Fundação CECIERJ - Vice Presidência de Educação Superior a Distância

Curso de Tecnologia em Sistemas de Computação
Disciplina: Programação III
AP2 2° semestre de 2009.

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 pontos)

Escreva um programa que leia um arquivo contendo números inteiros, um por linha, e escreva, num arquivo de saída, a primeira metade dos números lidos, mantendo a ordem de entrada do arquivo original, e a segunda metade escrita de maneira invertida. Por exemplo, se o arquivo de entrada fosse composto dos seguintes números:

```
1
2
3
4
```

O arquivo de saída seria:

```
1
2
4
3
```

Um exemplo de uso desse programa seria **`java mudaOrdem arq.txt saida.txt`**, onde **`arq.txt`** é o nome do arquivo de entrada e **`saida.txt`** é o nome do arquivo de saída.

RESPOSTA:

```
import java.io.*;

public class mudaOrdem {
 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{
       int vet[] = new int[n];
       in.close();

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

       int i = 0;
       while(i < (n/2)){
         s = in.readLine();
         vet[i++] = Integer.parseInt(s);
       }

       int j = n - 1;
       while (j >= (n/2)){
         s = in.readLine();
         vet[j--] = Integer.parseInt(s);
       }

       BufferedWriter out = new BufferedWriter(new FileWriter(args[1]));
       for(i = 0; i < n; i++) out.write(vet[i]+"\n");
       out.close();
    }
    catch (Exception e){
      System.out.println("Excecao2\n");
    }
    finally{
      in.close();
    }
  }
}
```

Questão 2) (3 pontos)

Crie um Programa que implemente e escreva o Triângulo de Pascal em um arquivo de saída. Neste Triângulo, cada elemento é igual à soma dos elementos que lhe ficam imediatamente acima, caso o elemento não seja nem o primeiro nem o último da sua linha. Senão, o elemento é um. A dimensão do Triângulo deve ser passada como parâmetro de entrada. Se a dimensão for seis, o seu programa deve reproduzir o seguinte arquivo:

```
Triangulo de Pascal de dimensao 6:
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
```

Um exemplo de uso desse programa seria **`java TriPascal 6 saida.txt`**, onde **`saida.txt`** é o nome do arquivo de saída.

RESPOSTA:

```
import java.io.*;
```

```
class TriPascal{
  public static void main(String args[]) throws IOException{
    int dimensao = Integer.parseInt(args[0]);
    BufferedWriter out = new BufferedWriter(new FileWriter(args[1]));
    int [][] matriz = new int [dimensao][dimensao];

    for(int i = 0;i < dimensao; i++)
      for(int j = 0; j <= i;j++)
        if((j==0) || (j == i))
          matriz[i][j] = 1;
        else
          matriz[i][j] = matriz[i-1][j-1] + matriz[i-1][j];

    try{
      for (int i = 0;i < dimensao; i++){
        for (int j = 0; j <= i;j++)
          out.write(matriz[i][j] + " " );
        out.write("\n");
      }
    }catch (Exception e){
      System.out.println("Excecao2\n");
    }finally{
      out.close();
    }
  }
}
```

Questão 3) (4 pontos)
Considere o código abaixo que exemplifica o uso de um sistema de vendas:

```
public class AP2_2009_2_Q3 {
      public static void main(String[] args) {
            Estoque produtos = new Estoque();
            produtos.adicionaProduto(new Produto ("monitor", 500));
            produtos.adicionaProduto(new Produto ("telefone", 150));
            produtos.adicionaProduto(new Produto ("teclado", 70));
            produtos.adicionaProduto(new Produto ("mouse", 50));

            CarrinhoCompra carrinho = new CarrinhoCompra(produtos);

            carrinho.adicionaItem("monitor", 2);
            carrinho.adicionaItem("telefone", 5);
            carrinho.adicionaItem("teclado", 2);

            System.out.println("A soma dos produtos é: " +
carrinho.calculaTotal());
      }
}
```

Apresente uma possível definição para as classes Estoque, Produto (cada produto possui um nome e seu valor) e CarrinhoCompra, as quais compõe este sistema de vendas. Observe que, na adição de um item no carrinho de compras, além do nome do produto, é fornecida a quantidade de itens a serem adquiridos. O método *calculaTotal()* retorna a soma total de produtos existentes no carrinho.

Resposta:

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

/*
 * Interface para um Produto que permite definirmos
 * que métodos são necessários no nosso sistema
 * (esta interface não foi pedida na questão)
 */
interface IProduto {
      float obtemValor();
      String obtemNome();
}

/* Classe produto contendo o valor e o nome do produto */
class Produto implements IProduto {
      private float valor;
      private String nome;
      public Produto(String n, float v) {
            nome = n;
            valor = v;
      }
      public float obtemValor() {
            return valor;
      }
      public void alteraValor (float v) {
            valor = v;
      }
      public String obtemNome() {
            return nome;
      }
      public void setNome(String nome) {
            this.nome = nome;
      }
}

/*
 * Classe que modela um item a ser inserido no carrinho.
 * Esta não foi explicitamente pedida, mas facilita a
 * criação da classe CarrinhoCompra
 * Além dos campos da classe Produto, declaramos um campo
 * quantidade, o qual define a quantidade de itens do
 * produto no carrinho
 */
class ItemCompra extends Produto {
      private int quantidade;
      public ItemCompra(String n, float v, int quant) {
            super(n, v);
            quantidade = quant;
      }
      public int obtemQuantidade () {
            return quantidade;
      }
```

```
}

/*
 * Classe que representa o estoque. Contém uma lista
 * de produtos e métodos para adição e obtenção de um
 * produto.
 * Naturalmente, para uma modelagem mais realista, deveríamos
 * criar outros métodos, como a remoção de produtos
 */
class Estoque {
	private List<IProduto> produtos;
	public Estoque() {
		produtos = new ArrayList<IProduto>();
	}
	public void adicionaProduto (IProduto prod) {
		produtos.add(prod);
	}
	public IProduto obtemProduto (String nome) {
		// Forma simplificada de percorrermos uma coleção
		// Alternativamente, poderíamos usar a interface Iterator
		for(IProduto prod : produtos) {
			if (prod.obtemNome() == nome)
				return prod;
		}
		return null;
	}
}

/*
 * Classe que modela um carrinho de compras.
 * Esta possui uma referência para o estoque sobre o qual
 * o carrinho se baseará e uma lista de itens
 */
class CarrinhoCompra {
	private List<ItemCompra> produtos;
	private Estoque estoque;
	public CarrinhoCompra(Estoque e) {
		produtos = new ArrayList<ItemCompra>();
		estoque = e;
	}
	// Estamos supondo um estoque infinito. Caso contrário, teríamos
que verificar se a quantidade desejada está disponível em estoque.
	public void adicionaItem (String nome, int quant) {
		IProduto prod = estoque.obtemProduto(nome);
		produtos.add(new ItemCompra(prod.obtemNome(),
prod.obtemValor(), quant));
	}
	public float calculaTotal () {
		float soma = 0;
		for (ItemCompra item : produtos) {
			soma = soma + (item.obtemValor() *
item.obtemQuantidade());
		}
		return soma;
	}
}
```