Herança, reescrita e polimorfismo

Capítulo VI

Pilares da Programação Orientada à Objetos

Herança em Java

```java
public class Professor {

    public String nome;
    public int idade;
    public double salario;

    public void mostraInfo(){
        System.out.println("Nome: " + nome);
        System.out.println("Idade: " + idade);
        System.out.println("Salario: " + salario);
    }

}
```

```java
public class Arquiteto{

    public String nome;
    public int idade;
    public double salario;

    public void mostraInfo(){
        System.out.println("Nome: " + nome);
        System.out.println("Idade: " + idade);
        System.out.println("Salario: " + salario);
    }

}
```

```java
public class Engenheiro{

    public String nome;
    public int idade;
    public double salario;

    public void mostraInfo(){
        System.out.println("Nome: " + nome);
        System.out.println("Idade: " + idade);
        System.out.println("Salario: " + salario);
    }

}
```

As três classes possuem os mesmos atributos e métodos. Desta forma, podemos usar da herança e criar uma classe Mãe, ou superclasse, que terá seus atributos e métodos herdados pelas classes filhas, ou subclasses

Herança em Java

Utilizaremos a classe Funcionário como classe mãe

```
public class Funcionario {

    public String nome;
    public int idade;
    public double salario;

    public void mostraInfo(){
        System.out.println("Nome: " + nome);
        System.out.println("Idade: " + idade);
        System.out.println("Salario: " + salario);
    }

}
```

As demais classes irão herdar seus atributos e métodos

Herança em Java

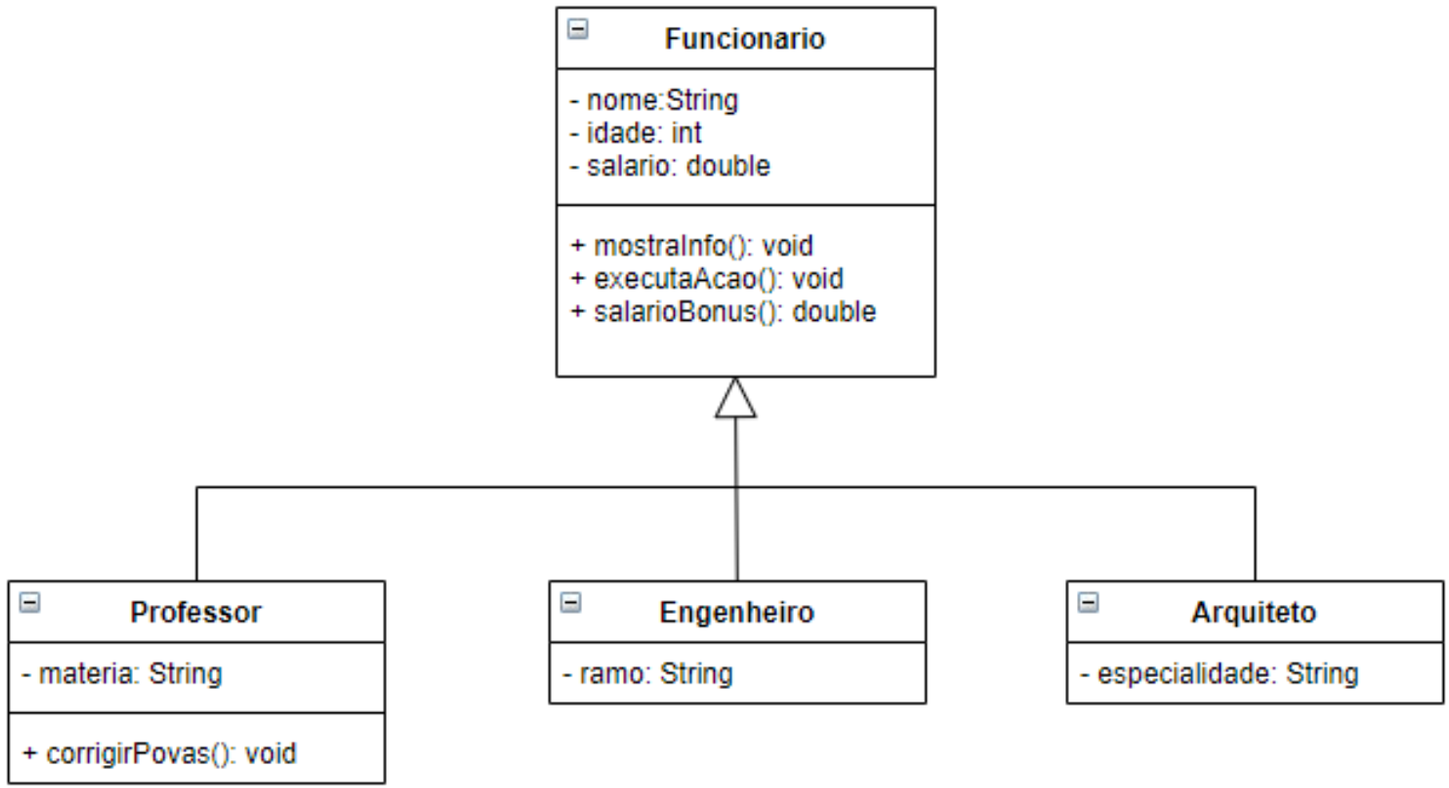
```
public class Professor extends Funcionario{


}
```

```
public class Arquiteto extends Funcionario{


}
```

```
public class Engenheiro extends Funcionario{


}
```

A palavra reservada extends permite que todos os atributos e métodos da classe mãe sejam utilizados nas classes filhas

Herança em Java



Herança em Java

```
public class Engenheiro extends Funcionario{


}
```

```
public static void main(String[] args) {

    Engenheiro eng = new Engenheiro();
    eng.nome = "João";
    eng.idade = 37;
    eng.salario = 5000;

    eng.mostraInfo();
}
```

```
Nome: João
Idade: 37
Salario: 5000.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Herança em Java

Agora os atributos de Funcionario estão privados

```java
public class Funcionario {

    private String nome;
    private int idade;
    private double salario;

    public void mostraInfo(){
        System.out.println("Nome: " + nome);
        System.out.println("Idade: " + idade);
        System.out.println("Salario: " + salario);
    }

}
```

Na main, não será possível acessá-los

```java
5
6       public static void main(String[] args) {
7
8           Engenheiro eng = new Engenheiro();
            eng.nome = "João";
            eng.idade = 37;
            eng.salario = 5000;
12
13          eng.mostraInfo();
14      }
15
16  }
```

Para corrigir este problema, podemos colocar o modificador de acesso protected nos atributos da classe mãe, ou ainda criar getters e setters na classe mãe. Assim, as classes filhas herdarão os métodos de get e set

Herança em Java

```java
public class Funcionario {

    private String nome;
    private int idade;
    private double salario;

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public int getIdade() {
        return idade;
    }

    public void setIdade(int idade) {
        this.idade = idade;
    }

    public double getSalario() {
        return salario;
    }

    public void setSalario(double salario) {
        this.salario = salario;
    }
```

```java
public static void main(String[] args) {

    Engenheiro eng = new Engenheiro();
    eng.setNome("João");
    eng.setIdade(37);
    eng.setSalario(5000);

    eng.mostraInfo();
}
```

```
Nome: João
Idade: 37
Salario: 5000.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Herança em Java

As classes filhas também podem ter seus atributos específicos, como a seguir:

```
public class Engenheiro extends Funcionario{

    private String ramo;

    public String getRamo() {
        return ramo;
    }

    public void setRamo(String ramo) {
        this.ramo = ramo;
    }
}
```

Não é necessário criar getters e setters
para idade, nome e salário,
já que foram implementados na classe mãe

```
public static void main(String[] args) {

    Engenheiro eng = new Engenheiro();
    eng.setNome("João");
    eng.setIdade(37);
    eng.setSalario(5000);
    eng.setRamo("Computação");

    System.out.println("Nome: " + eng.getNome());
    System.out.println("Idade: " + eng.getIdade());
    System.out.println("Salário: " + eng.getSalario());
    System.out.println("Ramo de atuação: " + eng.getRamo());
}
```

```
Nome: João
Idade: 37
Salário: 5000.0
Ramo de atuação: Computação
BUILD SUCCESSFUL (total time: 0 seconds)
```

Herança em Java

Agora, tentaremos utilizar o método mostraInfo() para mostrar as informações do Engenheiro

```
public static void main(String[] args) {

    Engenheiro eng = new Engenheiro();
    eng.setNome("João");
    eng.setIdade(37);
    eng.setSalario(5000);
    eng.setRamo("Computação");

    eng.mostraInfo();
}
```

```
Nome: João
Idade: 37
Salario: 5000.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Podemos perceber que apesar de termos preenchido o campo ramo do engenheiro, ele não apareceu quando fizemos a chamada do mostraInfo(). Isso aconteceu porque o mostraInfo() está implementado na classe mãe, e se quisermos mostrar atributos específicos, precisamos fazer a reescrita do método.

Herança em Java

Para fazer a reescrita do método, utilizaremos do @Override:

```
public class Engenheiro extends Funcionario{

    private String ramo;

    @Override
    public void mostraInfo(){

    }
```

Queremos mostrar todos os atributos do engenheiro, mas para isso não é necessário escrever o método por inteiro. Podemos usar da palavra reservada super para nos ajudar

```
public class Engenheiro extends Funcionario{

    private String ramo;

    @Override
    public void mostraInfo(){
        super.mostraInfo();
        System.out.println("Ramo de atuação: " + ramo);
    }
```

A palavra super faz referência à classe mãe, desta forma, estamos chamando o método da classe mãe e complementando-o

Herança em Java

Outro exemplo de reescrita:

```java
public class Funcionario {

    private String nome;
    private int idade;
    private double salario;

    public double salarioBonus(){
        return salario;
    }
```

```java
public class Engenheiro extends Funcionario{

    private String ramo;

    @Override
    public double salarioBonus(){
        return super.salarioBonus() + 700;
    }
```

```java
public static void main(String[] args) {

    Engenheiro eng = new Engenheiro();
    eng.setNome("João");
    eng.setIdade(37);
    eng.setSalario(5000);
    eng.setRamo("Computação");

    eng.mostraInfo();
    System.out.println("Salário com bônus: " + eng.salarioBonus());
}
```

```
Nome: João
Idade: 37
Salario: 5000.0
Ramo de atuação: Computação
Salário com bônus: 5700.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Herança em Java

Outro exemplo de reescrita:

```java
public class Funcionario {

    private String nome;
    private int idade;
    private double salario;

    public void executaAcao(){
        System.out.println(nome + " está fazendo algo");
    }
```

```java
public class Professor extends Funcionario{

    String materia;

    @Override
    public void executaAcao(){
        System.out.println(this.getNome() + " está dando aula");
    }
```

```java
public class Engenheiro extends Funcionario{

    private String ramo;

    @Override
    public void executaAcao(){
        System.out.println(this.getNome() + " está programando");
    }
```

Herança em Java

Outro exemplo de reescrita:

```
public static void main(String[] args) {

    Engenheiro eng = new Engenheiro();
    eng.setNome("João");
    Professor prof = new Professor();
    prof.setNome("Renzo");

    eng.executaAcao();
    prof.executaAcao();
}
```

```
João está programando
Renzo está dando aula
BUILD SUCCESSFUL (total time: 0 seconds)
```

Herança em Java

Vamos agora criar construtores em todas as classes

```java
public class Professor extends Funcionario{

    String materia;

    public Professor(String nome, int idade, double salario, String materia){
        this.setNome(nome);
        this.setIdade(idade);
        this.setSalario(salario);
        this.materia = materia;
    }
```

```java
public class Arquiteto extends Funcionario{

    String especialidade;

    public Arquiteto(String nome, int idade, double salario, String especialidade){
        this.setNome(nome);
        this.setIdade(idade);
        this.setSalario(salario);
        this.especialidade = especialidade;
    }
```

```java
public class Engenheiro extends Funcionario{

    private String ramo;

    public Engenheiro(String nome, int idade, double salario, String ramo){
        this.setNome(nome);
        this.setIdade(idade);
        this.setSalario(salario);
        this.ramo = ramo;
    }
```

Herança em Java

Na main, podemos instanciar objetos da seguinte forma:

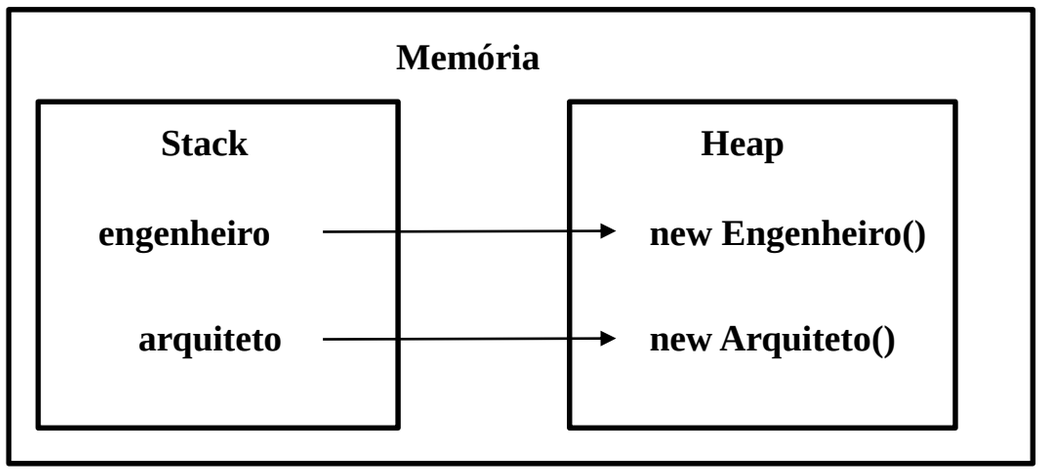
```
public static void main(String[] args) {

    Engenheiro eng = new Engenheiro("João", 37, 5000, "Computação");
    Professor prof = new Professor("Tonho", 31, 8000, "Análise de Dados");
    Arquiteto arq = new Arquiteto("Rita", 29, 4500, "Design");

    eng.mostraInfo();
    prof.mostraInfo();
    arq.mostraInfo();

}
```

```
run:
Nome: João
Idade: 37
Salario: 5000.0
Ramo de atuação: Computação
Nome: Tonho
Idade: 31
Salario: 8000.0
Matéria: Análise de Dados
Nome: Rita
Idade: 29
Salario: 4500.0
Especialidade: Design
BUILD SUCCESSFUL (total time: 0 seconds)
```

Até aqui, nenhuma novidade. Porém, podemos criar várias referências de Funcionarios com atributos diferentes. Isso é possível através do polimorfismo.

Polimorfismo:

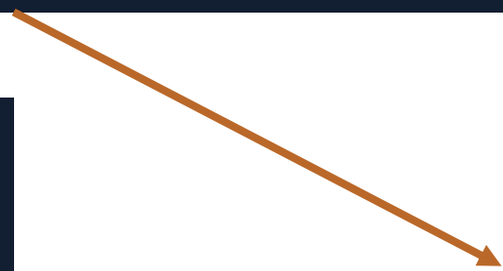
- **É um recurso que permite que variáveis de um mesmo tipo genérico possam apontar para objetos de tipo específico.**
- **Quando a instanciação do tipo específico com o tipo genérico acontece, isso é chamado de *upcasting*.**



Herança em Java

Na main, podemos instanciar objetos da seguinte forma:

```
public static void main(String[] args) {

    Funcionario eng = new Engenheiro("João", 37, 5000, "Computação");
    Funcionario prof = new Professor("Tonho", 31, 8000, "Análise de Dados");
    Funcionario arq = new Arquiteto("Rita", 29, 4500, "Design");

    eng.mostraInfo();
    prof.mostraInfo();
    arq.mostraInfo();

}
```



Upcasting

```
run:
Nome: João
Idade: 37
Salario: 5000.0
Ramo de atuação: Computação
Nome: Tonho
Idade: 31
Salario: 8000.0
Matéria: Análise de Dados
Nome: Rita
Idade: 29
Salario: 4500.0
Especialidade: Design
BUILD SUCCESSFUL (total time: 0 seconds)
```

Herança em Java

```
public static void main(String[] args) {

    Funcionario[] funcionarios = new Funcionario[5];

    Engenheiro eng = new Engenheiro("João", 37, 5000, "Computação");
    Professor prof = new Professor("Tonho", 31, 8000, "Análise de Dados");
    Arquiteto arq = new Arquiteto("Rita", 29, 4500, "Design");

    funcionarios[0] = eng;
    funcionarios[1] = prof;
    funcionarios[2] = arq;

    for (int i = 0; i < funcionarios.length; i++) {
        if(funcionarios[i] != null){
            if(funcionarios[i] instanceof Engenheiro){
                Engenheiro e = (Engenheiro)funcionarios[i];
                e.mostraInfo();
            }else if(funcionarios[i] instanceof Professor){
                Professor p = (Professor)funcionarios[i];
                p.mostraInfo();
                p.corrigirProvas();
            }else{
                Arquiteto a = (Arquiteto)funcionarios[i];
                a.mostraInfo();
            }
        }
    }
}
```

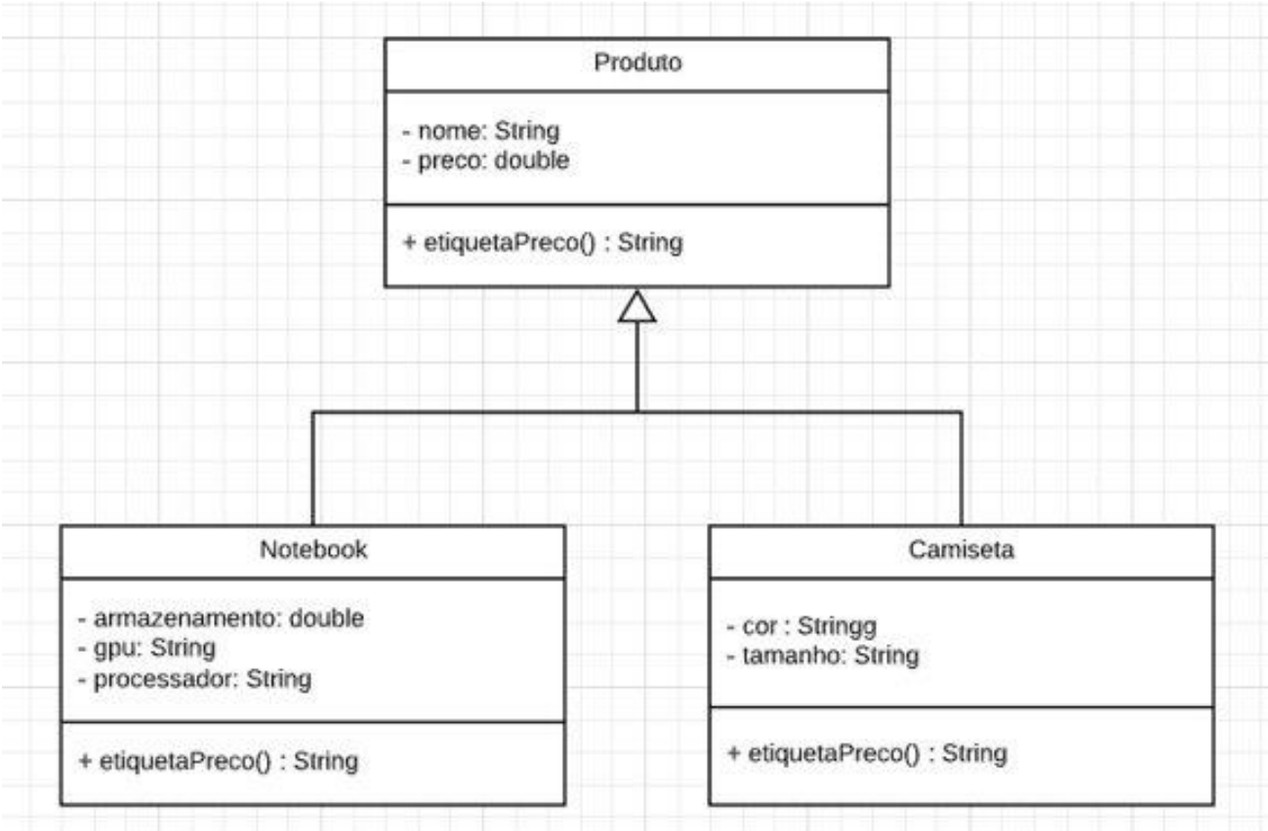
Desta forma, podemos criar um array da classe mãe e guardar objetos das classes filhas

Ao percorrer o array, verificamos a instancia de cada objeto através do instanceof. Assim, podemos acessar os atributos e métodos específicos de cada classe

Exercício

Fazer um programa para criar no mínimo 2 produtos. Ao final, mostrar a etiqueta de preço de cada produto na mesma ordem em que foram digitados.

Todo produto possui nome e preço. Notebook possui armazenamento, gpu e um processador e camiseta possui tamanho e cor. Estes dados específicos dever ser acrescentados na etiqueta de preço conforme o exemplo.



Obrigado!