corrigirProvas() {
        System.out.println("Corrigindo provas...");
    }

}
```

```java
public class Engenheiro {
    private String nome;
    private int idade;
    private double salario;

    @Override
    public String toString() {
        return "idade=" + idade + ", nome="
        + nome + ", salario=" + salario;
    }

    public void calcular() {
        System.out.println("Calculando cálculos...");
    }

}
```

Inatel

Herança

Classe mãe

```java
public class Profissao {
    String nome;
    int idade;
    double salario;

    @Override
    public String toString() {
        return "idade=" + idade + ", nome="
        + nome + ", salario=" + salario;
    }
}
```

Classes filhas

```java
public class Engenheiro extends Profissao {

    public void calcular() {
        System.out.println("Calculando cálculos...");
    }

}
```

```java
public class Professor extends Profissao {

    public void corrigirProvas() {
        System.out.println("Corrigindo provas...");
    }

}
```

Inatel

Herança

E se quisermos adicionar novos atributos numa classe filha? Como a classe mãe vai acessar essa informação, como fica o toString?

Como criamos construtores para as classes filhas???

Herança

Os métodos construtores das classes filhas só irão receber um tratamento especial caso a classe mãe tenha um construtor

Classe mãe “Profissão" implementa seu construtor

```
public Profissao(String nome, int idade, double salario) {
    this.nome = nome;
    this.idade = idade;
    this.salario = salario;
}
```

```
public Engenheiro(String nome, int idade, double salario, String area) {
    super(nome, idade, salario);
    this.area = area;
}
```

Classe filha “Engenheiro” implementa seu construtor, chamando o construtor da classe mãe e adicionando seus próprios atributos depois

Inatel

Herança

```java
public class Engenheiro extends Profissao {

    public String area;

    public void calcular() {
        System.out.println("Calculando cálculos...");
    }

    @Override
    public String toString() {
        return super.toString() + ", area=" + area;
    }

}
```

“super” indica que estamos chamando um método da classe mãe, e no caso estamos apenas adicionando a nova informação no retorno dela

Inatel

Herança

Podemos também utilizar sobrescrita para mudar completamente o método da Mãe

```java
public class Engenheiro extends Profissao {

    public String area;

    public void calcular() {
        System.out.println("Calculando cálculos...");
    }

    @Override
    public String toString() {
        return "o engenheiro se demitiu :(";
    }

}
```

Polimorfismo

Polimorfismo é o terceiro dos 4 pilares de POO, ele nos concede a capacidade de um método poder se comportar de maneiras diferentes dependendo do objeto que o chama

Polimorfismo

```java
Profissao eng = new Engenheiro(nome:"Pedrinho", idade:30, salario:10000, area:"Software");
Profissao profissao = new Profissao(nome:"Joãozinho", idade:40, salario:5000);
Profissao prof = new Professor(nome:"Mariazinha", idade:35, salario:8000);

Profissao[] lista = new Profissao[3];
lista[0] = eng;
lista[1] = profissao;
lista[2] = prof;

for (Profissao p : lista) {
    System.out.println(p);
    // output:
    // idade=30, nome=Pedrinho, salario=10000.0, area=Software
    // idade=40, nome=Joãozinho, salario=5000.0
    // idade=35, nome=Mariazinha, salario=8000.0
}
```

Inatel

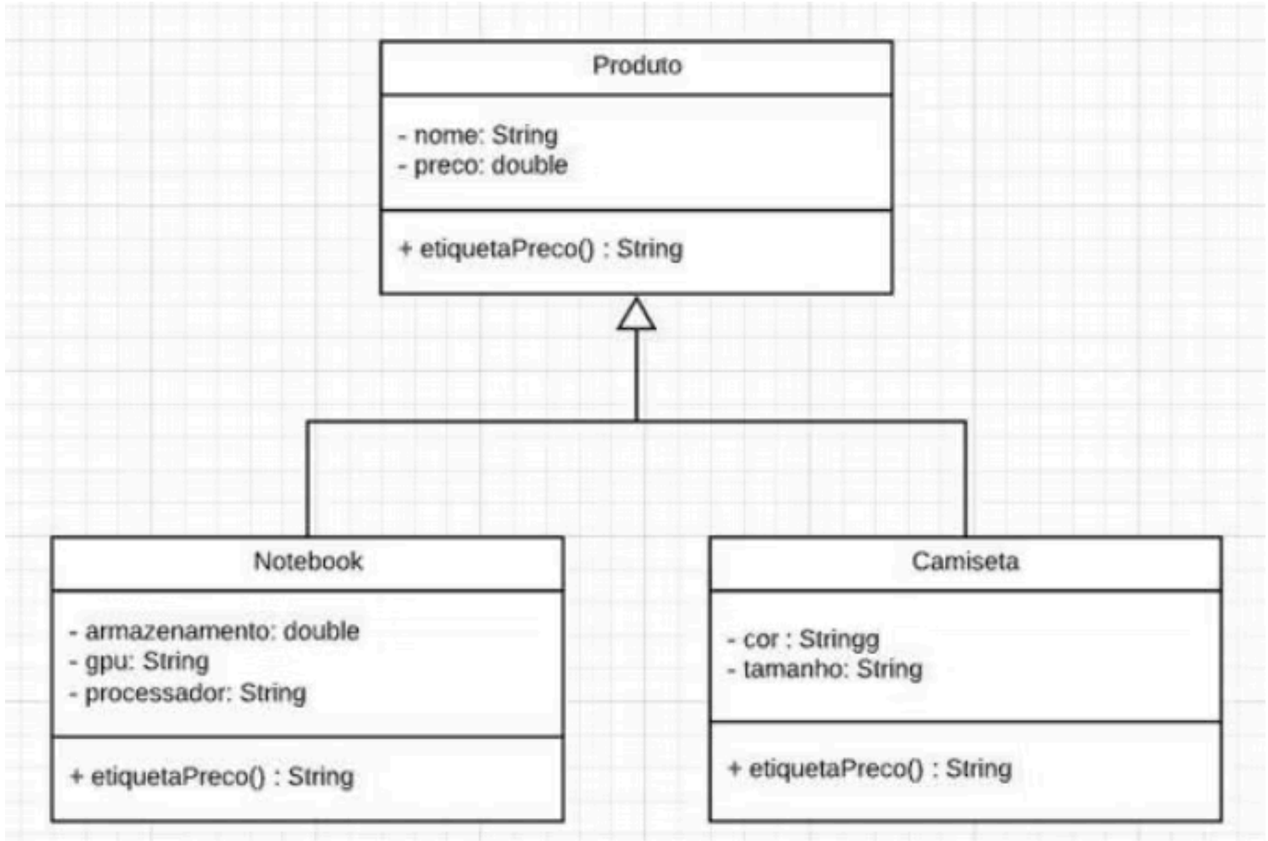
Polimorfismo

Quando utilizamos polimorfismo, é usual que usemos “instanceof” para fazer tratamentos ou lidar com a necessidade de um objeto ser de um tipo específico

```
public void mostraNome(Profissao profissao) {
    System.out.println(profissao.nome);
    if (profissao instanceof Engenheiro) {
        System.out.println(((Engenheiro) profissao).area);
    }
}
```

Exercício

- A seta indica de qual classe Mãe as classes filhas devem herdar
- **etiquetaPreco()** deve printar todas as informações e ser reescrita nas classes filhas para incluir seus atributos específicos
- Crie métodos construtores com parâmetros
- Na Main, crie um array para os produtos, faça um objeto de cada tipo, e itere pelo array verificando as instâncias e chamando os métodos para cada instância
- Crie apenas os getters e setters que forem necessários



Inatel