Orientação a Objetos: Conceitos de Herança

Programação Aplicada - Aula 07
Prof. Sergio Bonato

fonte: Barnes & Kölling, Programação orientada a objetos com Java - Uma introdução prática usando o BlueJ, 4a edição, ed. Prentice Hall

Herança - Conceito

- É a característica da programação orientadas a objetos que permite criar uma nova classe, como extensão de outra já existente.

- Isto faz com que a nova classe herde o código-fonte da “classe-mãe”, o que proporciona a reutilização, na “classe-filha”, do código já existente na “classe-mãe”.

- Um dos impactos do uso de herança, no desenvolvimento de projeto de aplicações e sistemas orientados a objetos, é a redução do tempo empregado para desenvolver a programação, além da consequente redução da quantidade de linhas de código-fonte.

- Com tudo isso, evita-se a desnecessária duplicação de código e manutenção da aplicação torna-se mais fácil, principalmente quando a herança é combinada com a modularização, a abstração e o polimorfismo.

Herança - Generalização

- Generalização é o processo de criação de uma nova classe (“classe-mãe”), a partir de classes já criadas (“classes-filhas”), que possuam características comuns.

- Na generalização, as características comuns (atributos e métodos) das “classes-filhas” são retiradas destas e escritas na “classe-mãe”.

- A partir disso, as “classes-filhas” passam a herdar, usar e compartilhar o código-fonte da “classe mãe”.

- A consequência natural é a redução do códigofonte e a não duplicação de parte desse código.

Herança - Especialização

- Especialização é o processo inverso da generalização, ou seja, é o processo de criação de uma nova classe (“classe-filha”), a partir de uma “classe-mãe” já criada.

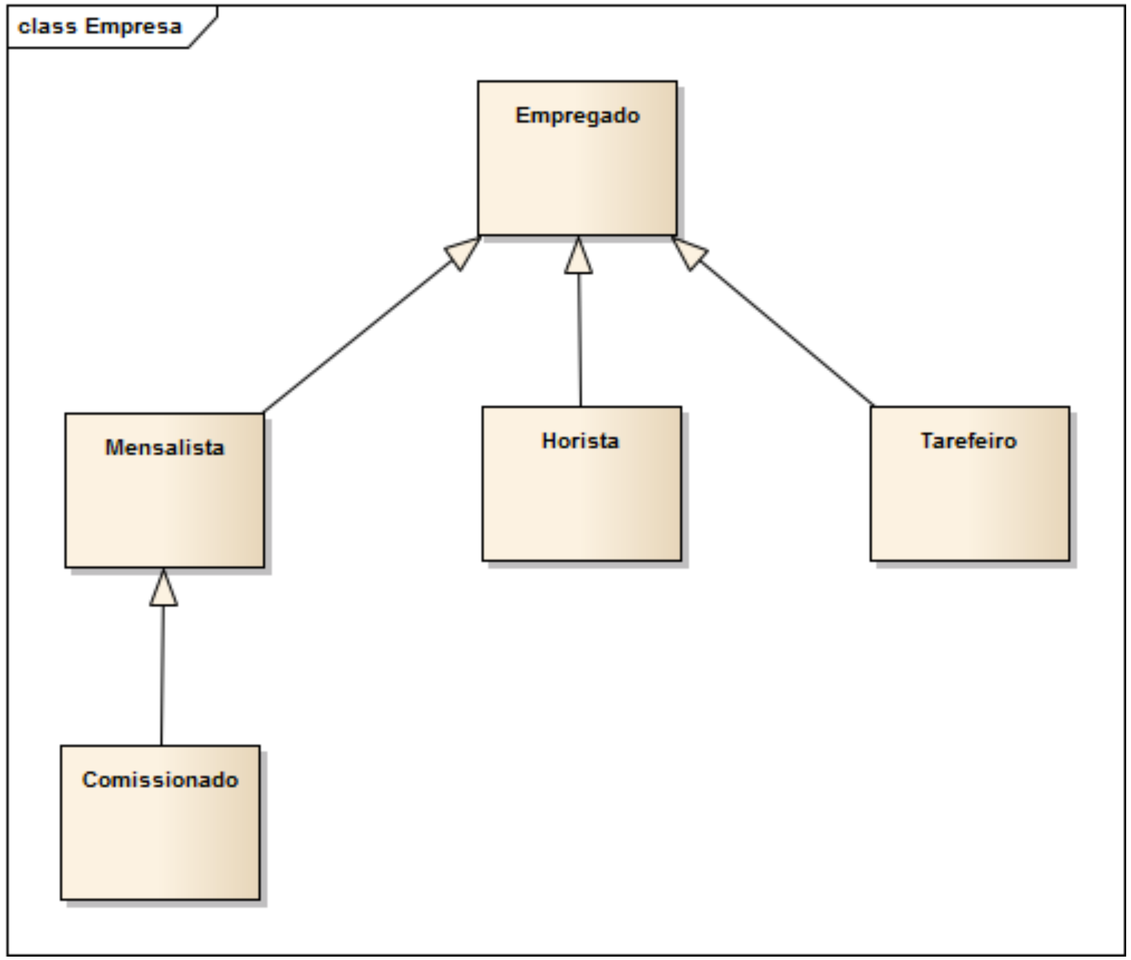
- A especialização é usada, quando se deseja que a nova “classe-filha” criada tenha, além das características herdadas da “classe-mãe “, outras características mais específicas ou mais diferenciadas.

- Um benefício consequente é o reaproveitamento do código da classe-mãe e menor tempo dispendido para desenvolver a nova classe (“classe-filha”).

Hierarquia de Classes

- Com o uso da generalização e da especialização, as “classes-mães” (superclasses) e as “classes-filhas” (subclasses) ficam organizadas em hierarquias, que ilustram e definem suas dependências e relacionamentos.

Exemplo:



Hierarquia de Classes

- O método construtor de uma subclasse sempre deve chamar o construtor da superclasse, como sua primeira instrução.
- A hierarquia de classes e a herança também permitem que um método de uma superclasse possa ser especializado ou diferenciado na subclasse (polimorfismo por **superposição** ou **sobreposição**).
- Outra forma de polimorfismo é a **sobrecarga** de método.
- Sobreposição e sobrecarga acontecem com classes diferentes da mesma hierarquia de classes, mas somente a sobrecarga ocorre com métodos da mesma classe.

Herança em Java

- Para que uma classe herde o código de outra,na linguagem Java, deve-se usar o comando extends na assinatura da classe.
- Exemplo: Supondo que exista uma classe Empregado

```
public class Mensalista extends Empregado
{
....
}
```

Encapsulamento – Identificadores de Acesso em Java

- Restringem o acesso externo os atributos e métodos de uma classe. Abaixo, estão ordenado do mais restritivo para o mais aberto:
  - **private**: atributos e métodos private não são “vistos” fora da classe a que pertencem.
  - **default**: atributos e métodos default são "vistos" dentro do mesmo pacote em que se localizam
  - **protected**: atributos e métodos protected são são “vistos” fora da classe a que pertencem, porém somente pelas classes da mesma hierarquia e/ou do mesmo pacote.
  - **public**: atributos e métodos public são “vistos” por todas as classes fora da classe a que pertencem.

```java
public class Empregado{
    private String nome;

    public Empregado(String nome){
        this.nome = nome;
    }

    public String getNome(){
        return nome;
    }

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

    public double salario(){
        return 0.0;
    }
}
```

```java
public class Mensalista extends Empregado{
    private double salario;

    public Mensalista(String nome, double salario){
        super(nome);
        this.salario = salario;
    }

    public double salario(){
        return this.salario;
    }
}
```

```java
public class Comissionado extends Mensalista{
    private double comissao;

    public Comissionado(String nome, double salario, double comissao){
        super(nome, salario);
        this.comissao = comissao;
    }

    public double salario(){
        return super.salario()+comissao;
    }
}
```

```java
public class TesteEmpregado{
   public static void main(String[] args){
      Empregado emp1 = new Empregado("Joao da Silva");
      System.out.println(emp1.getNome());
      System.out.println(emp1.salario());

      Mensalista emp2 = new Mensalista("Jose Pereira", 3500.00);
      System.out.println(emp2.getNome());
      System.out.println(emp2.salario());

      Comissionado emp3 = new Comissionado("Maria Pereira", 1500.00, 5000.00);
      System.out.println(emp3.getNome());
      System.out.println(emp3.salario());
   }

}
```

Resultado da Execução

```
 ----jGRASP exec: java TesteEmpregado
Joao da Silva
0.0
Jose Pereira
3500.0
Maria Pereira
6500.0

 ----jGRASP: operation complete.
```

- A classe Empregado é a classe base. Todos na firma são empregados, mas o empregado em si não sabe calcular o salário. Ele só sabe falar o nome dele.

- O Mensalista estende o Empregado. Ele tem um atributo a mais , o salário e agora ele sabe calcular seu salário. E, como ele é subclasse de Empregado, ele sabe falar também falar seu nome.

- O Comissionado estende o Mensalista. Portanto, ele sabe calcular seu salário e também sabe falar seu nome. Como extensão, ele sabe também calcular sua comissão. Note que o Comissionado chama o método do pai para calcular o salário ao invocar **super**.

- Aliás, as duas subclasses invocam **super** em seu construtor; Mensalista na linha 5 e Comissionado na linha 5). Com isso estão chamando o construtor de seu pai (superclasse) e passando para eles os parâmetros necessários para sua inicialização.

- Note que a chamada a **super** sempre tem que ocorrer na primeira linha do construtor. Saiba que se você não fizer a chamada explícita o compilador Java coloca uma chamada implícita ao construtor padrão da classe pai. Mas no exemplo isso não ia funcionar porque a classe Empregado não tem o construtor padrão Empregado().