Folder POO_2023_Lab5/src/main/java

7 printable files

(file list disabled)

POO_2023_Lab5/src/main/java/Input.java

```
import static java.lang.System.exit;

public class Input {

  static String help = """
      Please input n1, m1, n2, m2, modulus, separated by spaces.
      n1 : number of lines of the first matrix
      m1 : number of columns of the first matrix
      n2 : number of lines of the second matrix
      m2 : number of columns of the second matrix
      modulus : the two matrices' modulus
      """;

  static final int expectedSize = 5;

  static int[] parseInput(String[] args) {
    int[] parsedValues = new int[args.length];
    if (args.length == 0) {
      System.out.print(
          "\nNo input found.\n" + help
      );
      exit(1);
    }
    if (args.length != 5) {
      System.out.print(
          "\nInvalid count of values (expected " + expectedSize + ", got " + args.length + ").\n" + help
      );
      exit(1);
    }
    try {
      for (int i = 0; i < args.length; ++i) {
        parsedValues[i] = Integer.parseInt(args[i]);
      }
    } catch (NumberFormatException nfe) {
      throw new IllegalArgumentException(
          "Illegal input detected. Please input integers only."
      );
    }
    return parsedValues;
  }
}
```

POO_2023_Lab5/src/main/java/Main.java

```
/**
 * @author Gonçalo Carvalheiro Heleno
 * @author Sven Ferreira Silva
 */

import matrix.Matrix;
```

```java
public class Main {

  public static void main(String[] args) {
    int[] values = Input.parseInput(args);

    int modulus = values[args.length - 1];

    Matrix matrix1 = new Matrix(values[0], values[1], modulus),
        matrix2 = new Matrix(values[2], values[3], modulus);

    System.out.println("\nThe modulus is " + modulus + ".\n");
    System.out.println("one:");
    matrix1.printMatrix();
    System.out.println("\ntwo:");
    matrix2.printMatrix();

    System.out.println("\none + two:");
    Matrix result = matrix1.addTo(matrix2);
    result.printMatrix();

    System.out.println("\none - two:");
    result = matrix1.subtractWith(matrix2);
    result.printMatrix();

    System.out.println("\none x two:");
    result = matrix1.multiplyBy(matrix2);
    result.printMatrix();

  }
}
```

POO_2023_Lab5/src/main/java/matrix/Matrix.java

```java
package matrix;

import java.util.Random;
import matrix.binaryOperation.*;

/**
 * @author Gonçalo Carvalheiro Heleno
 * @author Sven Ferreira Silva
 */

public class Matrix {

  private final int modulus;
  private final int nLines;
  private final int mColumns;
  private final int[][] matrixArray;

  public Matrix() {
    this(new int[0][0], 1);
  }

  public Matrix(int nLines, int mColumns, int modulus) {
    if (modulus <= 0) {
      throw new IllegalArgumentException("Modulus should be strictly greater than 0.");
    }
    if (nLines < 0 || mColumns < 0) {
      throw new IllegalArgumentException("Number of lines and/or columns cannot be negative.");
    }
```

```
    this.modulus = modulus;
    if (nLines == 0 || mColumns == 0) {
      this.nLines = this.mColumns = 0;
      this.matrixArray = new int[this.nLines][this.mColumns];
    } else {
      this.nLines = nLines;
      this.mColumns = mColumns;
      this.matrixArray = new int[this.nLines][this.mColumns];

      Random random = new Random();
      for (int line = 0; line < this.nLines; ++line) {
        for (int column = 0; column < this.mColumns; ++column) {
          this.matrixArray[line][column] = random.nextInt(modulus);
        }
      }
    }
  }

  public Matrix(int[][] matrixArray, int modulus) {
    if (modulus <= 0) {
      throw new IllegalArgumentException("Modulus should be strictly greater than 0.");
    }
    for (int line = 0; line < matrixArray.length - 1; ++line) {
      if (matrixArray[line].length != matrixArray[line + 1].length) {
        throw new IllegalArgumentException(
            "Invalid matrix array! Lines of the matrix are of different size.");
      }
    }
    for (int[] line : matrixArray) {
      for (int element : line) {
        if (element < 0 || element >= modulus) {
          throw new IllegalArgumentException(
              "Invalid matrix array! Elements must be in the range `0 <= element < modulus`.");
        }
      }
    }

    this.modulus = modulus;
    if (matrixArray.length == 0) {
      this.nLines = this.mColumns = 0;
    } else {
      this.nLines = matrixArray.length;
      this.mColumns = matrixArray[0].length;
    }
    this.matrixArray = matrixArray;
  }

  /**
   * Constructor to copy a Matrix into a new object. This constructor uses the resizing constructor,
   * but instead of passing new values for {@code nLines} and {@code mColumns}, it simply uses the
   * current values from the Matrix passed in argument.
   *
   * @param matrix The Matrix to be copied.
   * @implNote
   * @see matrix.Matrix#Matrix(Matrix, int, int)
   */
  public Matrix(Matrix matrix) {
    this(matrix, matrix.nLines, matrix.mColumns);
  }

  /**
   * Constructor to create a bigger/smaller Matrix based on the Matrix passed as an argument.
   * <p>
   * In the case that the new {@code nLines} or {@code mColumns} is bigger than the current value,
```

```java
   * the new positions will be filled with zeros. In the case that these are smaller, then the new
   * Matrix will be truncated.
   *
   * @param matrix The Matrix that is the basis of the new smaller/bigger Matrix.
   * @param newN   The new number of lines.
   * @param newM   The new number of columns.
   */
  public Matrix(Matrix matrix, int newN, int newM) {
    if (newN < 0 || newM < 0) {
      throw new IllegalArgumentException("Number of lines and/or columns cannot be negative.");
    }

    nLines = newN;
    mColumns = newM;
    modulus = matrix.modulus;
    matrixArray = new int[this.nLines][this.mColumns];

    final int MIN_LINES = Math.min(this.nLines, matrix.nLines);
    final int MIN_COLUMNS = Math.min(this.mColumns, matrix.mColumns);

    for (int line = 0; line < MIN_LINES; ++line) {
      for (int column = 0; column < MIN_COLUMNS; ++column) {
        this.matrixArray[line][column] = matrix.matrixArray[line][column];
      }
    }
  }

  public int getN() {
    return nLines;
  }

  public int getM() {
    return mColumns;
  }

  public int getModulus() {
    return modulus;
  }

  public int getElement(int line, int column) {
    return this.matrixArray[line][column];
  }

  public void printMatrix() {
    for (int line = 0; line < nLines; ++line) {
      for (int column = 0; column < mColumns; ++column) {
        System.out.print(matrixArray[line][column] + " ");
      }
      System.out.println();
    }
  }

  public Matrix addTo(Matrix otherMatrix) {
    return Addition.add(this, otherMatrix);
  }

  public Matrix subtractWith(Matrix otherMatrix) {
    return Subtraction.subtract(this, otherMatrix);
  }

  public Matrix multiplyBy(Matrix otherMatrix) {
    return Multiplication.multiply(this, otherMatrix);
  }
}
```

POO_2023_Lab5/src/main/java/matrix/binaryOperation/Addition.java

```java
package matrix.binaryOperation;

import matrix.Matrix;

/**
 * @author Gonçalo Carvalheiro Heleno
 * @author Sven Ferreira Silva
 */

public class Addition extends BinaryOperation {

  private Addition() {
    super();
  }

  public static Matrix add(Matrix matrix1, Matrix matrix2) {
    Addition addition = new Addition();
    return loopAndPerformOperation(matrix1, matrix2, addition);
  }

  @Override
  protected int operation(int operand1, int operand2, int modulus) {
    return (operand1 + operand2) % modulus;
  }
}
```

POO_2023_Lab5/src/main/java/matrix/binaryOperation/BinaryOperation.java

```java
package matrix.binaryOperation;

import matrix.Matrix;

/**
 * @author Gonçalo Carvalheiro Heleno
 * @author Sven Ferreira Silva
 */

public abstract class BinaryOperation {

  protected BinaryOperation() {
    super();
  }

  protected static Matrix loopAndPerformOperation(Matrix matrix1, Matrix matrix2,
      BinaryOperation binaryOperation) {
    if (matrix1 == null || matrix2 == null) {
      throw new NullPointerException(
          "Invalid reference! One of the matrices passed as an argument is null.");
    }
    if (matrix1.getModulus() != matrix2.getModulus()) {
      throw new ArithmeticException("Modulus of matrices is not identical.");
    }

    int resultN = matrix1.getN(),
        resultM = matrix1.getM(),
        resultModulus = matrix1.getModulus();

    if (matrix1.getN() != matrix2.getN() || matrix1.getM() != matrix2.getM()) {
      resultN = Math.max(matrix1.getN(), matrix2.getN());
```

```
        resultM = Math.max(matrix1.getM(), matrix2.getM());
        matrix1 = new Matrix(matrix1, resultN, resultM);
        matrix2 = new Matrix(matrix2, resultN, resultM);
      }

      int[][] resultMatrixArray = new int[resultN][resultM];

      for (int line = 0; line < resultN; ++line) {
        for (int column = 0; column < resultM; ++column) {
          resultMatrixArray[line][column] =
              binaryOperation.operation(matrix1.getElement(line, column),
                  matrix2.getElement(line, column),
                  resultModulus);
        }
      }

      return new Matrix(resultMatrixArray, resultModulus);
    }

    protected abstract int operation(int operand1, int operand2, int modulus);
  }
```

POO_2023_Lab5/src/main/java/matrix/binaryOperation/Multiplication.java

```
  package matrix.binaryOperation;

  import matrix.Matrix;

  /**
   * @author Gonçalo Carvalheiro Heleno
   * @author Sven Ferreira Silva
   */

  public class Multiplication extends BinaryOperation {

    private Multiplication() {
      super();
    }

    public static Matrix multiply(Matrix matrix1, Matrix matrix2) {
      Multiplication multiplication = new Multiplication();
      return loopAndPerformOperation(matrix1, matrix2, multiplication);
    }

    @Override
    protected int operation(int operand1, int operand2, int modulus) {
      return (operand1 * operand2) % modulus;
    }
  }
```

POO_2023_Lab5/src/main/java/matrix/binaryOperation/Subtraction.java

```
  package matrix.binaryOperation;

  import matrix.Matrix;

  /**
   * @author Gonçalo Carvalheiro Heleno
   * @author Sven Ferreira Silva
```

```java
 */

public class Subtraction extends BinaryOperation {

  private Subtraction() {
    super();
  }

  public static Matrix subtract(Matrix matrix1, Matrix matrix2) {
    Subtraction subtraction = new Subtraction();
    return loopAndPerformOperation(matrix1, matrix2, subtraction);
  }

  @Override
  protected int operation(int operand1, int operand2, int modulus) {
    return Math.floorMod(operand1 - operand2, modulus);
  }
}
```