

## TestMatrix Class

```
/**  
 * Used to test the static methods of class Matrix.  
 *  
 * @author (Antonios Symvonis)  
 * @version (18 April 2002)  
 */  
public class TestMatrix  
{  
  
    public static void main(String args[])  
    {  
        double m1[][]={{ {1,2,3,4},  
                      {5,6,7,8},  
                      {9,10,11,12} } ;  
  
        double m2[][]={{ {1,1,1},  
                      {1,1,1},  
                      {1,1,1},  
                      {1,1,1} } ;  
  
        double m3[][]=new double[3][4];  
        double I4[][]=new double[4][4];  
  
        for(int i=0; i<4; i++)  
            I4[i][i]=1;  
  
        System.out.println("-----");  
  
        //test print(), fillRandomData(), clone()  
        System.out.println("----TESTING print, fillRandomData, clone----");  
  
        System.out.println("matrix m1=");  
        Matrix.print(m1);  
        System.out.println("matrix m2=");  
        Matrix.print(m2);  
        System.out.println("matrix I4=");  
        Matrix.print(I4);  
  
        Matrix.fillRandomData(m3);  
        System.out.println("matrix m3= fillRandomData(m3)=");  
        Matrix.print(m3);  
  
        double r1[][]=Matrix.clone(m1);  
        System.out.println("matrix r1= Matrix.clone(m1)=");  
        Matrix.print(r1);  
  
        //test add(), subtract(), scalar Multiply()  
        //System.out.println("----TESTING add, subtract, scalar multiply----");  
        //double r2[][]=Matrix.add(m1,m3);  
        //System.out.println("matrix r2= Matrix.add(m1,m3)=");  
        //Matrix.print(r2);  
  
        //double r3[][]=Matrix.subtract(m1,m3);  
        //System.out.println("matrix r3= Matrix.subtract(m1,m3)=");  
        //Matrix.print(r3);  
    }  
}
```

```

//          double r4[ ][ ]=Matrix.multiply(2,m1);
//          System.out.println("matrix r4= Matrix.multiply(2,m1)=");
//          Matrix.print(r4);

//          //test matrix-multiply
//          System.out.println("----TESTING matrix multiply----");
//          //

//          double lin1[][]={{1,2,3,4}};
//          double lin2[][]={{ 1},
//                           { 2},
//                           { 3},
//                           { 4} };

//          System.out.println("data used:");
//          System.out.println("matrix m1=");
//          Matrix.print(m1);
//          System.out.println("matrix m2=");
//          Matrix.print(m2);
//          System.out.println("matrix lin1=");
//          Matrix.print(lin1);
//          System.out.println("matrix lin2=");
//          Matrix.print(lin2);

//          double r5[][]=Matrix.multiply(m1,m2);
//          System.out.println("matrix r5= Matrix.multiply(m1,m2)=");
//          Matrix.print(r5);

//          double r6[][]=Matrix.multiply(lin1, m2);
//          System.out.println("matrix r6= Matrix.multiply(lin1, m2)=");
//          Matrix.print(r6);

//          double r7[][]=Matrix.multiply(lin1, lin2);
//          System.out.println("matrix r7= Matrix.multiply(lin1, lin2)=");
//          Matrix.print(r7);

//          //test maxRows, maxCols, minRows, minCols, sumRows, sumCols
//          System.out.println("----TESTING maxRows, maxCols, minRows, minCols---");
//          //

//          System.out.println("data used:");
//          System.out.println("matrix m3=");
//          Matrix.print(m3);

//          double r8[][]=Matrix.maxRows(m3);
//          System.out.println("matrix r8= Matrix.maxRows(m3)=");
//          Matrix.print(r8);

//          double r9[][]=Matrix.maxCols(m3);
//          System.out.println("matrix r9= Matrix.maxCols(m3)=");
//          Matrix.print(r9);

//          double r10[][]=Matrix.minRows(m3);
//          System.out.println("matrix r10= Matrix.minRows(m3)=");
//          Matrix.print(r10);

//          double r11[][]=Matrix.minCols(m3);
//          System.out.println("matrix r11= Matrix.minCols(m3)=");
//          Matrix.print(r11);

```

```

//      double r12[][]=Matrix.sumRows(m3);
//      System.out.println("matrix r12= Matrix.sumRows(m3)=");
//      Matrix.print(r12);

//      double r13[][]=Matrix.sumCols(m3);
//      System.out.println("matrix r13= Matrix.sumCols(m3)=");
//      Matrix.print(r13);

}//main

}//TestMatrix

```

**Matrix Class**

```

import java.util.Random;
/**
 * Class Matrix provides several static operation
 * on 2-dimensional arrays of double. The methods of class
 * Matrix can operate correctly on 2 dimensional arrays which
 * are rectangular (all rows are of equal length) and
 * non-empty (not null and minimum size double[1][1]).
 *
 * @author (Antonios Symvonis)
 * @version (18 April 2002)
 */
public class Matrix
{
    /**
     * Prints the elements of a 2d-array, each row at a different line.
     *
     * @param m The array of which the elements are printed.
     *           Must be rectangular and non-empty.
     */
    public static void print(double m[][])
    {
        int rows=m.length;
        int cols=m[0].length;

        for( int i=0; i<rows; i++){
            for (int j=0; j<cols; j++)
                System.out.print(m[i][j] + "   ");
            System.out.println();
        }
        System.out.println();
    }//print

    /**
     * Creates an exact copy of a 2d-array.
     *
     * @param m The array to be copied. Must be rectangular and non-empty.
     * @return An exact copy of the argument array.
     */
    public static double[][] clone(double m[][])
    {
        int rows=m.length;
        double result[][]=new double[rows][];

        for(int i=0; i<rows; i++){
            int cols=m[i].length;
            result[i] = new double[cols];

```

```

        for(int j=0; j<cols; j++)
            result[i][j]=m[i][j];
    }
    return result;
}//clone

/**
 * Fills a matrix with random data in the range [0..1). The random
 * numbers have precision of 2 decimal digits.
 *
 * @param m      The array to be filled. Must be rectangular and non-
empty.
 */
public static void fillRandomData(double m[][])
{
    Random random= new Random(0);
    int rows=m.length;
    int cols=m[0].length;

    for (int i=0; i<rows; i++)
        for (int j=0; j<cols; j++)
            m[i][j]=(double) random.nextInt(100)/100;
}//fillRandomData

/**
 * Computes the sum of two matrices of equal dimensions.
 *
 * @param m1      First matrix to be added
 * @param m2      Second matrix to be added
 */
public static double[][] add(double m1[][], double m2[][])
{
    int rows=m1.length;
    int cols=m1[0].length;
    double m[][] = new double[rows][cols];

    for (int i=0; i<rows; i++)
        for (int j=0; j<cols; j++)
            m[i][j]=m1[i][j] + m2[i][j];

    return m;
}//add

/**
 * Computes the difference of two matrices of equal dimensions.
 *
 * @param m1      Minuend matrix
 * @param m2      Subtrahend matrix
 */
public static double[][] subtract(double m1[][], double m2[][])
{
    int rows=m1.length;
    int cols=m1[0].length;
    double m[][] = new double[rows][cols];

    for (int i=0; i<rows; i++)
        for (int j=0; j<cols; j++)
            m[i][j]=m1[i][j] - m2[i][j];

    return m;
}

```

```

} //subtract

/**
 * Calculates the product of a matrix with a real number
 *
 * @param r          Real number
 * @param m1         Matrix to be multiplied
 */
public static double[][] multiply(double r, double m1[][])
{
    int rows=m1.length;
    int cols=m1[0].length;
    double m[][] = new double[rows][cols];

    for (int i=0; i<rows; i++)
        for (int j=0; j<cols; j++)
            m[i][j]=r * m1[i][j];

    return m;
} //multiply

/**
 * Calculates the internal product of two matrices
 *
 * @param m1
 * @param m2
 */
public static double[] multiply(double m1[], double m2[])
{
    int rows=m1.length;
    int cols=m2.length;
    int common=m1.length; // Could be common=m2.length
    double m[] = new double[rows*cols];

    for (int i=0; i<rows; i++)
        for (int j=0; j<cols; j++) {
            m[i*cols+j]=0;
            for (int k=0; k < common; k++)
                m[i*cols+j] += m1[i*k] * m2[k];
        }

    return m;
} //multiply

/**
 * Find the maximum of each row within the passed matrix.
 * Note: The dimension of the return matrix is 1 x rows.
 *
 * @param m         Matrix to search
 */
public static double[][] maxRows(double m[][])
{
    int rows=m.length;
    int cols=m[0].length;
    double max[][] = new double[rows][1];

    for (int i=0; i<rows; i++) {
        max[i][0]=m[i][0];
        for (int j=1; j<cols; j++)
            if (m[i][j] > max[i][0])
                max[i][0] = m[i][j];
    }
}

```

```

    }

    return max;
}//maxRows

/**
 * Find the minimum of each row within the passed matrix.
 * Note: The dimension of the return matrix is 1 x rows.
 *
 * @param m          Matrix to search
 */
public static double[][] minRows(double m[][])
{
    int rows=m.length;
    int cols=m[0].length;
    double min[][] = new double[rows][1];

    for (int i=0; i<rows; i++) {
        min[i][0]=m[i][0];
        for (int j=1; j<cols; j++)
            if (m[i][j] < min[i][0])
                min[i][0] = m[i][j];
    }

    return min;
}//minRows

/**
 * Find the maximum of each column within the passed matrix.
 * Note: The dimension of the return matrix is cols x 1.
 *
 * @param m          Matrix to search
 */
public static double[][] maxCols(double m[][])
{
    int rows=m.length;
    int cols=m[0].length;
    double max[][] = new double[1][cols];

    for (int j=0; j<cols; j++) {
        max[0][j]=m[0][j];
        for (int i=1; i<rows; i++)
            if (m[i][j] > max[0][j])
                max[0][j] = m[i][j];
    }

    return max;
}//maxCols

/**
 * Find the minimum of each column within the passed matrix.
 * Note: The dimension of the return matrix is cols x 1.
 *
 * @param m          Matrix to search
 */
public static double[][] minCols(double m[][])
{
    int rows=m.length;
    int cols=m[0].length;
    double min[][] = new double[1][cols];

```

```
        for (int j=0; j<cols; j++) {
            min[0][j]=m[0][j];
            for (int i=1; i<rows; i++)
                if (m[i][j] > min[0][j])
                    min[0][j] = m[i][j];
        }

        return min;
} //minCols

/**
 * Calculates the sum of each row of the passed matrix
 * Note: The dimension of the return matrix is 1 x rows.
 *
 * @param m          Matrix to search
 */
public static double[][] sumRows(double m[][])
{
    int rows=m.length;
    int cols=m[0].length;
    double sum[][] = new double[rows][1];

    for (int i=0; i<rows; i++) {
        sum[i][0]=m[i][0];
        for (int j=1; j<cols; j++)
            sum[i][0] += m[i][j];
    }

    return sum;
} //sumRows

/**
 * Calculates the sum of each column of the passed matrix
 * Note: The dimension of the return matrix is cols x 1.
 *
 * @param m          Matrix to search
 */
public static double[][] sumCols(double m[][])
{
    int rows=m.length;
    int cols=m[0].length;
    double sum[][] = new double[1][cols];

    for (int j=0; j<cols; j++) {
        sum[0][j]=m[0][j];
        for (int i=1; i<rows; i++)
            sum[0][j] += m[i][j];
    }

    return sum;
} //sumCols

} //Matrix
```