Weslyn Wagner EE 5343 UTSA: Java Assignment #2 9/20/2016

* **Flowcharts for Matrix.java:**

1. **MatAdd**

Take two 3x3 matrices of integers as inputs

Int[][] matA(3x3)

Int[][] matB(3x3)

For every int[] array inside the matA and matB matrix

For every int inside those int[] arrays

**Add** both ints in that position and store in a new int[][] array called matC

Return int[][] matC

1. **MatMult**

Take two 3x3 matrices of integers as inputs

Int[][] matA(3x3)

Int[][] matB(3x3)

For every int[] array inside the matA and matB matrix

For every int inside those int[] arrays

**Multiply** both ints in that position and store in a new int[] array

Return int[][] matC

1. **MatInv**

Int[][] matA(3x3)

Take one 3x3 matrix of integers as input

Calculate the matrix of minors for every int in matA by ignoring same row and column and finding ad-bc for remaining values

Create a new int[][] array called matCoFact and set equal to matMinor

Store these values in a new int[][] array called matMinor

Multiply every other value in int[][] matCoFact by -1

Find the determinant (needs to be double) by multiplying the value in matA[0][j] by the corresponding value in matMinor[0][j]

Create a new int[][] array called matTr and set every [i][j] entry equal to matCoFact[j][i]

Create a new double[][] array called matFinal and store the product of (1/det) and matTr[i][j] in the corresponding position of matFinal

Return double matFinal[][]

Int[][] matA(3x3)

Take one 3x3 matrix of integers/doubles as input

For every int[]/double[] array inside the matA matrix

**Print** that int/double with “ “ after

For every int/double inside that int[]/double[] array

**Println** “” to start a new line

1. **MatShow**

* **Explanations:**

f) Overloading the methods for this assignment could be useful to make the program more robust. For instance, in the code below, there are two different “show” methods—one for inputs of type int[][] and one for type double[][]. This is necessary for displaying the two different return types of the other methods in the program. The code below is also built to handle exact 3x3 matrices as inputs and to return one 3x3 matrix. Overloading could be useful so that larger or smaller matrix inputs could be used or returned. This way, the appropriate method would be called based on input size without throwing errors, making this program much more robust.

* **Code:**

/\*

\* September 20, 2016

\*Version 1.0

\*EE 5343 UTSA: Java Assignment #2 Parts a-e

\*Student: Weslyn Wagner

\*/

import java.util.Arrays;

import java.math.\*;

import java.lang.\*;

class Matrix{

public static void main(String[] args){

int[][] matA= {{1,2,3},{6,5,7},{4,8,9}};

int[][] matB= {{0,1,4},{2,3,8},{0,2,3}};

System.out.println("a)Calling matShow on the matAdd method");

matShow(matAdd(matA,matB));

System.out.println("b)Calling matShow on the matMult method");

matShow(matMult(matA,matB));

System.out.println("c)Calling matShow on the matInv method result");

matShow(matInv(matA));

System.out.println("d)Calling matShow on matA and matB");

matShow(matA);

matShow(matB);

}

public static int[][] matAdd(int[][] matA, int[][] matB){

int[][] matC= {{0,0,0},{0,0,0},{0,0,0}};

int num;

for(int i=0;i<matA.length;i++){

for(int j=0;j<matA[i].length;j++){

num = matA[i][j]+matB[i][j];

matC[i][j] = num;

}

}

return matC;

}

public static int[][] matMult(int[][] matA, int[][] matB){

int[][] matC= {{0,0,0},{0,0,0},{0,0,0}};

int num;

for(int i=0;i<matA.length;i++){

for(int j=0;j<matA[i].length;j++){

num = matA[i][j]\*matB[i][j];

matC[i][j] = num;

}

}

return matC;

}

public static double[][] matInv(int[][] matA){

int[][] matMinor= {{0,0,0},{0,0,0},{0,0,0}};

matMinor[0][0]= matA[1][1]\*matA[2][2]-matA[1][2]\*matA[2][1];

matMinor[0][1]= matA[1][0]\*matA[2][2]-matA[1][2]\*matA[2][0];

matMinor[0][2]= matA[1][0]\*matA[2][1]-matA[1][1]\*matA[2][0];

matMinor[1][0]= matA[0][1]\*matA[2][2]-matA[0][2]\*matA[2][1];

matMinor[1][1]= matA[0][0]\*matA[2][2]-matA[0][2]\*matA[2][0];

matMinor[1][2]= matA[0][0]\*matA[2][1]-matA[0][1]\*matA[2][0];

matMinor[2][0]= matA[0][1]\*matA[1][2]-matA[0][2]\*matA[1][1];

matMinor[2][1]= matA[0][0]\*matA[1][2]-matA[0][2]\*matA[1][0];

matMinor[2][2]= matA[0][0]\*matA[1][1]-matA[0][1]\*matA[1][0];

int[][] matCoFact = matMinor;

matCoFact[0][1]= matMinor[0][1]\*-1;

matCoFact[1][0]= matMinor[1][0]\*-1;

matCoFact[1][2]= matMinor[1][2]\*-1;

matCoFact[2][1]= matMinor[2][1]\*-1;

int[][] matTr= {{0,0,0},{0,0,0},{0,0,0}};

for(int i=0;i<matCoFact[0].length;i++){

for(int j=0;j<matCoFact.length;j++){

matTr[i][j] = matCoFact[j][i];

}

}

double det;

det = matA[0][0]\*matMinor[0][0]- matA[0][1]\*matMinor[0][1] + matA[0][2]\*matMinor[0][2];

double[][] matFinal={{0.0,0.0,0.0},{0.0,0.0,0.0},{0.0,0.0,0.0}};

for(int i=0;i<matFinal.length;i++){

for(int j=0;j<matFinal[i].length;j++){

matFinal[i][j]= (double)matTr[i][j];

}

}

for(int i=0;i<matFinal.length;i++){

for(int j=0;j<matFinal[i].length;j++){

matFinal[i][j]= (1/det)\*(double)matTr[i][j];

}

}

return matFinal;

}

public static void matShow(int[][] matA){

System.out.println("Matrix in grid form");

for(int i=0;i<matA.length;i++){

for(int j=0;j<matA[i].length;j++){

String value = Integer.toString(matA[i][j]);

if(value.length()==1){

System.out.print(value+ " ");

}

else if(value.length()==2){

System.out.print(value+ " ");

}

else{

System.out.print(value+ " ");

}

}

System.out.println("");

}

}

public static void matShow(double[][] matA){

System.out.println("Matrix in grid form");

for(int i=0;i<matA.length;i++){

for(int j=0;j<matA[i].length;j++){

String value = Double.toString(matA[i][j]);

int index = value.indexOf(".");

if(value.length()<=3){

System.out.print(" "+value+ " ");

}

else if(value.length()==4){

System.out.print(" "+value+ " ");

}

else{

System.out.print(value.substring(0,index+4) + " ");

}

}

System.out.println("");

}

}

}

* **Screenshot of results:**

