Assignment: Write a simple Matrix class. Represent matrix as a 2-dimensional array of doubles. The Matrix class should include:

- get methods for all private members;
- assign method to change the content of particular cell;
- methods for addition and multiplication of matrices;
- Matrix gauss(Matrix m) method that solves a system of linear equations in a matrix form using Gaussian elimination;
- any other additional members if needed.

Solution:

```
package matrix;

public class Matrix {
    private static final double EPS = 1e-12;
    private int rows, cols;
    private double a[][];

    // Returns number of rows
    public int getRows()
    {
        return rows;
    }

    // Returns number of columns
    public int getCols()
    {
        return cols;
    }

    // Returns a[i][j]. Attention! No error-returning value is here, only
    // error message!
    public double getValue(int i, int j)
    {
        if (i < 0 || i >= rows || j < 0 || j >= cols) {
            System.out.println(i+", "+j);
            System.out.println("Error in Matrix.getValue method! Arguments
are out of range!");
        }
        return a[i][j];
    }

    // Constructor with parameters: size
    public Matrix(int rows, int cols) // creates a matrix of size
i_size*i_size
    {
        int i;
```

```
        this.rows = rows;
        this.cols = cols;
        a = new double[rows][];
        for (i = 0; i < rows; ++i)
            a[i] = new double[cols];
    }

    public Matrix (double[] v) {
        int i;
        this.rows = v.length;
        this.cols = 1;
        a = new double[rows][];
        for (i = 0; i < rows; ++i) {
            a[i] = new double[1];
            a[i][0] = v[i];
        }
    }

    // assigns value to the a[i][j].
    // param: (i,j) - element of matrix, value to assign.
    // returns: true if everything is correct, false - error.
    public boolean assign(int i, int j, double value)
    {
        // check if parameters are valid
        if (i < 0 || i >= rows || j < 0 || j >= cols)
            return false;
        a[i][j] = value;
        return true;
    }

    // Prints the matrix into console.
    public void printscn()
    {
        int i, j;

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

    // Swaps rows. Returns false if errors.
    public boolean swapRows(int i, int j)
    {
        int k;
        double temp;

        if (i < 0 || i >= rows || j < 0 || j >= rows)
            return false;
        for (k = 0; k < cols; ++k)
        {
            temp = a[i][k];
            a[i][k] = a[j][k];
            a[j][k] = temp;
        }
```

```
        return true;
    }

    // Multiplies the row by value
    public boolean multiplyRow(int row, double value)
    {
        int k;

        if (row < 0 || row >= rows)
            return false;
        for (k = 0; k < cols; ++k)
            a[row][k] *= value;
        return true;
    }

    // Adds value to a[i][j].
    // Returns false if error.
    public boolean addValue(int i, int j, double value)
    {
        if (i < 0 || i >= rows || j < 0 || j >= cols)
            return false;
        a[i][j] += value;
        return true;
    }

    // Compares a[i][j] with a zero.
    // Returns 1 if a[i][j] == 0, 0 if a[i][j] != 0
    // and -1 - error
    public int isZero(int i, int j)
    {
        if (i < 0 || i >= rows || j < 0 || j >= cols)
            return -1;
        if (Math.abs(a[i][j]) < EPS)
            return 1;
        else
            return 0;
    }

    public Matrix transpose() {
        Matrix m = new Matrix(cols, rows);
        int i, j;

        for (i = 0; i < rows; ++i)
            for (j = 0; j < cols; ++j)
                m.assign(j, i, getValue(i, j));
        return m;
    }

    public Matrix mul(Matrix m2) throws Exception {
        if (getCols() != m2.getRows())
            throw new Exception("Size mismatch in Matrix.mul");
        Matrix m = new Matrix(rows, m2.getCols());
        int i, j, k;

        for (i = 0; i < rows; ++i)
            for (j = 0; j < m2.getCols(); ++j)
                for (k = 0; k < cols; ++k)
```

```
                m.addValue(i, j, getValue(i, k)*m2.getValue(k, j));

        return m;
    }

    public void copy(Matrix m2) throws Exception {
        if (rows != m2.getRows() || cols != m2.getCols())
            throw new Exception("Size mismatch in Matrix.copy");
        int i, j;

        for (i = 0; i < rows; ++i)
            for (j = 0; j < cols; ++j)
                a[i][j] = m2.getValue(i, j);
    }

    public Matrix add(Matrix m2) throws Exception {
        if (rows != m2.getRows() || cols != m2.getCols())
            throw new Exception("Size mismatch in Matrix.add");
        int i, j;
        Matrix m = new Matrix(rows, cols);

        for (i = 0; i < rows; ++i)
            for (j = 0; j < cols; ++j)
                m.assign(i, j, a[i][j]+m2.getValue(i, j));
        return m;
    }

    public Matrix mul(double value) {
        int i, j;
        Matrix m = new Matrix(rows, cols);

        for (i = 0; i < rows; ++i)
            for (j = 0; j < cols; ++j)
                m.assign(i, j, a[i][j]*value);
        return m;
    }

    public Matrix gauss(Matrix c) throws Exception {
        Matrix b = new Matrix(c.getRows(), c.getCols());

        b.copy(c);
        if (b.getCols() > 1 || cols != b.getRows())
            throw new Exception("Size mismatch in Matrix.gauss");
        if (rows != cols)
            throw new Exception("Non-squared matrix is not allowed in
Matrix.gauss");
        int i, j, k;
        double x;
        Matrix m = new Matrix(rows, cols);

        m.copy(this);

        for (i = 0; i < rows; ++i) {
            // Find non-zero element in the column
            for (j = i; j < rows; ++j)
                if (m.getValue(i, j) != 0) break;
            if (j == rows)
```

```
                throw new Exception("Cannot find the solution in
Matrix.gauss");
            if (i != j) {
                // swap rows
                m.swapRows(i, j);
                b.swapRows(i, j);
            }

            // Divide row
            x = m.getValue(i, i);
            b.assign(i, 0, b.getValue(i, 0)/x);
            m.multiplyRow(i, 1.0/x);
            m.assign(i, i, 1.0);

            for (j = 0; j < i; ++j) {
                x = m.getValue(j, i);
                m.assign(j, i, 0.0);
                for (k = i+1; k < rows; ++k)
                    m.addValue(j, k, -x*m.getValue(i, k));
                b.addValue(j, 0, -x*b.getValue(i, 0));
            }
            for (j = i + 1; j < rows; ++j) {
                x = m.getValue(j, i);
                m.assign(j, i, 0.0);
                for (k = i+1; k < rows; ++k)
                    m.addValue(j, k, -x*m.getValue(i, k));
                b.addValue(j, 0, -x*b.getValue(i, 0));
            }
        }

        return b;
    }
}
```