Fundação CECIERJ - Vice Presidência de Educação Superior a Distância

Curso de Tecnologia em Sistemas de Computação
Disciplina: Programação Orientada a Objetos
AP2 1° semestre de 2017.

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) (1.5 pontos)

Modificadores de visibilidade estão fortemente ligados ao princípio de encapsulamento. Logo, é muito importante saber de que maneira eles afetam o acesso a uma classe ou interface declarada em um pacote ou o acesso a um membro (i.e., atributo e método) declarado em uma classe ou interface. Preencha as células da tabela abaixo com "Sim" e "Não", conforme a visibilidade observada em diferentes escopos (linhas) no uso dos modificadores de a visibilidade (colunas) na declaração de um membro de uma classe. Para facilitar a correção, esta questão deve ser resolvida aqui, na própria folha de questões.

		Modificadores			
		`public`	`protected`	`private`	sem modificador
Escopo	Subclasse da que declara	**SIM**	**SIM**	**NÃO**	*
	Outras classes no mesmo pacote	**SIM**	**SIM**	**NÃO**	**SIM**
	Outras classes em outro pacote	**SIM**	**NÃO**	**NÃO**	**NÃO**

*** NÃO: se a subclasse está em outro pacote**
SIM: se a subclasse está no mesmo pacote

Questão 2) (4.5 pontos)
Observe o código apresentado abaixo:

```
import java.io.*;

public class NoSimples {
    public double valor;
    NoSimples proximo;

    public NoSimples(double valor) {
        this.valor = valor;
        this.proximo = null;
    }

    public NoSimples(double valor, NoSimples proximo) {
        this.valor = valor;
        this.proximo = proximo;
    }
}

public class Q2_AP2_2017_1 {
      public static NoSimples montarLista(File arquivo) throws
IOException {
        /* Implemente sua solução aqui! */
    }
}
```

Note que o método `montarLista` da classe `Q2_AP2_2017_1` está incompleto. Para resolver esta questão você terá que implementar este método conforme a seguinte especificação:

a) O método recebe um arquivo texto, cujo conteúdo é composto por valores ponto flutuante de precisão dupla (double) contidos sequencialmente no restante do arquivo.

b) O método deve: (i) abrir o arquivo; (ii) utilizar bufferização na leitura dos dados; (iii) construir uma lista linear simplesmente encadeada contendo os valores; e (iv) fechar o arquivo antes de retornar a referência para o primeiro nó da lista. Caso a lista esteja vazia, o método deve retornar a referência nula (null).

c) O arquivo em si pode conter valores duplicados e que não estejam necessariamente ordenados. Entretanto, a lista deve ser preenchida de modo a guardar, em ordem crescente, uma única ocorrência de cada valor.

Para resolver essa questão, fique atento às seguintes permissões e restrições:

a) Utilize subprogramação se você achar que isso ajudará na organização da solução. Nesse caso, crie os subprogramas como métodos estáticos e privados na classe `Q2_AP2_2017_1`.

b) Não podem ser utilizados arrays ou classes de coleção do pacote java.util na solução.

c) Não podem ser declarados atributos adicionais nas classes `NoSimples` e `Q2_AP2_2017_1`, nem podem ser declaradas outras classes.

d) Não é preciso tratar exceções, pois note que o método `montarLista` repassa qualquer `IOException` que venha a ocorrer. Entretanto, você deve garantir que o arquivo seja fechado antes do encerramento do método, aconteça ou não aconteça um caso de exceção.

Se o arquivo "exemplo.bin" contém os valores:

3.5	2.4	7.3	2.4

então o encadeamento final será: `2.4 → 3.5 → 7.3 → null`

RESPOSTA:

```
public class Q2_AP2_POO_2017_1 {
  //
  //Os dois métodos a seguir são exemplos de resolução da questão
  //
   public static NoSimples montarLista(File arquivo) throws IOException
{
    NoSimples primeiro = null;
    BufferedReader in;
    try{
      in = new BufferedReader(new FileReader(arquivo));
      String s = in.readLine();
      int n = Integer.parseInt(s);
      for (int i = 0; i < n; i++) {
        s = in.readLine();
        primeiro = inserir(primeiro, Double.parseDouble(s));
      }
      in.close();
    } catch (Exception e) {System.out.println("Excecao\n");}
    return primeiro;
  }

  private static NoSimples inserir(NoSimples primeiro, double valor) {
    if (primeiro == null || primeiro.valor > valor) {
      return new NoSimples(valor, primeiro);
    }
    if (primeiro.valor != valor) {
      NoSimples anterior = primeiro;
       while(anterior.proximo != null && anterior.proximo.valor < valor)
{
        anterior = anterior.proximo;
      }
       if (anterior.proximo == null || anterior.proximo.valor != valor)
{
        NoSimples novo = new NoSimples(valor, anterior.proximo);
        anterior.proximo= novo;
      }
    }
    return primeiro;
  }
```

```
  //
  //Os dois métodos a seguir são para a simples conferência da questão
  //
  public static void main(String[] args) throws Exception{
    NoSimples l = montarLista(new File(args[0]));
    imprime(l);
  }

  public static void imprime(NoSimples l){
    NoSimples p = l;
    while(p != null){
      System.out.println(p.valor);
      p = p.proximo;
    }
  }
}
```

Questão 3) (4.0 pontos)

Considere a classe principal apresentada abaixo, a qual é parte de um sistema para controle de impostos sobre imóveis:

```
1: public class AP2_2017_1_Q3 {
2:     public static void main(String[] args) {
3:           Imovel apto = new Imovel(100000);
4:           apto.adicionaImposto(new IPTU(0.03));
5:           apto.adicionaImposto(new Bombeiro());
6:           System.out.println(apto.calculaSomaImpostos());
7:     }
8: }
```

Neste código, na linha 3 é criado um imóvel com valor de 100 mil dinheiros. Na linha 4 é adicionado ao imóvel um imposto referente a taxa de IPTU. Neste caso, o valor do imposto equivale a 3% do imóvel. Na linha 5 é adicionado uma taxa de Bombeiro, a qual tem valor fixo de 100 dinheiros. Por fim, na linha 6 é calculado e impresso o valor total de impostos devido do imóvel.

Apresente uma possível implementação para Imovel, IPTU e Bombeiro. Crie qualquer outra classe ou interface necessária para que o código acima funcione.

Dica: Observe atentamente o código para definir nomes de métodos, argumentos, tipos de retorno, e qualquer outra informação necessária para definir os tipos acima.

RESPOSTA:

```
// Classe utilitárias necessárias para podermos
//trabalhar com listas
```

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

// Interface necessária para permitir que objetos do tipo IPTU e Bombeiro possam ser
adicionados numa mesma lista
interface Imposto {
	double calculaValor(Imovel b);
}

// Classe pedida na questão.
class IPTU implements Imposto {
	double taxa;

	public IPTU (double tx) {
		this.taxa = tx;
	}

	public double calculaValor(Imovel b) {
		return b.getValor() * this.taxa;
	}
}

// Classe pedida na questão.
class Bombeiro implements Imposto {
	public double calculaValor(Imovel b) {
		return 100;
	}
}

// Classe requerida na questão.
class Imovel {
	private double valor;
	private List<Imposto> impostos;

	public Imovel (double valor) {
		this.valor = valor;
		impostos = new ArrayList<Imposto>();
	}

	public double getValor() {
		return this.valor;
	}

	public void adicionaImposto(Imposto i) {
		impostos.add(i);
	}

	// Neste método, a chamada a calculaValor é polimórfica,
	//uma vez que qualquer objeto do subtipo de Imposto pode
	//ser adicionado à lista
	public double calculaSomaImpostos() {
		double temp = 0;
		for (Imposto i : impostos) {
			temp = temp + i.calculaValor(this);
		}
		return temp;
	}
}
```