Дисциплина: *СД и ООП* **Тема:** *Основни концепции на ООП - продължение*
Специалност: *КН 1 курс, 3 и 4 групи* **Дата**: *10.03.2010 г.*

Задача 1: Да се реализират програмни средства за работа с различни видове матрици:

Решение:

1.Операции

public int rows() //Връща броя на редовете на текущата матрица
public int columns() //Връща броя на стълбовете на текущата матрица
public double elementAt(int i,int j) //Връща i,j-ия елемент на текущата матрица
public double setElementAt(int i,int j,double value) //Променя i,j-ия елемент на текущата матрица на value и // връща старата стойност
public Matrix add(Matrix B) //Връща матрицата-сума на текущата матрица и матрицата B
public Matrix mult(Matrix B) //Връща матрицата-произведение на текущата матрица и матрицата B
public Matrix mult(double value) //Връща матрица, която е произведение на value и текущата матрица
public Matrix transpose() //Връща транспонираната матрица на текущата матрица
public boolean equals(Object obj) //Проверява дали текущата матрица и матрицата obj, съвпадат

2.Представяне на елементите a_{ij} на матрица A(nxm), i∈[1;n], j∈[1;m] чрез масив:

2.1. Матрица (правоъгълна, триъгълна, разредена)

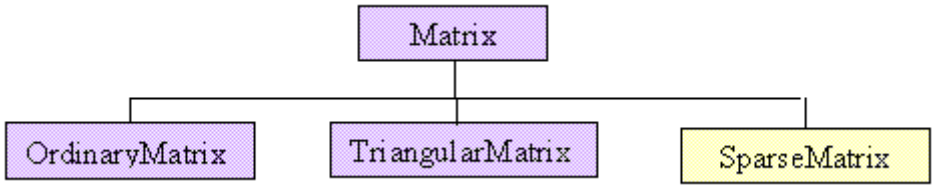
2.1.1. Масив double[][] elements=new double[n][m], като elements[i-1][j-1]=a_{ij}

2.1.2. Масив double[] elements=new double[n*m], като elements=($a_{11},\ldots,a_{1m},a_{21},\ldots,a_{2m},\ldots,a_{n1},\ldots,a_{nm}$) и elements[(i-1)*m+j-1]=a_{ij}

2.2. Триъгълна матрица (n=m): масив double[] elements=new double[(1+n)*n)/2], като elements=($a_{11},a_{21},a_{22},\ldots,a_{n1},\ldots,a_{nn}$) и elements[(i*(i-1))/2+j-1]=a_{ij}

2.3. Разредена матрица: масив elements, като elements={(i,j,a_{ij}) | $a_{ij}\neq 0$}

Реализация: Йерархия от класове:



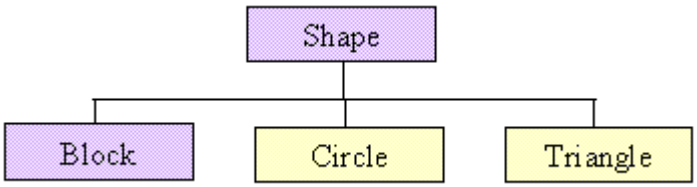

```
public abstract class Matrix {
        //Data - number of rows and columns
        protected int n;  // number of rows
        protected int m;  // number of columns

        //Abstract methods
        public abstract double elementAt(int i,int j);
        public abstract double setElementAt(int i,int j,double value);
        protected abstract Matrix create(int rows,int cols);

        //Protected method
        protected int f(int i,int j) {
                if (i < 1 || i > n || j < 1 || j > m)  throw new IndexOutOfBoundsException("Incorrect indexes!");
                return -1;
        }

        //Public methods
        public int getRows(){ return n; }

        public int getColumns(){ return m; }
```

```
        public Matrix add(Matrix B) {
                Matrix A = this;
                if (B.n != A.n || B.m != A.m) throw new RuntimeException("Illegal matrix dimensions!");
                Matrix C = create(n,m);
                for (int i = 1; i <= n; i++)
                        for (int j = 1; j <= m; j++)
                                C.setElementAt(i,j,A.elementAt(i,j) + B.elementAt(i,j));
                return C;
        }
    }

public class OrdinaryMatrix extends Matrix {
        //Data-elements
        private double[][] elements;

        //Constructors
        public OrdinaryMatrix(int rows,int cols) {
                n=rows;
                m=cols;
                elements=new double[n][m];
        }

        public OrdinaryMatrix(double[][] data) {
                this(data.length,data[0].length);
                for (int i = 0; i < data.length; i++)
                        for (int j = 0; j < data[0].length; j++)
                                elements[i][j]=data[i][j];
        }

        public OrdinaryMatrix(Matrix arg) {
                this(arg.getRows(),arg.getColumns());
                for (int i = 1; i <= arg.getRows(); i++)
                        for (int j = 1; j <= arg.getColumns(); j++)
                                elements[i-1][j-1]=arg.elementAt(i,j);
        }

        //Protected method f - inherited from Matrix

        //Implementation of abstract methods
        public double elementAt(int i,int j) {
                f(i,j);
                return elements[i-1][j-1];
        }

        public double setElementAt(int i,int j,double value) {
                f(i,j);
                double oldValue=elements[i-1][j-1];
                elements[i-1][j-1]=value;
                return oldValue;
        }

        protected Matrix create(int rows,int cols) {
                return new OrdinaryMatrix(rows,cols);
        }

        //Public methods - inherited from Matrix
}

public class TriangularMatrix extends Matrix {
        //Data-elements
        private double[] elements;

        //Constructors
        public TriangularMatrix(int rows,int cols) {
```

```
                if(rows != cols) throw new RuntimeException("Illegal matrix dimensions!");
                n=m=rows;
                elements=new double[((1+n)*n)/2];
        }

        public TriangularMatrix(double[][] data) {
                this(data.length,data[0].length);
                for (int i = 0; i < n; i++)
                        for (int j = 0; j < n; j++)
                                setElementAt(i+1,j+1,data[i][j]);
        }

        public TriangularMatrix(Matrix arg) {
                this(arg.getRows(),arg.getColumns());
                for (int i = 1; i <= n; i++)
                        for (int j = 1; j <= n; j++)
                                setElementAt(i,j,arg.elementAt(i,j));
        }

        //Override
        protected int f(int i,int j) {
                super.f(i,j);
                return (i*(i-1))/2+j-1;
        }

        //Implementation of abstract methods
        public double elementAt(int i,int j) {
                int k=f(i,j);
                double result=elements[k];
                return j<=i ? result : 0;
        }

        public double setElementAt(int i,int j,double value) {
                int k=f(i,j);
                double oldValue=elements[k];
                if(j <= i) elements[k]=value;
                else if(value!=0) throw new RuntimeException("Incorrect value!");
                return oldValue;
        }

        protected Matrix create(int rows,int cols) {
                return new TriangularMatrix(rows,cols);
        }

        //Override
        public Matrix add(Matrix B) {
                if(B instanceof TriangularMatrix) return plus((TriangularMatrix)B);
                Matrix A=new OrdinaryMatrix(this);
                return A.add(B);
        }

        //Private method
        private TriangularMatrix plus(TriangularMatrix B) {
                Matrix A = this;
                if (B.n != A.n) throw new RuntimeException("Illegal matrix dimensions!");
                TriangularMatrix C = new TriangularMatrix(n);
                for (int i = 1; i <= n; i++)
                        for (int j = 1; j <= i; j++)
                                C.setElementAt(i,j,A.elementAt(i,j) + B.elementAt(i,j));
                return C;
        }
}

public class TestMatrix {
        public static void show(Matrix A) {
```

```
                for (int i = 1; i <= A.getRows(); i++)
                        for (int j = 1; j <= A.getColumns(); j++)
                                System.out.println("El["+i+","+j+"] = "+A.elementAt(i,j));
        }

        public static void main(String[] args) {
                double[][] d = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
                Matrix M = new OrdinaryMatrix(d);
                System.out.println("     OrdinaryMatrix M:"); show(M);

                double[][] c = { { 9, 0, 0}, { 6, 5, 0 }, { 3, 2, 1 } };
                Matrix T = new TriangularMatrix(c);
                System.out.println("     TriangularMatrix T:"); show(T);

                System.out.println("     Matrix M+M  :"); show(M.add(M));

                System.out.println("     Matrix M+T  :"); show(M.add(T));

                System.out.println("     Matrix T+T:"); show(T.add(T));

                System.out.println("     Matrix T+M:"); show(T.add(M));
        }
}
```

Задача 2: Да се реализира йерархията от класове за представяне на геометрични фигури в равнината – окръжност, блок, триъгълник:




```
public class Point {
        //Data
        private double x,y;

        //Constructor
        public Point(double x,double y) { this.x=x; this.y=y; }

        //Public methods
        public double getX() { return x; }

        public double getY() { return y; }

        public double dest(Point p) { return Math.sqrt(Math.pow(x-p.x,2)+Math.pow(y-p.y,2)); }

        public Object clone() {return new Point(x,y); }

        public boolean equals(Object obj) {
                Point p=(Point)obj;
                return p.x==x && p.y==y;
        }

        public String toString() { return "Point("+x+","+y+")"; }
}

public abstract class Shape {
        public abstract double area();
        public abstract double p();
        public abstract boolean isMember(Point p);
}

public class Block extends Shape {
```

```
	//Data
	private Point d1;     //top  left corner
	private Point d2;    //down right corner

	//Constructors
	public Block(double x1,double y1,double x2,double y2) throws RuntimeException {
		if(x1==x2 || y1==y2) throw new RuntimeException("Invalid data!");
		d1=new Point(x1,y1);
		d2=new Point(x2,y2);
	}

	public Block(Point d1,Point d2) throws RuntimeException {
		this(d1.getX(),d1.getY(),d2.getX(),d2.getY());
	}

	//Public methods
	public double area() {
		Point m=new Point(d1.getX(),d2.getY());
		return m.dest(d1)*m.dest(d2);
	}

	public double p() {
		Point m=new Point(d1.getX(),d2.getY());
		return 2*(m.dest(d1)+m.dest(d2));
	}

	public boolean isMember(Point p) {
		return d1.getX()<=p.getX() && p.getX()<=d2.getX() &&
			d2.getY()<=p.getY() && p.getY()<=d1.getY();
	}

	public Object clone() {
		return new Block((Point)(d1.clone()),(Point)(d2.clone()));
	}

	public boolean equals(Object obj) {
		Block b=(Block)obj;
		return d1.equals(b.d1) && d2.equals(b.d2);
	}

	public String toString() { return "Block="+d1.toString()+","+d2.toString(); }
}

public class TestShape {
	public static void main(String[] args) {
		Circle a=new Circle(new Point(0,0),5);
		System.out.println(a.toString());
		System.out.println("Area="+a.area()+"\nP="+a.p());
		Point p=new Point(2,2);
		System.out.print(p.toString());
		System.out.println((a.isMember(p)?" is in":" is not in")+" circle!");
		System.out.println();

		Block b=new Block(new Point(1,1),new Point(10,15));
		System.out.println(b.toString());
		System.out.println("Area="+b.area()+"\nP="+b.p());
		System.out.print(p.toString());
		System.out.println((b.isMember(p)?" is in":" is not in")+" block!");
		System.out.println();

		Shape[] array={ a, b,(Circle)a.clone() };
		System.out.println("Array of shapes:");
		for(Shape s: array) System.out.println(s.toString());
	}
}
```

Задачи за упражнение:

1.Да се реализират и тестват останалите операции над матрици.

2.В класа **TriangularMatrix** е предефиниран наследеният от класа **Matrix** метод add. Какво извежда програмата **TestMatrix**, след всяка от корекциите на класа **TriangularMatrix**:

–Изтриване на методите plus и add

–Добавяне на метода

```
public Matrix add(Matrix B) {
        Matrix A = new OrdinaryMatrix(this);
        return A.add(B);
}
```