# Projeto e Construção de Sistemas

## Lista de Exercícios 4

**Exercício 1:**

Parte 1:
Defina uma classe Produto que tenha:

- Atributos nome e preço
- Operações get e set para os seus atributos
- Construtor que recebe como parâmetros valores de inicialização dos seus dois atributos.

Parte 2:
Defina uma classe ProdutoEspecial derivada de Produto que tenha:
a) um atributo adicional: desconto (double) – um número entre 0 e 1.
b) operações get/set para o desconto.
c) construtor que recebe nome, preço e desconto e inicializa os respectivos atributos.
d) uma redefinição da operação getPreco de forma a aplicar o desconto percentual dado pelo atributo desconto.

Parte 3:
Faça um programa que:
a) crie um vetor com duas referencias para objetos da classe Produto.
b) Faça o primeiro elemento do vetor referenciar um objeto da classe Produto (ex: sabonete, 1.80).
c) Faça o segundo elemento do vetor referenciar um objeto da classe ProdutoEspecial (ex: cerveja, 2.00, 0.10 -> 10% desconto sobre 2.00).
d) Imprima o resultado da chamada das operações getPreco para cada produto a partir das referencias armazenadas no vetor.
e) Compare os preços dos dois produtos e imprima o resultado da comparação indicando se os preços são iguais ou o nome do produto com o maior preço, caso contrário.

**Exercício 2:**
Um programa em Java trabalha com dois tipos de sólidos: esferas e paralelepípedos.

Implemente três classes (Solido, Esfera e Paralelepipedo) da seguinte forma:

- Esfera e Paralelepipedo são classes derivadas da classe Solido;
- todo objeto Solido possui uma propriedade **densidade** do tipo **double**;
- a massa de qualquer objeto Solido pode ser obtida através da multiplicação do volume pela densidade do sólido;
- o volume de um objeto Esfera é dado por (4.0/3.0) * PI * raio$^3$, onde raio é um atributo double da Esfera.
- o volume de um objeto Paralelepipedo é dado por (alt*comp*larg), onde alt, comp e larg são atributos double do Paralelepipedo;
- o construtor de Esfera deve receber os valores de raio e densidade.
- o construtor de Paralelepipedo deve receber os valores de altura, largura, comprimento e densidade.

Faça um programa que:

a) crie um array de referencias para objetos do tipo Solido com tamanho 4.
b) Crie duas esferas e dois paralelepípedos com valores arbitrários para suas propriedades referenciando esses quatro objetos através do array criado anteriormente.
c) Crie um método que receba esse array como parâmetro e imprima o volume, o tipo (Esfera ou Paralelepipedo) e a massa de cada sólido e, ao final, o volume total e a massa total de todos os sólidos presentes no array.

## Exercício 3:

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 200 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.

Assim, 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)

Note que 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 na classe Horista.

Para cada classe, defina um construtor que receba como parâmetros todas as informações que objetos dessa classe possuam.

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 referências para empregados com 6 elementos, e crie 2 objetos de cada tipo de empregado (Horista, Assalariado, Bonificado), referenciando-os através do array criado.
- Crie uma operação que imprima no console a folha de pagamento. Essa operação deve receber um array de empregados e ser chamada pela operação main. De cada empregado imprima o nome, nome do regime (Horista, Assalariado, Bonificado) e o respectivo pagamento líquido.

## Exercício 4:

Gere uma nova versão do programa anterior, considerando a seguinte estrutura de projeto: As classes Empregado (Horista, Assalariado e Bonificado) implementam uma interface IElementoFolha. A interface IElementoFolha estende duas interfaces: IPagamento e INome.

A interface IPagamento possui as seguintes operações:

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

A interface INome possui a operação getNome que retorna um string com um nome (no caso de Empregados, será o nome do empregado).

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 2 elementos de cada tipo, ou seja, 2 horistas, 2 assalariados, 2 bonificados e 2 empresas.

## Exercício 5:

Passo 1:
Crie uma classe ContaCorrente com os seguintes atributos:
   a) nome do correntista (usando a classe string)
   b) data de abertura (classe data criada na lista anterior)
   c) saldo (double)

Passo 2:
Defina as operações get/set para os atributos da conta

Passo 3:
Defina um construtor que receba valores para os três atributos

Passo 4:
Crie um programa que permita que o usuário entre com os dados da conta (nome do correntista, data de abertura e saldo), crie um objeto conta corrente com esses dados e, em seguida, imprima o valor dos atributos desse objeto.

Passo 5:
Crie uma classe Movimentação com os seguintes atributos:
   a) data da movimentação
   b) valor da movimentação (> 0 crédito; < 0 débito)

Passo 6:
Defina as operações get/set para os atributos da Movimentação

Passo 7:
Defina um construtor que receba os valores para os dois atributos.

Passo 8:
Modifique a classe ContaCorrente para que ela passe a definir um array de referências para as suas movimentações. Para este exercício, assuma que cada classe ContaCorrente terá capacidade para armazenar, no máximo, 100 movimentações.

Passo 9:
Crie uma operação em ContaCorrente chamada registrarMovimentacao que recebe como parâmetros a data e o valor da movimentação e cria um objeto Movimentação, armazenando a referência para esse objeto Movimentação no array criado no passo anterior. Essa operação deve atualizar o saldo da conta.

Passo 10:
Crie uma operação em ContaCorrente chamada totalDebitos que não recebe parâmetro e retorna o valor total de todas as movimentações de débito registradas na conta.

Passo 11:
Crie uma operação em ContaCorrente chamada totalCreditos que não recebe parâmetro e retorna o valor total de todas as movimentações de credito registradas na conta.

Passo 12:
Faça um programa que crie um objeto ContaCorrente, registre algumas movimentações de crédito e débito e ao final imprima o saldo final da conta, o total de débitos efetuados e o total de créditos efetuados. A criação da conta e as chamadas para o registro de movimentação podem ser *hard-coded* na operação main, ao invés de entrar todos os dados pelo console.

## Exercício 6:

Faça uma nova versão do programa do exercício anterior com as seguintes alterações:

Considere que agora temos dois tipos de contas (ContaCorrente e ContaInvestimento) que são subclasses de uma classe abstrata Conta.

Atributos de Conta:

- dataAbertura (use a classe Data que já implementamos anteriormente).
- saldo
- movimentacoes: array de referências para Movimentação (todas as movimentações da conta, como no exercício anterior)

Operações de Conta:

- Construtor (que recebe dataAbertura e saldo) e cria o array de referências para movimentações (idem ao exercício anterior).
- get/set para os atributos data de abertura e saldo
- registrarMovimentacao (igual ao exercício anterior)
- totalDebitos (igual ao exercício anterior)
- totalCreditos (igual ao exercício anterior)

Atributos Específicos de ContaCorrente:

- nomeCorrentista
- array de referências para objetos da classe ContaInvestimento (representando todas as contas de investimento vinculadas à conta corrente).

Operações Específicas de ContaCorrente:

- Construtor (recebe nomeCorrentista, data de abertura e saldo inicial) que inicializa os atributos e cria um array de referências para ContaInvestimento com a capacidade de até 5 contas.
- get/set para o atributo nome do correntista
- saldoConsolidado: operação que retorna o saldo da conta corrente + somatório do saldo de todas as contas de investimento vinculadas a ela.
- vincularConta: operação que recebe como parâmetro uma ContaInvestimento e a vincula à conta corrente (armazenando a referência no array que é atributo da classe).

Atributos Específicos de ContaInvestimento:

- taxaRentabilidade : double (0.1 = 10%)
- referência para o objeto ContaCorrente ao qual ela está vinculada.

Operações Específicas de ContaInvestimento:

- Construtor (recebe a data de abertura, saldo inicial, taxa de rentabilidade e a ContaCorrente à qual ela será vinculada) que inicializa os atributos e chama a operação vincularConta do objeto ContaCorrente recebido como parâmetro
- get/set para o atributo taxa de rentabilidade.

- aplicar: operação que recebe um valor como parâmetro e registra uma movimentação de crédito neste valor na conta de investimento e uma movimentação de débito de mesmo valor na ContaCorrente vinculada.
- resgatar: operação que recebe um valor como parâmetro e registra uma movimentação de débito na conta de investimento e uma movimentação de crédito de mesmo valor na ContaCorrente vinculada.
- aplicarRentabilidade: operação que recebe como parâmetro uma data e cria uma movimentação no valor correspondente à aplicação da taxaRentabilidade ao saldoAtual da conta. A movimentação deve ser criada e adicionada ao array de ponteiros para movimentação definida na superclasse conta.
- Exemplo:
  ContaInvestimento: saldoAtual = 100.00  taxaRentabilidade = 0.1
  Após aplicarRentabilidade (10/10/2008):
  saldoAtual = 110.00 e uma nova Movimentação(10/10/2008, 10.00) foi criada e adicionada ao array de movimentações da conta investimento.

Faça um programa para testar essa classe com esses passos de exemplo:

- Criar conta corrente (c1) com saldo inicial de 200,00
- Registrar movimentação de crédito de 300,00 em c1
- Registrar movimentação de débito de 50,00 em c1
- Registrar movimentação de crédito de 230,00 em c1
- Registrar movimentação de débito de 20,00 em c1
- Imprimir o saldo atual, o total de créditos e o total de débitos da conta corrente.
- Criar uma conta de investimento (c2) vinculada à conta corrente (c1) com saldo inicial de 0.0 e taxa de rentabilidade de 0.1 (10%).
- Aplicar 100,00 em c2.
- Baixar 45,00 de c2.
- Aplicar Rentabilidade à conta c2.
- Imprimir o saldo atual, o total de créditos e o total de débitos da conta investimento (c2).
- Imprimir o saldo atual, o total de créditos e o total de débitos da conta corrente (c1).
- Imprimir o saldo consolidado da conta c1.
- Criar uma conta de investimento (c3) vinculada à conta corrente com saldo inicial de 0.0 e taxa de rentabilidade de 0.05 (5%).
- Aplicar 200,00 em c3.
- Baixar 55,00 de c3.
- Aplicar 100,00 em c3
- Aplicar Rentabilidade à conta c3.
- Imprimir o saldo atual, o total de créditos e o total de débitos da conta investimento (c3).
- Imprimir o saldo atual, o total de créditos e o total de débitos da conta corrente (c1).
- Imprimir o saldo consolidado da conta c1.

## Exercício 7:

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* (conversão de tipo, neste caso de Object para Vagão, Locomotiva e Trem). 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). Veja que todo recurso ocupa um espaço (seja Locomotiva, Vagão ou Trem). Seu desafio é criar uma solução genérica que remova os downcastings (conversão de tipo da superclasse para a subclasse) a partir do fato de que todo recurso ocupa um espaço (propriedade genérica de qualquer recurso). Note também que todos os atributos estão declarados como public, o que viola o princípio de *information hiding*.

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 Locomotiva.java**

```
public class Locomotiva {
	public  int comprimento;

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

**Arquivo Vagao.java**

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

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

**Arquivo Trem.java**

```
public class Trem {

      public  Object[]  recursosDoTrem;

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

**Arquivo LinhaFerroviaria.java**

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

  public LinhaFerroviaria(Object[] 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);

      Object[] recursosTrem =
            new Object[] { l1, vagoes[0], vagoes[1], vagoes[2] };
      Trem trem = new Trem(recursosTrem);

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

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