Fundação CECIERJ - Vice Presidência de Educação Superior a Distância

Curso de Tecnologia em Sistemas de Computação
Disciplina: Programação III
AP3 1° semestre de 2010.

Nome –

Assinatura –

Observações:

1. Prova sem consulta e sem uso de máquina de calcular.
2. Use caneta para preencher o seu nome e assinar nas folhas de questões e nas folhas de respostas.
3. Você pode usar lápis para responder as questões.
4. Ao final da prova devolva as folhas de questões e as de respostas.
5. Todas as respostas devem ser transcritas nas folhas de respostas. As respostas nas folhas de questões não serão corrigidas.

Questão 1) (3.0 pontos)

A matriz de Hadamard H(N), usada em projetos de programas corretores de erros, é uma matriz N por N, onde N é potência de dois, de elementos booleanos que satisfaz a seguinte propriedade: dadas duas linhas distintas i (0 <= i < N) e j (0 <= j < N) desta matriz, a quantidade de elementos distintos nestas linhas é sempre igual a N/2. Abaixo exemplica-se H(1), H(2) e H(4).

```
H(1)        H(2)        H(4)
-------------------------------
V           V V     V V V V
            V F     V F V F
                    V V F F
                    V F F V
```

Para construir H(M), onde M = 2 * N, divide-se a matriz H(M) em quatro partes iguais, chamadas de quadrantes, repete-se três vezes a matriz H(N) nos quadrantes de menores índices, e no quadrante de maiores índices de H(M), inverte-se a matriz H(N). Implemente um programa que imprima a matriz H(N) na console. A dimensão desta matriz deve ser passada como parâmetro de entrada.

RESPOSTA:

```
public class Had {
   public static void main(String[] args){
      int N = Integer.parseInt(args[0]);
```

```
      boolean[][] H = Calcula(N);
      for(int i = 0; i < N; i++){
         for(int j = 0; j < N; j++)
            if (H[i][j]) System.out.print("V ");
            else System.out.print("F ");
         System.out.println();
      }
   }

   public static boolean[][] Calcula (int m){
    boolean result[][];
    if(m == 1){
      result = new boolean[1][1];
      result[0][0] = true;
      return result;
    }
    int n = m/2, i, j;
    boolean aux[][] = Calcula (n);
    result = new boolean[m][m];
    for(i = 0; i < n; i++)
      for(j = 0; j < n; j++)
        result[i][j] = aux[i][j];

   for(i = 0; i < n; i++)
      for(j = n; j < m; j++)
        result[i][j] = aux[i][j - n];

    for(i = n; i < m; i++)
      for(j = 0; j < n; j++)
        result[i][j] = aux[i - n][j];

    for(i = n; i < m; i++)
      for(j = n; j < m; j++)
        result[i][j] = !aux[i - n][j - n];

    return result;
  }
}
```

Questão 2) (3.0 pontos)

Escreva um programa que receba como parâmetro de entrada, o nome de um arquivo texto, cujo conteúdo são números inteiros, um em cada linha, e que imprima, em outro arquivo texto, denominado saída acrescido do nome do arquivo de entrada, o total de cada um dos números que existem no arquivo de entrada, em ordem decrescente. Por exemplo, se o arquivo de entrada fosse:

```
1
2
1
2
5
1
```

A saída deste programa seria:

```
num = 1     qtde = 3
num = 2     qtde = 2
```

```
num = 5     qtde = 1
```

Um exemplo de uso desse programa seria java Total inteiros.txt, onde inteiros.txt é o nome do arquivo de entrada.

RESPOSTA:

```
import java.io.*;

class no{
  int info;
  int qtde;
  no(int info){
    this.info = info;
    qtde = 1;
  }
  public String toString(){
    String s = "num = " + info + "\t" + "qtde = " + qtde + "\n";
    return s;
  }
}

public class Total{
 public static void main(String[] args) throws IOException {
   BufferedReader in = new BufferedReader(new FileReader(args[0]));
   int n = 0;
   String s;
   try{
     while ((s = in.readLine()) != null) n++;
   }
   catch (Exception e) {
     System.out.println("Excecao1\n");
   }
   try {
      no vet[] = new no[n];
      int cont = 0, x, i;

      in.close();
      in = new BufferedReader(new FileReader(args[0]));

      while((s = in.readLine()) != null){
        x = Integer.parseInt(s);
        for(i = 0; i < cont; i++)
          if(vet[i].info == x){
            vet[i].qtde++;
            break;
          }
        if(i == cont) vet[cont++] = new no(x);
      }
       BufferedWriter      out      =      new      BufferedWriter(new
FileWriter("saida-"+args[0]));
      Ordena(vet, cont);
      for(i = 0; i < cont; i++){
        String str = vet[i].toString();
        out.write(str);
      }
      out.close();
```

```
    }
    catch (Exception e){
      System.out.println("Excecao2\n");
    }
    finally{
      in.close();
    }
  }

  public static void Ordena (no[] vet, int tam){
    int i, j, maior;
    no temp;

    for (i = 0; i < tam; i++){
       maior = i;

       for (j = i + 1; j < tam; j++)
          if (vet[j].qtde > vet [maior].qtde) maior = j;

       temp = vet[maior];
       vet[maior] = vet[i];
       vet[i] = temp;
    }
  }
}
```

Questão 3) (4.0 pontos)

Escreve uma programa para gerenciar o estoque de um depósito. A classe Item define as propriedades básicas de um item armazenado no depósito, com subclasses representando os tipos dos itens reais. Todos os itens têm um conjunto comum de propriedades (características) como identificação, descrição, quantidade, datas de armazenagem e venda. Além destas propriedades, cada item real ainda pode conter propriedades específicas. Por exemplo, se o depósito vai ser utilizado por uma empresa de eletrodomésticos, itens possíveis são: Televisor, Geladeira, Armário, Fogão, etc. Quanto aos elétricos, as características podem ser a cor e a voltagem. Quanto aos móveis, podemos armazenar o tipo de material (madeira, vidro, plástico, etc). O sistema deve permitir que os itens sejam adicionados e removidos do depósito. Defina também um método para a exibição de uma lista completa do conteúdo corrente do depósito. Cada tipo de item deve ser impresso de maneira específica. Ou seja, cada subclasse de um tipo de objeto real, que representa um tipo de item no depósito, deve conter o seu próprio método de exibição. Utilize conceitos de OO sempre que possível.

Resposta:

```
package br.cederj.comp.ano2010;

import java.awt.Color;
import java.util.ArrayList;
import java.util.GregorianCalendar;
import java.util.List;

/* Alguns pontos importantes neste gabarito:
 * - Definição de classes abstratas para Item, Item Eletrico e Movel
```

```
 * - Naturalmente, herança das classes concretas Televisor, Geladeira, etc, para estas
abstratas
 * - Definição da classe Deposito que deverá conter uma forma de armazenar itens
 * - Esta classe Deposito deve utilizar a classe item para definir o tipo da coleção, de forma a
permitir o polimorfismo
 * - Define métodos get e set para todos os campos, alem de construtores completos. Neste
ponto, para o código não ficar muito grande, poderíamos ter definidos os campos como
protected ou public.
 */

abstract class Item {
	private int codigo;
	private String descricao;
	private int quantidade;
	private GregorianCalendar armazenagem , venda;

	public Item(int codigo, String descricao, int qtd, GregorianCalendar armazenagem,
GregorianCalendar venda) {
		this.codigo = codigo;
		this.descricao = descricao;
		this.quantidade = peso;
		this.armazenagem = armazenagem;
		this.venda = venda;
	}
	public int getCodigo() {
		return codigo;
	}
	public void setCodigo(int codigo) {
		this.codigo = codigo;
	}
	public String getDescricao() {
		return descricao;
	}
	public void setDescricao(String descricao) {
		this.descricao = descricao;
	}
	public int getQuantidade() {
		return quantidade;
	}
	public void setQuantidade(int qtd) {
		this.quantidade = qtd;
	}
	public GregorianCalendar getArmazenagem() {
		return armazenagem;
	}
	public void setArmazenagem(GregorianCalendar armazenagem) {
		this.armazenagem = armazenagem;
	}
	public GregorianCalendar getVenda() {
		return venda;
	}
	public void setVenda(GregorianCalendar venda) {
		this.venda = venda;
	}
	abstract public void exibe();
}

abstract class Eletrico extends Item {
	private Color cor;
	private int voltagem;
```

```
	public Eletrico(int codigo, String descricao, int qtd, GregorianCalendar armazenagem,
GregorianCalendar venda, Color cor, int voltagem) {
		super(codigo, descricao, qtd, armazenagem, venda);
		this.cor = cor;
		this.voltagem= voltagem;
	}
	public Color getCor() {
		return cor;
	}
	public void setCor(Color cor) {
		this.cor = cor;
	}
	public int getVoltagem() {
		return voltagem;
	}
	public void setVoltagem(int voltagem) {
		this.voltagem= voltagem;
	}
}

abstract class Movel extends Item {
	private String material; // Poderia ser definido um campo enum também

	public Movel(int codigo, String descricao, int qtd, GregorianCalendar armazenagem,
GregorianCalendar venda, String material) {
		super(codigo, descricao, qtd, armazenagem, venda);
		this.material= material;
	}
	public String getMaterial() {
		return material;
	}
	public void setMaterial(String material) {
		this.material= material;
	}
}

class Televisor extends Eletrico {
	public Televisor(int codigo, String descricao, int qtd, GregorianCalendar
armazenagem, GregorianCalendar venda, Color cor, int voltagem) {
		super(codigo, descricao, qtd, armazenagem, venda, cor, voltagem);
	}
	public void exibe() {
		System.out.println("Televisor: " + this.getDescricao() + " " +
this.getVoltagem());
	}
}

class Geladeira extends Eletrico {
	public Geladeira(int codigo, String descricao, int qtd, GregorianCalendar
armazenagem, GregorianCalendar venda, Color cor, int voltagem) {
		super(codigo, descricao, qtd, armazenagem, venda, cor, voltagem);
	}
	public void exibe() {
		System.out.println("Geladeira: " + this.getDescricao() + " " + this.getPeso() +
" " + this.getVoltagem());
	}
}

class Armario extends Movel {
	public Armario(int codigo, String descricao, int qtd, GregorianCalendar
armazenagem, GregorianCalendar venda, String material) {
```

```
                super(codigo, descricao, qtd, armazenagem, venda, material);
        }
        public void exibe() {
                System.out.println("Armario: " + this.getDescricao() + this.getMaterial());
        }
}

class Fogao extends Item {
        public Fogao(int codigo, String descricao, int qtd, GregorianCalendar armazenagem,
GregorianCalendar venda) {
                super(codigo, descricao, qtd, armazenagem, venda);
        }
        public void exibe() {
                System.out.println("Fogao: " + this.getDescricao() + " " + this.getPeso());
        }
}

class Estoque {
        List <Item> itens;
        public Estoque() {
                itens= new ArrayList<Item>();
        }

        public void adicionaItem (Item i) {
                itensadd(i);
        }

        public void removeItem (int cod) {
                itensremove(buscaItem(cod));
        }

        public Item buscaItem (int cod) {
                for (Item it : itens) {
                        if (it.getCodigo() == cod)
                                return it;
                }
                return null;
        }
        public void exibeItens () {
                for (Item it : itens) {
                        it.exibe();
                }
        }
}

public class Tutoria_2010_1 {
        public static void main(String[] args) {
                Estoque dep = new Estoque();
                dep.adicionaItem(new Fogao(1, "6 bocas", 20, 20, 40, 60, new
GregorianCalendar(), new GregorianCalendar()));
                dep.adicionaItem(new Geladeira(1, "6 bocas", 20, 20, 40, 60, new
GregorianCalendar(), new GregorianCalendar(), Color.WHITE , 110));
                dep.exibeItens();
        }
}
```