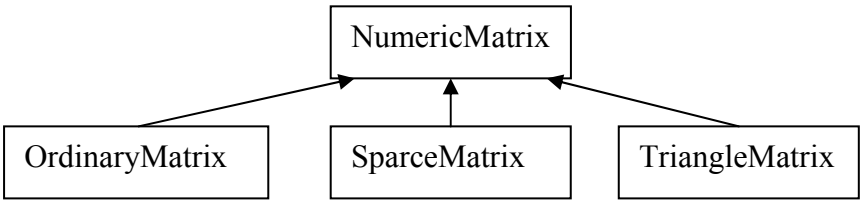
Структури от Данни и Обектно-Ориентирано Програмиране
спец. Компютърни Науки
Упражнение №9
28.04.2009г.

ТЕМА: Списъци

Задача: Да се реализира следната йерархия от класове за представяне на числови матрици, като класът *NumericMatrix* е абстрактен. Елементите на матриците са тип *double*. Трите реални класа се различават по вида на матриците – правоъгълни или триъгълни или разредени, което се отразява в различното вътрешно представяне. Целта е двуаргументните операции с матрици, като събиране и умножение, да се извършват успешно независимо от вида на двата аргумента. Ако аргументите са от един вид, то резултатът е от същия вид, а ако са от различни видове – то резултатът е правоъгълна матрица.




```
public abstract class NumericMatrix {

        protected int rows;
        protected int columns;

        abstract double elementAt(int i, int j) throws IllegalArgumentException;

        abstract void setElement(int i,int j,double val) throws IllegalArgumentException;

        public abstract NumericMatrix copyMatrix();

        boolean canAdd(NumericMatrix mat) {
          return this.rows==mat.rows && this.columns==mat.columns;
        }

        boolean canMult(NumericMatrix mat) {
          return this.columns==mat.rows;
        }

        public NumericMatrix() {}

        public NumericMatrix addTo(NumericMatrix mat) throws
                        IllegalArgumentException {
```

```
                System.out.println("Here I am in Numeric!");
                if (!canAdd(mat)) throw new
                                IllegalArgumentException("Can't do addition!");
                NumericMatrix result=new OrdinaryMatrix(rows, columns);
                for (int i=0; i<rows; i++)

                  for (int j=0; j<columns; j++)
                    try {
                      result.setElement(i,j, this.elementAt(i,j)+mat.elementAt(i,j));
                    }
                    catch(IllegalArgumentException e) {
                    System.out.println(e.toString());
                }
                return result;
        }

        public NumericMatrix multWith(NumericMatrix mat) throws
                                IllegalArgumentException {
                throw new UnsupportedOperationException();
        }

        public void printMatrix() {
                for(int i=0; i<rows; i++) {
                        try {
                                for(int j=0; j<columns-1; j++)
                                        System.out.print(elementAt(i, j) + " , ");
                                System.out.println(elementAt(i, columns-1));
                        }
                        catch(IllegalArgumentException e) {}
                }
        }
}

import java.util.LinkedList;
import java.util.Iterator;

public class SparceMatrix extends NumericMatrix {

        private static class Triple implements Comparable<Triple> {
                int i, j;
                double element;

                Triple(int i, int j, double e) {
                        this.i = i;
                        this.j = j;
                        element = e;
```

```
        }

        public int compareTo(Triple other) {
                if(this.i == other.i && this.j == other.j)
                        return 0;
                else if(this.i < other.i || (this.i == other.i && this.j < other.j))
                        return -1;
                return 1;
        }

        public String toString() {
                return "(" + i + ", " + j + ", " + element + ")";
        }
}

private LinkedList<Triple> body;

public SparceMatrix(int rows, int columns) {
        this.rows = rows;
        this.columns = columns;
        body = new LinkedList<Triple>();
}


public SparceMatrix(double [][] array) {
        rows = array.length;
        columns = array[0].length;
        body = new LinkedList<Triple>();
        for(int i=0; i<array[0].length; i++)
                for(int j=0; j<array[0].length; j++)
                        if(array[i][j] != 0)
                                body.add(new Triple(i, j, array[i][j]));
}

public double elementAt(int i, int j) throws OutOfMatrixRange {
        if (i<0 || i>=rows || j<0 || j>=columns)
           throw new OutOfMatrixRange();
        Triple tmp = new Triple(i, j, 1);
        Triple tmp1;
        Iterator<Triple> it = body.iterator();
        while(it.hasNext()) {
                if((tmp1 = it.next()).compareTo(tmp) == 0)
                        return tmp1.element;
        }
        return 0;
}
```

```
public void setElement(int i, int j, double val) throws OutOfMatrixRange {
        if (i<0 || i>=rows || j<0 || j>=columns)
           throw new OutOfMatrixRange();
        Triple tmp = new Triple(i, j, val); // new element
        Triple tmp1 = null;
        Iterator<Triple> it = body.iterator();
        int index = 0;
        while(it.hasNext() && (tmp1 = it.next()).compareTo(tmp) == -1)
                index++;
        if(tmp.compareTo(tmp1) == 0) // element found, value changed
                tmp1.element = val;
        else                                    // new element inserted
                body.add(index, tmp);
}

public NumericMatrix clone() {
        SparceMatrix result = new SparceMatrix(rows, columns);
        result.body = (LinkedList<Triple>)this.body.clone();
        return result;
}

public SparceMatrix addTo(SparceMatrix mat) throws CannotAdd {
        if (!canAdd(mat)) throw new CannotAdd();
        SparceMatrix result;
        if(this.body.isEmpty())
                result = (SparceMatrix)mat.clone();
        else if(mat.body.isEmpty())
                result=(SparceMatrix)this.clone();
        else {
                result = new SparceMatrix(rows, columns);
                Iterator<Triple> it = this.body.iterator();
                Iterator<Triple> it1 = mat.body.iterator();
                Triple itNext, it1Next;
                itNext = it1Next = null;
                if(it.hasNext())   // initial values
                        itNext = it.next();
                if(it1.hasNext())  // initial values
                        it1Next = it1.next();
                while(itNext != null && it1Next != null)
                        if(itNext.compareTo(it1Next) == -1){
                                result.body.add(itNext);
                                itNext = it.hasNext() ? it.next() : null;
                        }
                        else if(itNext.compareTo(it1Next) == 0) {
                                itNext = new Triple(itNext.i, itNext.j,
                                        itNext.element+it1Next.element);
```

```
                                result.body.add(itNext);
                                itNext = it.hasNext() ? it.next() : null;
                                it1Next = it1.hasNext() ? it1.next() : null;
                        }
                        else {
                                result.body.add(it1Next);
                                it1Next = it1.hasNext() ? it1.next() : null;
                        }
                if(itNext == null)
                        while(it1Next != null){
                                result.body.add(it1Next);
                                it1Next = it1.hasNext() ? it1.next() : null;
                        }
                else
                        while(itNext != null){
                                result.body.add(itNext);
                                itNext = it.hasNext() ? it.next() : null;
                        }
        }
        return result;
    }

    public NumericMatrix multWith(TriangleMatrix mat) throws CannotMult {
        throw new UnsupportedOperationException();
    }
}
```