Programmation Orientée Objet

Rapport – Laboratoire 5 : Matrices

Gonçalo Carvalheiro Heleno

Sven Ferreira Silva

Table of Contents

Modélisation

Celui-ci est inclus en annexe à ce rapport en format PDF, dans le même dossier que ce fichier Word. Le fichier SLY original est dans le dossier UML du dossier parent à celui-ci.

Choix de conception

Le programme qui contient le code implémentant les matrices et opérations est divisé en plusieurs packages :

- un premier package `matrix` contenant la classe `Matrix` ;
- un second package `binaryOperation`, sous-package de `matrix`, contenant les classes de calcul.

Celles-ci sont décrites plus en détail dans la section suivante.

Considérations techniques

La valeur du module peut être que positive, selon les contraintes de l'énoncé. Les valeurs contenues dans les matrices ne peuvent être comprise que dans l'intervalle [0, module – 1], prenant les valeurs entières dans cet intervalle.

Notre implémentation utilise la fonction `floorMod` dû au besoin d'un résultat positif ou nul pour les valeurs des matrices. La soustraction de matrices pouvant engendrer des résultats partiels négatifs (avant l'application du module), nous nous sommes inspirés d'exemples dans la documentation de la fonction pour obtenir un module positif en toute circonstance. Ces exemples sont disponibles à cette adresse.

Nous avons décidé que les constructeurs des sous-classes de `BinaryOperation` doivent être privés, ceci afin de limiter l'instanciation de ces objets en-dehors des classes dédiées à l'implémentation des diverses opérations. Une contrainte de cette approche, que nous sommes conscients, c'est que les opérations doivent être appelés par une méthode statique que prends les 2 matrices en paramètres (p.e. `Addition.add(matrix1, matrix2)`, ce qui n'est pas très orienté objet. Toutefois, nous trouvons que c'est plus propre de cette façon, car nous pouvons garantir que ces classes ne sont instanciées que lors d'appels aux respectives fonctions statiques.

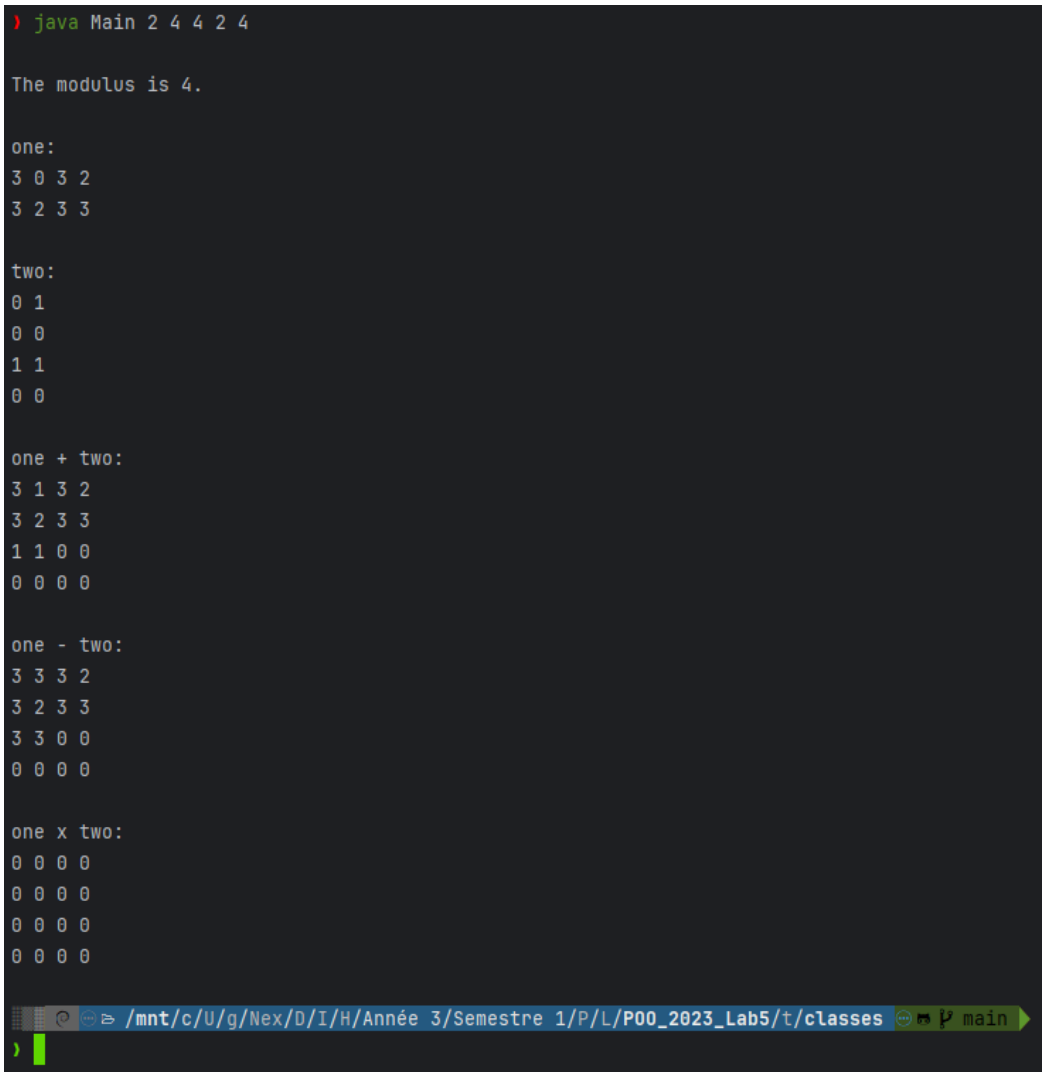
Pour rendre le code plus orienté objet, nous avons créé de méthodes dans la classe `Matrix` pour faire les opérations, ce qui appellent respectivement les méthodes statiques de `BinaryOperation`.

Classes du programme

Classe `Main`

Contient le programme principal et les fonctions accessoires. Cette classe fait appel à la classe `Input` implémentant une fonction d'analyse et de traitement des arguments d'entrée. Une fois les données d'entrée vérifiées et converties en entier, le programme génère deux matrices grâce à ces valeurs numériques (ces valeurs étant les dimensions de la première matrice, celles de la deuxième, ainsi que leur module commun).

Enfin, après avoir affiché ces matrices, il effectue les opérations disponibles ainsi que leur résultat.



Classe `Matrix`

La classe `Matrix` implémente les attributs privés suivants :

- le module,
- nombre de lignes,
- nombre de colonnes,
- les données numériques stockées en tant que tableau de tableaux d'entiers.

Les constructeurs implémentés sont :

- le constructeur par défaut, créant une matrice vide et de module unitaire ;
- un constructeur prenant les dimensions attendues et le module, créant une matrice avec de valeurs aléatoires ;
- un constructeur prenant directement un tableau de tableaux d'entiers ainsi que le module attendu ;
- un constructeur de copie ;
- un constructeur prenant une matrice et des dimensions différentes de celle-ci ;

Les *getters* usuels sont naturellement implémentés, ainsi que les fonctions membres suivantes :

- `printMatrix()`, afin d'afficher le contenu d'une matrice en respectant ses dimensions à l'affichage,
- `addTo`, `subtractWith`, `multiplyBy`, qui effectuent les opérations arithmétiques homonymes.

Classe abstraite `BinaryOperation`

Cette classe abstraite implémente la fonction de calcul principal, `loopAndPerformOperation`, qui itère sur les valeurs de mêmes coordonnées dans les deux matrices reçues en paramètre afin d'effectuer l'opération implémentée par l'une de ses sous-classes.

Également, cette classe implémente la fonction membre abstraite `operation`, prenant en paramètre les deux valeurs courantes des matrices, ainsi que leur module. Cette fonction membre, qui doit être implémentée dans les sous-classes de la classe `BinaryOperation`, doit effectuer l'opération arithmétique attendue et retourner la valeur numérique résultante de cette opération.

À l'heure actuelle, trois sous-classes de `BinaryOperation` sont disponibles : `Addition`, `Subtraction`, `Multiplication`.

Classes de test

Classe `MatrixTest`

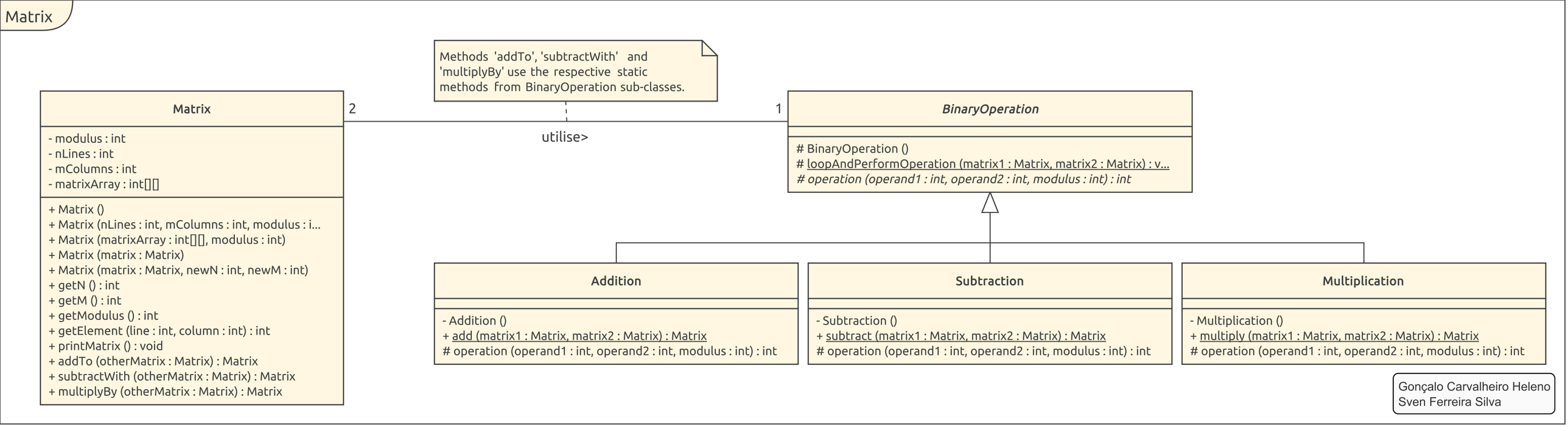
Cette classe implémente les tests suivants :

- test du constructeur par défaut,
- test du constructeur aléatoire (construction par dimensions et module),
- test des exceptions du constructeur aléatoire,
- test du constructeur par tableau de tableaux d'entiers,
- test des exceptions du constructeur susmentionné,
- test du constructeur de copie,
- test des exceptions du constructeur de copie,
- test du constructeur de redimensionnement,
- test des exceptions du constructeur de redimensionnement

Classe `BinaryOperationTest`

Cette classe implémente les tests suivants :

- test d'addition,
- test d'addition, cas limite : l'une des matrices est nulle,
- test des exceptions du mécanisme d'addition,
- test de soustraction,
- test de soustraction, cas limite: l'une des matrices est nulle,
- test des exceptions du mécanisme de soustraction,
- test de multiplication,
- test de multiplication, cas limite : l'une des matrices est nulle,
- test des exceptions du mécanisme de multiplication.


Matrix
Methods 'addTo', 'subtractWith' and 'multiplyBy' use the respective static methods from BinaryOperation sub-classes.
Matrix
- modulus : int
- nLines : int
- mColumns : int
- matrixArray : int[][]
+ Matrix ()
+ Matrix (nLines : int, mColumns : int, modulus : i...
+ Matrix (matrixArray : int[][], modulus : int)
+ Matrix (matrix : Matrix)
+ Matrix (matrix : Matrix, newN : int, newM : int)
+ getN () : int
+ getM () : int
+ getModulus () : int
+ getElement (line : int, column : int) : int
+ printMatrix () : void
+ addTo (otherMatrix : Matrix) : Matrix
+ subtractWith (otherMatrix : Matrix) : Matrix
+ multiplyBy (otherMatrix : Matrix) : Matrix
2
1
utilise>
BinaryOperation
BinaryOperation ()
loopAndPerformOperation (matrix1 : Matrix, matrix2 : Matrix) : v...
operation (operand1 : int, operand2 : int, modulus : int) : int
Addition
- Addition ()
+ add (matrix1 : Matrix, matrix2 : Matrix) : Matrix
# operation (operand1 : int, operand2 : int, modulus : int) : int
Subtraction
- Subtraction ()
+ subtract (matrix1 : Matrix, matrix2 : Matrix) : Matrix
# operation (operand1 : int, operand2 : int, modulus : int) : int
Multiplication
- Multiplication ()
+ multiply (matrix1 : Matrix, matrix2 : Matrix) : Matrix
# operation (operand1 : int, operand2 : int, modulus : int) : int
Gonçalo Carvalheiro Heleno
Sven Ferreira Silva

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);
  }
}
```