Projeto e Construção de Sistemas

Módulo 4 – Herança e Polimorfismo

Lista de Exercícios

Herança e Polimorfismo

Programa Folha de Pagamento – Pessoa Física:

Define uma hierarquia de classes de regime de empregado (EmpregoEmpregadoHorista, EmpregadoAssalariado e EmpregadoBonificado), onde:

a) Todo empregado Horista tem nome, valor hora e numero de horas trabalhadas e uma taxa fixa de 100 reais que é subtraída do seu valor bruto.
b) Todo empregado Assalariado tem nome, salário base e um desconto percentual aplicado ao seu salário base.
c) Todo empregado Bonificado tem nome, salário base, um desconto percentual aplicado ao seu salário base e um bônus percentual acrescido ao valor líquido.

Todo empregado tem um valor líquido a receber que é dado pelas seguintes fórmulas:

- Horista: (valor hora * numero de horas trabalhadas) – 100
- Assalariado: salário base * (1 – desconto)
- Bonificado: (salário base * (1 – desconto)) * (1 + bônus)

Desta forma, desconto e bônus são valores reais entre 0 e 1, e.g., 10% = 0.1.
Defina a taxa fixa de desconto do Horista como uma constante.
Para cada classe, defina um construtor que receba como parâmetros todas as informações que ela possua.

Uma vez que as classes tenham sido criadas e que as fórmulas de cálculo estejam funcionando, implemente uma classe FolhaPagamento da seguinte forma:

- Defina um único array de empregados com 9 empregados, sendo 3 de cada tipo (Horista, Assalariado, Bonificado).
- Crie uma operação que imprima no console a folha de pagamento. De cada empregado imprima o nome, regime (Horista, Assalariado, Bonificado) e o respectivo pagamento líquido a receber.

Programa Folha de Pagamento – Pessoas e Empresas:

Gere uma nova versão do programa anterior, considerando o seguinte design: As classes Empregado* implementam uma interface IElementoFolha. A interface IElementoFolha estende duas interfaces: IPagamento e INome.
A interface IPagamento possui a seguinte operação:getPagamentoLiquido e getRegime.

- A operação getPagamentoLiquido retorna o pagamento líquido conforme as fórmulas de cálculo definidas anteriormente.
- A operação getTipo retorna o tipo do elemento para impressão na folha (Horista, Assalariado, Bonificado).

A interface INome possui a operação getNome.

Modifique também a classe FolhaPagamento de forma que o array não seja mais da superclasse Empregado, mas sim um array de IElementoFolha, ou seja, de objetos que implementem esta interface.

Uma vez que a nova versão do programa esteja funcionando, agora com interfaces, faça a seguinte extensão:

Crie uma classe Empresa que também implementará a interface IElementoFolha, com nome, um valor bruto, uma taxa de IR e uma taxa de ISS como descontos.
O valor líquido de uma empresa é dado pela seguinte fórmula:
Valor Bruto * (1 – taxaIR – taxaISS).
A implementação da operação getRegime de empresa deverá retornar o string "Pessoa Jurídica".

Modifique a classe FolhaPagamento de forma a imprimir uma relação de pagamentos a serem efetuados, considerando horistas, assalariados, bonificados e empresas. Para tal, inicialize o vetor de elementos do tipo IElementoFolha definido em FolhaPagamento para que tenha 3 elementos de cada tipo, ou seja, 3 horistas, 3 assalariados, 3 bonificados e 3 empresas.

Programa Recursos Ferroviários:

Este é um exercício de reestruturação de um fragmento de uma aplicação, que embora tenha sido feita em Java, está longe de ser aderente ao paradigma de orientação a objetos. Note por exemplo que os atributos foram definidos com o modificador *public* e a implementação horrível da operação getOcupacao da classe LinhaFerroviaria, com várias expressões de *downcasting*. Sua tarefa é reestruturar a implementação destas classes de forma a torná-la orientada a objetos, a partir da utilização efetiva de recursos como information hiding e polimorfismo.

Apenas para explicar o significado das classes, uma Linha Ferroviária em uma estação de manobra pode ter diversos recursos nela estacionados. Os recursos podem ser locomotivas ou vagões isolados ou um trem (formado por locomotivas e vagões). O espaço ocupado por cada recurso depende do tipo do recurso. No caso de uma locomotiva, o espaço é dado pelo seu comprimento. No caso de um vagão, o espaço é dado pela soma do comprimento dos engates mais o comprimento entre testeiras (este último corresponde ao local onde a carga efetivamente é acondicionada). Finalmente, no caso de um trem, o comprimento é dado pela soma do comprimento dos recursos que o compõem (vagões e locomotivas, calculados conforme descrito acima).

Neste exercício, você nem precisa se preocupar muito com o significado da aplicação. A idéia é que você aprenda a reconhecer padrões de estruturação ruim de uma aplicação

orientada a objetos (conhecidos também como anti-padrões) e sugerir uma solução mais adequada.

Arquivo Recurso.java

```
public abstract class Recurso {

}
```

Arquivo Locomotiva.java

```
public class Locomotiva extends Recurso {
	public	int comprimento;

	public Locomotiva(int comprimento) {
		this.comprimento = comprimento;
	}
}
```

Arquivo Vagao.java

```
public class Vagao extends Recurso {
	public	int	comprimentoTesteiras;
	public	int	comprimentoEngates;

	public	Vagao(int compTesteiras, int compEngates) {
		this.comprimentoTesteiras = compTesteiras;
		this.comprimentoEngates = compEngates;
	}
}
```

Arquivo Trem.java

```
public class Trem extends Recurso {

	public	Recurso[]	recursosDoTrem;

	public Trem(Recurso[] recursos) {
		this.recursosDoTrem = recursos;
	}
}
```

Arquivo LinhaFerroviaria.java

```
public class LinhaFerroviaria {
  public	int	numero;
  public	Recurso[] recursosEstacionados;

  public LinhaFerroviaria(Recurso[] recursos) {
      this. recursosEstacionados = recursos;
  }

  public	int	getOcupacao() {
      int  ocupacao = 0;

      for (int i = 0; i < recursosEstacionados.length; i++) {
        if (recursosEstacionados[i] instanceof Vagao)
```

```
            ocupacao = ocupacao +
               ((Vagao) recursosEstacionados[i]).comprimentoEngates +
               ((Vagao) recursosEstacionados[i]).comprimentoTesteiras;
          else if (recursosEstacionados[i] instanceof Locomotiva)
            ocupacao = ocupacao +
               ((Locomotiva) recursosEstacionados[i]).comprimento;
          else {
              Trem trem = ((Trem) recursosEstacionados[i]);
              for (int j = 0; j < trem.recursosDoTrem.length; j++) {
                 if (trem.recursosDoTrem[j] instanceof Vagao)
                    ocupacao = ocupacao +
                    ((Vagao) trem.recursosDoTrem[j]).comprimentoEngates +
                    ((Vagao) trem.recursosDoTrem[j]).comprimentoTesteiras;
                 else if (trem.recursosDoTrem[j] instanceof Locomotiva)
                    ocupacao = ocupacao +
                    ((Locomotiva) trem.recursosDoTrem[j]).comprimento;
                 }
              }
          }
          return ocupacao;
        }
}
```

Arquivo TesteLinha.java

```
public class TesteLinha {
  public static void main(String[] args) {
      Locomotiva l1 = new Locomotiva(50);
      Locomotiva l2 = new Locomotiva(30);
      Vagao[] vagoes = { new Vagao(10, 10),
                         new Vagao(10, 20),
                         new Vagao(10, 20) };
      Vagao v1 = new Vagao(20, 20);
      Recurso[] recursosTrem =
            new Recurso[] { l1, vagoes[0], vagoes[1], vagoes[2] };
      Trem trem = new Trem(recursosTrem);

      Recurso[] recursosEstacionados = new Recurso[] { l2, v1, trem };
      LinhaFerroviaria linha =
            new LinhaFerroviaria(recursosEstacionados);

      System.out.println("ocupacao da linha = " + linha.getOcupacao());
  }
}
```