Fundação CECIERJ - Vice Presidência de Educação Superior a Distância

Curso de Tecnologia em Sistemas de Computação
AD1 de Programação III
1° semestre de 2015

Nome:
Pólo:

Exercício (ENTREGAR OS ARQUIVOS EM MÍDIA, PARA FINS DE TESTE, JUNTAMENTE COM A AD IMPRESSA):

Programação orientada a objetos é uma estratégia para mapear conceitos pertencentes ao domínio do problema no mundo real para um conjunto de componentes de software que seja fiel à representação deste domínio. Nesta questão, você deverá utilizar orientação a objetos para desenvolver o código Java de uma plataforma simplificada de comércio onde ocorre a interação entre vendedores e compradores, conforme as instruções abaixo:

a) **(2.5 pontos)** Tanto vendedores quanto compradores são pessoas dotadas de um nome e um endereço. A diferença entre estas entidades é que cada vendedor é responsável pela organização de uma fila de compradores. Esta fila é composta de compradores atendidos pelo vendedor que ainda não receberam a mercadoria. A organização da fila é feita a partir dos métodos propostos pela seguinte interface:

```
public interface IFluxoDeVendaEEntrega {
    void realizarVendaPara(Comprador comprador);
    void realizarEntrega();
}
```

Na prática, dadas uma instância de vendedor e outra de comprador, a realização de uma venda é feita pela chamada do método `realizarVendaPara` do vendedor, utilizando como argumento o comprador. Ainda, ao realizar a venda, o vendedor deve inserir o comprador em questão na fila de espera de entrega de mercadorias de sua responsabilidade. Realizar a entrega significa retirar o comprador da fila por meio de uma chamada ao método `realizarEntrega`.

Utilize o conceito de herança para modelar vendedores e compradores sem que haja duplicação explícita dos atributos e métodos em comum às classes.

Utilize o conceito de encapsulamento na escrita de construtores e métodos. Garanta que o nome e o endereço dessas entidades sejam fornecidos no ato de criação dos objetos. Garanta também que apenas o endereço e o conteúdo da fila de compradores possam ser alterados após a criação dos objetos.

RESPOSTA:

```
public interface IFluxoDeVendaEEntrega{
    void realizarVendaPara(Comprador comprador);
    void realizarEntrega();
}
```

```
public abstract class Pessoa{
    private String nome;
    private String endereco;

    public Pessoa(String nome, String endereco) {
        this.nome = nome;
        this.endereco = endereco;
    }

    public String obterNome() {
        return this.nome;
    }

    public String obterEndereco() {
        return this.endereco;
    }

    public void definirEndereco(String endereco) {
        this.endereco = endereco;
    }
}


public class Vendedor extends Pessoa implements IFluxoDeVendaEEntrega{
    private Fila fila = new Fila();

    public Vendedor(String nome, String endereco) {
        super(nome, endereco);
    }

    public void realizarVendaPara(Comprador comprador) {
        this.fila.inserir(comprador);
    }

    public void realizarEntrega() {
       this.fila.remover();
    }
}


public class Comprador extends Pessoa{
    public Comprador(String nome, String endereco) {
        super(nome, endereco);
    }
}
```

b) **(2.5 pontos)** Escreva uma classe para representar a fila de compradores. Esta classe deverá ser utilizada na modelagem da classe que representa o vendedor no item (a). Fica a critério do aluno o desenvolvimento de uma fila linear sequencial ou encadeada. A classe que representa a fila deve conter os métodos:

```
public void inserir(Comprador c)
public void remover()
public Comprador primeiro()
public boolean estaVazia()
```

O conceito de encapsulamento deverá ser empregado na escrita da(s) classe(s) envolvida(s) na implementação da fila.

RESPOSTA:

```
class NoFila{
    Comprador dado;
    NoFila proximo;

    NoFila(Comprador dado, NoFila proximo) {
        this.dado = dado;
        this.proximo = proximo;
    }
}


public class Fila{
    private NoFila primeiro = null;
    private NoFila ultimo = null;

    public void inserir(Comprador c) {
        NoFila novoNo = new NoFila(c, null);

        if (this.ultimo != null) {
            this.ultimo.proximo = novoNo;
        }
        else {
            this.primeiro = novoNo;
        }

        this.ultimo = novoNo;
    }

    public void remover() {
        this.primeiro = this.primeiro.proximo;

        if (this.primeiro == null) {
           this.ultimo = null;
        }
    }

    public Comprador primeiro() {
        return this.primeiro.dado;
    }

    public boolean estaVazia() {
        return (this.primeiro == null);
    }
}
```

c) **(2.0 pontos)** Vendedores e compradores são gerenciados por uma entidade que organiza o comércio. Esta entidade, chamada “mercado”, organiza duas listas independentes: a lista de vendedores e a lista de compradores cadastrados. A classe concreta que representa o mercado deve implementar a seguinte interface:

```
public interface IMercado {
        void cadastrarComprador(String nome, String
endereco);
        void cadastrarVendedor(String nome, String
endereco);
    Comprador buscarComprador(String nome);
    Vendedor buscarVendedor(String nome);
}
```

Implemente a classe que representa o mercado. Utilize o conceito de encapsulamento para garantir que vendedores e compradores presentes nas listas sejam acessíveis apenas por meio dos métodos definidos pelo mercado.

RESPOSTA:

```
public interface IMercado {
    void cadastrarComprador(String nome, String endereco);
    void cadastrarVendedor(String nome, String endereco);
    Comprador buscarComprador(String nome);
    Vendedor buscarVendedor(String nome);
}


public class Mercado implements IMercado {
    private ListaOrdenada compradores = new ListaOrdenada();
    private ListaOrdenada vendedores = new ListaOrdenada();

    public void cadastrarComprador(String nome, String endereco) {
        this.compradores.inserir(new Comprador(nome, endereco));
    }

    public void cadastrarVendedor(String nome, String endereco) {
        this.vendedores.inserir(new Vendedor(nome, endereco));
    }

    public Comprador buscarComprador(String nome) {
        return (Comprador)this.compradores.buscar(nome);
    }

    public Vendedor buscarVendedor(String nome) {
        return (Vendedor)this.vendedores.buscar(nome);
    }
}
```

d) **(3.0 pontos)** Utilize o conceito de polimorfismo para escrever uma classe de lista que possa ser aplicada tanto na organização da lista de compradores quanto na organização de lista de vendedores do mercado descrito no item (c). Fica a critério do aluno o desenvolvimento de uma lista linear sequencial ou encadeada.
A classe de lista deve conter, no mínimo, um método de inserção que garanta a inserção ordenada por nome, um método de busca por nome e um método que retorna se a lista está ou não está vazia.

Dica: Utilize o método `equals(String another)` da classe `String` para fazer a comparação entre nomes. O conceito de encapsulamento deverá ser empregado na escrita da(s) classe(s) envolvida(s) na implementação da lista.

RESPOSTA:

```
public class ListaOrdenada {
    private Pessoa[] items = new Pessoa[0];

    public void inserir(Pessoa pessoa) {
        int chave = 0;
        while ((chave < this.items.length) &&
(this.items[chave].obterNome().compareTo(pessoa.obterNome()) <= 0))
        {
            chave++;
        }

        Pessoa[] novoArray = new Pessoa[this.items.length + 1];
        System.arraycopy(this.items, 0, novoArray, 0, chave);
        novoArray[chave] = pessoa;
        System.arraycopy(this.items, chave, novoArray, chave + 1,
this.items.length - chave);
        this.items = novoArray;
    }

    public Pessoa buscar(String nome) {
        for (int i = 0; i < this.items.length; i++) {
            if (this.items[i].obterNome().equals(nome)) {
                return this.items[i];
            }
        }
        return null;
    }

    public boolean estaVazia() {
        return (this.items.length == 0);
    }
}
```