Name: Sarwan Heer

Matrix Calculator (Engineering Project)

A.txt

```
3 3
1 2 3
4 5 6
7 8 9
```

B.txt

```
3 3
9 8 7
6 5 4
3 2 1
```

Main.java

```
// Main class demonstrating matrix operations
public class Main {
    public static void main(String[] args) {
        // Using Scanner to collect user input from the console
        java.util.Scanner scanner = new java.util.Scanner(System.in);

        // Print statements to explain user choices
        System.out.println("Choose mode: 1 for Text-based, 2 for GUI-based (if implemented)");

        // Collect user choice
        int choice = scanner.nextInt();
        scanner.nextLine(); // Consume the newline character

        // Conditional execution based on user choice
        if (choice == 1) {
            // Text-based mode
            runTextBased();
        } else if (choice == 2) {
            // GUI-based mode
            javax.swing.SwingUtilities.invokeLater(new Runnable() {
                public void run() {
                    new MatrixGUI().createAndShowGUI(); // Create and display the GUI
                }
            });
        } else {
            // Invalid choice handling
            System.out.println("Invalid choice.");
        }
    }

    // Method to handle text-based operations
    public static void runTextBased() {
        // Using Scanner for reading user input
        java.util.Scanner scanner = new java.util.Scanner(System.in);

        // Prompt user for matrix operation input
        System.out.println("Enter matrix operation (e.g., A + B):");

        // Read the entire line of input
        String operation = scanner.nextLine();

        // Split input into parts
        String[] parts = operation.split(" ");

        // Validate input format
        if (parts.length == 3) {
            // Reading matrices from files
            Matrix a = MatrixUtils.readMatrixFromFile(parts[0] + ".txt");
            Matrix b = MatrixUtils.readMatrixFromFile(parts[2] + ".txt");

            // Check if matrices are successfully read
```

```
            if (a == null || b == null) {
                System.out.println("One or both matrix files not found or invalid.");
                return;
            }

            // Initialize result matrix
            Matrix result = null;

            // Determine the operation and perform it
            switch (parts[1]) {
                case "+":
                    result = MatrixOperations.add(a, b);
                    break;
                case "-":
                    result = MatrixOperations.subtract(a, b);
                    break;
                case "*":
                    result = MatrixOperations.multiply(a, b);
                    break;
                default:
                    System.out.println("Invalid operation.");
                    return;
            }

            // Output result to console
            System.out.println("Result:");
            System.out.println(result);

            // Write result to file
            try {
                writeResultToFile("R.txt", result);
                System.out.println("Result written to R.txt");
            } catch (java.io.IOException e) {
                System.out.println("An error occurred while writing the result to R.txt: " + e.getMessage());
            }
        } else {
            // Invalid input format handling
            System.out.println("Invalid input format.");
        }
    }

    // Method to write result matrix to file
    public static void writeResultToFile(String fileName, Matrix result) throws java.io.IOException {
        java.io.PrintWriter print = new java.io.PrintWriter(fileName);

        // Print matrix dimensions
        print.println(result.getRows() + " " + result.getCols());

        // Print matrix data
        for (int i = 0; i < result.getRows(); i++) {
            for (int j = 0; j < result.getCols(); j++) {
                print.print(result.get(i, j) + " ");
            }
            print.println();
        }
        // Close the PrintWriter
        print.close();
    }
}
```

Matrix.java

```
// Class representing a matrix
public class Matrix {
    // Matrix data stored in a 2D array
    private int[][] data;
    private int rows;
    private int cols;

    // Constructor to initialize matrix dimensions
    public Matrix(int rows, int cols) {
        this.rows = rows;
        this.cols = cols;
        data = new int[rows][cols];
    }

    // Method to get the number of rows
    public int getRows() {
        return rows;
    }

    // Method to get the number of columns
    public int getCols() {
        return cols;
    }

```

```
    // Method to get the value at a specific row and column
    public int get(int row, int col) {
        return data[row][col];
    }

    // Method to set the value at a specific row and column
    public void set(int row, int col, int value) {
        data[row][col] = value;
    }

    // Method to return the matrix as a string
    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                sb.append(data[i][j]).append(" ");
            }
            sb.append("\n");
        }
        return sb.toString();
    }
}
```

MatrixGUI.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

// Class to handle GUI for matrix operations
public class MatrixGUI {
    // GUI components
    private JFrame frame;
    private JTextField rowsField;
    private JTextField colsField;
    private JPanel matrixAPanel;
    private JPanel matrixBPanel;
    private JPanel resultPanel;
    private JTextField[][] matrixAFields;
    private JTextField[][] matrixBFields;
    private JTextField[][] resultFields;
    private JComboBox<String> operationBox;

    // Maximum dimensions for the matrices
    private static final int MAX_DIMENSION = 3;

    // Method to create and show the GUI
    public void createAndShowGUI() {
        frame = new JFrame("Matrix Operations");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(800, 600);

        // Input panel for matrix dimensions and operations
        JPanel inputPanel = new JPanel();
        inputPanel.setLayout(new GridLayout(3, 2));

        JLabel rowsLabel = new JLabel("Number of rows (max 3):");
        rowsField = new JTextField();
        inputPanel.add(rowsLabel);
        inputPanel.add(rowsField);

        JLabel colsLabel = new JLabel("Number of columns (max 3):");
        colsField = new JTextField();
        inputPanel.add(colsLabel);
        inputPanel.add(colsField);

        JButton createMatricesButton = new JButton("Create Matrices");
        createMatricesButton.addActionListener(new CreateMatricesListener());
        inputPanel.add(createMatricesButton);

        JLabel operationLabel = new JLabel("Operation (+, -, *):");
        operationBox = new JComboBox<>(new String[]{"+", "-", "*"});
        inputPanel.add(operationLabel);
        inputPanel.add(operationBox);

        // Panels for matrices and result
        matrixAPanel = new JPanel();
        matrixBPanel = new JPanel();
        resultPanel = new JPanel();

        JPanel matricesPanel = new JPanel();
        matricesPanel.setLayout(new GridLayout(1, 3));
        matricesPanel.add(matrixAPanel);
        matricesPanel.add(matrixBPanel);
```

```
        matricesPanel.add(resultPanel);

        // Setting layout of the frame
        frame.setLayout(new BorderLayout());
        frame.add(inputPanel, BorderLayout.NORTH);
        frame.add(matricesPanel, BorderLayout.CENTER);

        JButton calculateButton = new JButton("Calculate");
        calculateButton.addActionListener(new CalculateButtonListener());
        frame.add(calculateButton, BorderLayout.SOUTH);

        frame.setVisible(true);
    }

    // Listener for creating matrices
    private class CreateMatricesListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            try {
                // Reading dimensions from text fields
                int rows = Integer.parseInt(rowsField.getText());
                int cols = Integer.parseInt(colsField.getText());

                // Check for valid dimensions
                if (rows > MAX_DIMENSION || cols > MAX_DIMENSION) {
                    JOptionPane.showMessageDialog(frame, "The maximum number of rows and columns is 3.");
                    return;
                }

                // Reset panels
                matrixAPanel.removeAll();
                matrixBPanel.removeAll();
                resultPanel.removeAll();

                matrixAPanel.setLayout(new GridLayout(rows, cols));
                matrixBPanel.setLayout(new GridLayout(rows, cols));
                resultPanel.setLayout(new GridLayout(rows, cols));

                matrixAFields = new JTextField[rows][cols];
                matrixBFields = new JTextField[rows][cols];
                resultFields = new JTextField[rows][cols];

                // Create text fields for matrix input
                for (int i = 0; i < rows; i++) {
                    for (int j = 0; j < cols; j++) {
                        matrixAFields[i][j] = new JTextField();
                        matrixAPanel.add(matrixAFields[i][j]);

                        matrixBFields[i][j] = new JTextField();
                        matrixBPanel.add(matrixBFields[i][j]);

                        resultFields[i][j] = new JTextField();
                        resultFields[i][j].setEditable(false);
                        resultPanel.add(resultFields[i][j]);
                    }
                }

                frame.revalidate();
                frame.repaint();
            } catch (NumberFormatException ex) {
                JOptionPane.showMessageDialog(frame, "Please enter valid integers for rows and columns.");
            }
        }
    }

    // Listener for calculating matrix operations
    private class CalculateButtonListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            try {
                // Read dimensions
                int rows = matrixAFields.length;
                int cols = matrixAFields[0].length;

                // Create matrix objects
                Matrix a = new Matrix(rows, cols);
                Matrix b = new Matrix(rows, cols);

                // Read values from text fields
                for (int i = 0; i < rows; i++) {
                    for (int j = 0; j < cols; j++) {
                        a.set(i, j, Integer.parseInt(matrixAFields[i][j].getText()));
                        b.set(i, j, Integer.parseInt(matrixBFields[i][j].getText()));
                    }
                }

                // Determine operation and perform it
                Matrix result = null;
                switch ((String) operationBox.getSelectedItem()) {
                    case "+":
```

```
                        result = MatrixOperations.add(a, b);
                        break;
                    case "-":
                        result = MatrixOperations.subtract(a, b);
                        break;
                    case "*":
                        result = MatrixOperations.multiply(a, b);
                        break;
                }

                // Display result in the result text fields
                for (int i = 0; i < rows; i++) {
                    for (int j = 0; j < cols; j++) {
                        resultFields[i][j].setText(Integer.toString(result.get(i, j)));
                    }
                }
            } catch (NumberFormatException ex) {
                JOptionPane.showMessageDialog(frame, "Please enter valid integers in the matrix fields.");
            }
        }
    }
}
```

MatrixOperations.java

```
// Class for performing matrix operations
public class MatrixOperations {
    // Method to add two matrices
    public static Matrix add(Matrix a, Matrix b) {
        int rows = a.getRows();
        int cols = a.getCols();
        Matrix result = new Matrix(rows, cols);

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                result.set(i, j, a.get(i, j) + b.get(i, j));
            }
        }

        return result;
    }

    // Method to subtract matrix b from matrix a
    public static Matrix subtract(Matrix a, Matrix b) {
        int rows = a.getRows();
        int cols = a.getCols();
        Matrix result = new Matrix(rows, cols);

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                result.set(i, j, a.get(i, j) - b.get(i, j));
            }
        }

        return result;
    }

    // Method to multiply two matrices
    public static Matrix multiply(Matrix a, Matrix b) {
        int rowsA = a.getRows();
        int colsA = a.getCols();
        int rowsB = b.getRows();
        int colsB = b.getCols();

        if (colsA != rowsB) {
            throw new IllegalArgumentException("Number of columns in Matrix A must equal number of rows in Matrix B.");
        }

        Matrix result = new Matrix(rowsA, colsB);

        for (int i = 0; i < rowsA; i++) {
            for (int j = 0; j < colsB; j++) {
                int sum = 0;
                for (int k = 0; k < colsA; k++) {
                    sum += a.get(i, k) * b.get(k, j);
                }
                result.set(i, j, sum);
            }
        }

        return result;
    }
}
```

MatrixUtils.java

```java
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

// Utility class for reading matrices from files
public class MatrixUtils {
    // Method to read a matrix from a file
    public static Matrix readMatrixFromFile(String fileName) {
        try {
            // Use Scanner to read from file
            Scanner scanner = new Scanner(new File(fileName));

            // Read dimensions
            int rows = scanner.nextInt();
            int cols = scanner.nextInt();
            Matrix matrix = new Matrix(rows, cols);

            // Read matrix data
            for (int i = 0; i < rows; i++) {
                for (int j = 0; j < cols; j++) {
                    matrix.set(i, j, scanner.nextInt());
                }
            }

            // Close the scanner
            scanner.close();
            return matrix;
        } catch (FileNotFoundException e) {
            System.out.println("File not found: " + fileName);
            return null;
        }
    }
}
```