Fundação CECIERJ - **Vice Presidência de Educação Superior a Distância**

Curso de Tecnologia em Sistemas de Computação
Disciplina: Programação III
AP2 1° semestre de 2014.

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) (4.0 pontos)

Matrizes são ferramentas matemáticas utilizadas, por exemplo, para a resolução de sistemas lineares e aplicação de transformações lineares. Uma matriz é uma tabela $m \times n$ de símbolos a_{ij} sobre um conjunto (e.g., números reais), onde m é o número de linhas e n o número de colunas, para $1 \leq i \leq m$ e $1 \leq j \leq n$:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

Matrizes podem ser classificadas quanto a sua forma e/ou conteúdo. Por exemplo:

Matriz linha é uma matriz de ordem $1 \times n$.

Matriz coluna é uma matriz de ordem $m \times 1$.

Matriz diagonal é uma matriz de ordem $n \times n$ onde $a_{ij} = 0$ para $i \neq j$.

Matriz identidade é um caso especial de matriz diagonal onde $a_{ij} = 1$ para $i = j$.

Matriz nula é a matriz de ordem $m \times n$ onde todos os elementos são iguais a zero ($a_{ij} = 0$)

$$L=\begin{bmatrix} a_{11} & \cdots & a_{1n} \end{bmatrix} \qquad C=\begin{bmatrix} a_{11} \\ \vdots \\ a_{m1} \end{bmatrix} \qquad D=\begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix} \qquad I=\begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \qquad N=\begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

Matriz Linha Matriz Coluna Matriz Diagonal Matriz Identidade Matriz Nula

(a) (1 ponto) Declare as classes `Matriz`, `MatrizLinha`, `MatrizColuna`, `MatrizDiagonal`, `MatrizIdentidade` e `MatrizNula` de modo que elas estejam relacionadas por herança.

(b) (1 ponto) Cada classe declarada deverá conter um único construtor de inicialização que recebe o tamanho da matriz. As classes `MatrizLinha`, `MatrizColuna` e `MatrizDiagonal` devem declarar o construtor com um único argumento. As demais classes devem declarar o construtor com dois argumentos.

(c) (2 pontos) Todas as classes devem conter os métodos:

```
double obter(int lin, int col)
void atribuir(int lin, int col, double valor)
int numeroDeLinhas()
int numeroDeColunas()
```

RESPOSTA:

```
abstract class Matriz {
   private int nLinhas;
   private int nColunas;

   public Matriz(int nLinhas, int nColunas) {
      if ((nLinhas <= 0) || (nColunas <= 0)) {
          throw new IllegalArgumentException("Tamanho inválido para a
   matriz.");
      }
      this.nLinhas = nLinhas;
      this.nColunas = nColunas;
   }

   public abstract double obter(int lin, int col);

   public abstract void atribuir(int lin, int col, double val);

   public int numeroDeLinhas() { return this.nLinhas; }
```

```
    public int numeroDeColunas() { return this.nColunas; }


    protected void validarIndices(int lin, int col) {
          if ((lin < 0) || (this.nLinhas <= lin) || (col < 0) || (this.nColunas <=
    col))
         throw new IllegalArgumentException("Indexação de célula fora
    dos limites permitidos.");
    }
}


class MatrizLinha extends Matriz {
    private double[] elems;

    public MatrizLinha(int nColunas) {
        super(1, nColunas);
          elems = new double[nColunas];
    }

    public double obter(int lin, int col) {
        validarIndices(lin, col);
        return this.elems[col];
    }

    public void atribuir(int lin, int col, double valor) {
        validarIndices(lin, col);
        this.elems[col] = valor;
    }
}


class MatrizColuna extends Matriz {
    private double[] elems;

    public MatrizColuna(int nLinhas) {
        super(nLinhas, 1);
        elems = new double[nLinhas];
    }
```

```
    public double obter(int lin, int col) {
        validarIndices(lin, col);
        return this.elems[lin];
    }

    public void atribuir(int lin, int col, double valor) {
        validarIndices(lin, col);
        this.elems[lin] = valor;
    }
}


class MatrizDiagonal extends Matriz{
    private double[] elems;

    public MatrizDiagonal(int tamanho) {
        super(tamanho, tamanho);
        this.elems = new double[tamanho];
    }

    public double obter(int lin, int col) {
        validarIndices(lin, col);
        double resp = (lin == col) ? this.elems[lin] : 0.0;
        return resp;
    }

    public void atribuir(int lin, int col, double valor) {
         validarIndices(lin, col);
        if (((lin != col) && (valor != 0)) || ((lin == col) && (valor == 0)))
             throw new IllegalArgumentException("Os coeficientes que não
    estão na diagonal da matriz não podem ser alterados.");
        else if(lin == col) this.elems[lin] = valor;
    }
}


class MatrizIdentidade extends MatrizDiagonal {
    public MatrizIdentidade(int tamanho) {
            super(tamanho);
    }
```

```
    public double obter(int lin, int col) {
        validarIndices(lin, col);
        return (lin == col) ? 1.0 : 0.0;
    }

    public void atribuir(int lin, int col, double valor) {
         validarIndices(lin, col);
        if (((lin == col) && (valor != 1)) || ((lin != col) && (valor != 0)))
            throw new IllegalArgumentException("Os coeficientes de uma
    matriz identidade não podem ser alterados.");
    }
}


class MatrizNula extends Matriz{
    public MatrizNula(int nLinhas, int nColunas) {
        super(nLinhas, nColunas);
    }
    public double obter(int lin, int col) {
        validarIndices(lin, col);
        return 0.0;
    }

    public void atribuir(int lin, int col, double valor) {
          validarIndices(lin, col);
                if(valor != 0) throw new IllegalArgumentException("Os
    coeficientes de uma matriz nula não podem ser alterados.");
    }
}
```

Questão 2) (2.0 pontos)

Considerando que você realizou a implementação de maneira correta da Questão 1, implemente a classe de teste com os seguintes passos:

1. Ler de um arquivo, passado como parâmetro para a main, o tamanho da matriz $m \times n$ a ser criada.
2. Descobrir, na main, qual é o tipo de matriz usada, analisando o tamanho da matriz e alguns de seus elementos.
3. Criar, na main, uma instância de matriz com o tamanho informado e ler os coeficientes.
4. Chamar o método estático `imprimir na main` para escrever na tela a matriz criada. Esse método aplica polimorfismo para receber como argumento a referência a uma instância de qualquer classe declarada (isto é, além da main, você deve implementar esse método estático).

Esse método estático deve imprimir as matrizes como nos exemplos mostrados na Questão 1: a cada linha de coeficientes impressos, seu método deve pular de linha.

O arquivo de entrada possui o seguinte formato:

m //**uma dimensão**

n //**outra dimensão**

i j a_{ij} //**dar para todos os elementos, sua posição e seu coef.**

RESPOSTA:

```
public class Q2_AP2_2014_1 {
   public static void main(String[] args) throws Exception{
  BufferedReader in = new BufferedReader(new FileReader(args[0]));
      Matriz m = null;
      try{
       int l, c, i , j, k, total;
       double elem;
       l = Integer.parseInt(in.readLine());
       c = Integer.parseInt(in.readLine());
       if((l != c) && (l == 1)) m = new MatrizLinha(c);
       else if ((l != c) && (c == 1)) m = new MatrizColuna(l);
       else if(l != c) m = new MatrizNula(l, c);
       String s, vs[];
       s = in.readLine();
       vs = s.split(" ");
       i = Integer.parseInt(vs[0]) - 1;
       j = Integer.parseInt(vs[1]) - 1;
       elem = Double.parseDouble(vs[2]);

       if((l == c) && (elem == 0)) m = new MatrizNula(l, c);
       else if (l == c) m = new MatrizDiagonal(l);
       m.atribuir(i, j, elem);

          total = l * c - 1;
       for(k = 0; k < total; k++){
         s = in.readLine();
         vs = s.split(" ");
         i = Integer.parseInt(vs[0]) - 1;
         j = Integer.parseInt(vs[1]) - 1;
         elem = Double.parseDouble(vs[2]);
         m.atribuir(i, j, elem);
       }
```

```
            imprimir(m);
        } catch (Exception e){
            System.out.println(e);
        } finally{
            in.close();
        }
    }

    private static void imprimir(Matriz m) {
        for (int lin = 0; lin < m.numeroDeLinhas(); lin++) {
            for (int col = 0; col < m.numeroDeColunas(); col++)
                System.out.print(m.obter(lin, col) + " ");
            System.out.println();
        }
    }
}
```

Questão 3) (3.0 pontos)

Considere o código abaixo que representa um valor de IPTU (Imposto Predial e Territorial Urbano) a ser pago por um imóvel.

```
class IPTU implements Imposto {
	double taxa;

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

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

a) Defina **Imposto** e **Imovel**, utilizados na definição desta classe. Para **Imovel**, saiba que esta classe representa um imóvel, a qual possui uma lista de impostos (objetos do tipo Imposto) referentes a este imóvel. **Imposto** pode ser deduzido do próprio código.

b) Adicione um novo tipo de imposto para a taxa de bombeiros. Considere que o valor cobrado por este é fixo e vale 200.

c) Num método main(), crie um imóvel com valor de 100000 e adicione 2 impostos a este: i) IPTU com taxa de 3% (0.03), e; ii) uma taxa de bombeiro. Imprima o valor do cálculo do imposto para este imóvel.

RESPOSTA:

```
// Classe utilitárias necessárias para podermos
//trabalhar com listas
import java.util.ArrayList;
import java.util.List;

// Interface deduzida pelo fato de termos a definição
//class IPTU implements Imposto. Ou seja, Imposto
//precisa ser uma interface
// Solução de parte do item a)
interface Imposto {
	double calculaValor(Imovel b);
}

// Classe fornecida 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 pelo item b) da questão.
class Bombeiro implements Imposto {
	public double calculaValor(Imovel b) {
		return 200;
	}
}

// Classe requerida no item a) da questão.
//Apesar da questão só informar explicitamente que imóvel
//possui uma lista impostos, o campo valor também é necessário
//uma vez que o método getValor() da classe dada na questão
//precisa funcionar. Ou seja, a classe Imovel precisar
//armazenar o valor de alguma maneira.
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 calculaImposto() {
		double temp = 0;
		for (Imposto i : impostos) {
			temp = temp + i.calculaValor(this);
		}
		return temp;
	}
}

// Manipulações objetos pedidos no item c) desta questão
public class AP2_2014_1_Q3 {
	public static void main(String[] args) {
		Imovel apto = new Imovel(100000);
		apto.adicionaImposto(new IPTU(0.03));
		apto.adicionaImposto(new Bombeiro());
		System.out.println(apto.calculaImposto());
	}
}
```