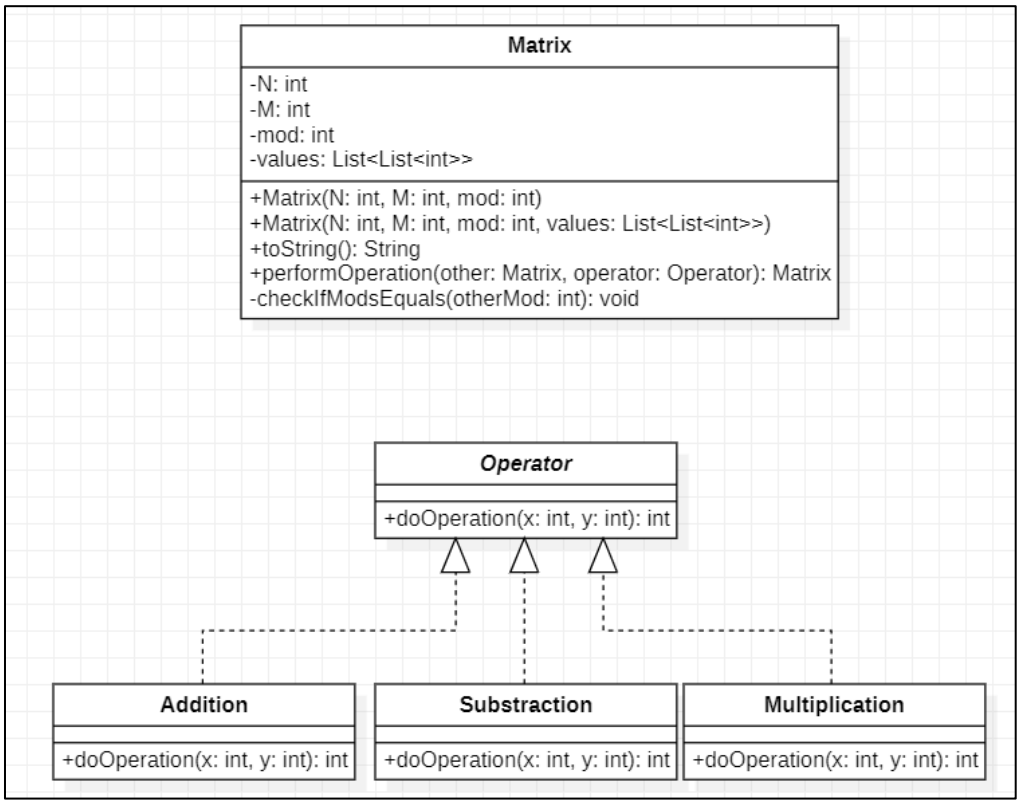
Gruber Adam
Pittet Axel

Rapport labo 05 POO

Modélisation UML



Choix de conception

Nous avons décidé de créer 2 constructeurs différents pour la classe Matrix afin de couvrir les différents moyens d'instanciation de l'objet.

La fonction checkIfModsEquals renvoie void car elle jettera directement une exception si les modulos ne sont pas égaux. Elle est également privée car elle ne sera jamais utilisée en dehors de la classe Matrix.

L'interface Operator et ses sous-classes ont été crées afin d'être utilisées dans la fonction performOperation de Matrix afin de ne pas avoir besoin de créer une fonction différente dans Matrix pour chaque type de calcul.

Les fonctions doOperation des classes Addition, Substraction et Mulitplication vont toutes override la fonction de l'interface Operator.

Tests et résultats

Description du test	Résultat	Commentaires
Instanciation d'une Matrice avec le constructeur utilisant la taille et le modulo	OK	La Matrice est bien générée avec des valeurs aléatoires correctes.
Instanciation d'une Matrice avec le constructeur utilisant la taille, le modulo et les valeurs.	OK	
Instanciation d'une Matrice avec des valeurs plus grandes que le modulo	OK	Les valeurs sont adaptées.
Instanciation d'une matrice avec un tableau de valeurs ne correspondant pas à la taille spécifiée	OK	La Matrice remplace les trous par des 0 et ignore les valeurs en dehors de la taille spécifiée
Exécution d'une opération avec des Matrices de même modulo.	OK	Les opérations sont correctes.
Exécution d'une opération avec des Matrices de modulos différents.	OK	Le système indique que les modulos ne sont pas égaux et renvoie « null » à chaque opération.
Exécution d'une opération avec des Matrices de même taille.	OK	Les opérations sont correctes.
Exécution d'une opération avec des Matrices de tailles différentes.	OK	La taille de la Matrice de résultat est adaptée et les opérations sont correctes.

Listing du code

Le listing du code est présent à la fin de ce document

Annexes

Le code source de ce projet est disponible dans le répertoire « Code source ».

Folder src

6 printable files

(file list disabled)

src\main\java\ch\heigvd\poo\Main.java

```java
package ch.heigvd.poo;
import ch.heigvd.poo.operators.*;

/**
 * @author Gruber Adam
 * @author Pittet Axel
 */
public class Main {
    public static void main(String[] args) {

        int MODULUS = 5;

        int[][] values1 = {{1, 2, 3, 4}, {1, 2}};

        Matrix matrix1 = new Matrix(3, 3, MODULUS, values1);
        Matrix matrix2 = new Matrix(3, 3,MODULUS);

        Operator operatorAdd = new Addition();
        Operator operatorSub = new Subtraction();
        Operator operatorMul = new Multiplication();


        System.out.println("The modulus is " + MODULUS);

        System.out.println("one :");
        System.out.println(matrix1);

        System.out.println("two :");
        System.out.println(matrix2);

        System.out.println("one + two:");
        System.out.println(matrix1.performOperation(matrix2, operatorAdd));

        System.out.println("one - two:");
        System.out.println(matrix1.performOperation(matrix2, operatorSub));

        System.out.println("one x two:");
        System.out.println(matrix1.performOperation(matrix2, operatorMul));
    }
}
```

src\main\java\ch\heigvd\poo\Matrix.java

```java
package ch.heigvd.poo;

import ch.heigvd.poo.operators.Operator;

/**
 * @author Gruber Adam
 * @author Pittet Axel
 * Matrix class for handling matrices with operations constrained by a modulus.
 * Provides methods for creating matrices, displaying them, and performing
 * operations with other matrices.
 *
 */
public class Matrix {
```

```java
static final int SEED = 1;
private int N;
private int M;
private int mod;
private int[][] values;

/**
 * Constructor that generates a matrix with random values.
 *
 * @param N   Number of rows in the matrix.
 * @param M   Number of columns in the matrix.
 * @param mod Modulus used for constraining values.
 */
public Matrix(int N, int M, int mod) {
    this.N = N;
    this.M = M;
    this.mod = mod;
    this.values = new int[N][M];

    java.util.Random random = new java.util.Random(SEED);

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++) {
            this.values[i][j] = random.nextInt(mod);
        }
    }
}

/**
 * Constructor that generates a matrix from provided values.
 * Values will be constrained by the modulus.
 *
 * @param N      Number of rows in the matrix.
 * @param M      Number of columns in the matrix.
 * @param mod    Modulus used for constraining values.
 * @param values 2D array of values to initialize the matrix.
 *               If the provided array has fewer rows or columns,
 *               missing values are replaced by 0.
 */
public Matrix(int N, int M, int mod, int[][] values) {
    this.N = N;
    this.M = M;
    this.mod = mod;
    this.values = new int[N][M];

    int currentValuesN = values.length;

    for (int i = 0; i < N; i++) {

        if (i < currentValuesN) {
            int currentValuesM = values[i].length;
            for (int j = 0; j < M; j++) {

                if (j < currentValuesM) this.values[i][j] = values[i][j] % mod;
                else this.values[i][j] = 0;
            }
        } else {

            for (int j = 0; j < M; j++) {
                this.values[i][j] = 0;
            }
        }
    }
}

/**
```

```
     * Generates a string representation of the matrix.
     *
     * @return String representing the matrix in a grid format.
     */
    @Override
    public String toString() {
        StringBuilder str = new StringBuilder();
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < M; j++) {
                str.append(values[i][j]).append(" ");
            }
            str.append("\n");
        }
        return str.toString();
    }

    /**
     * Performs an element-wise operation between this matrix and another matrix
     * using a specified operator, constrained by the modulus.
     *
     * @param other    The other matrix to operate with.
     * @param operator The operation to perform, represented by the Operator interface.
     * @return A new Matrix containing the result of the operation.
     * Returns null if the operation fails due to incompatible matrices.
     */
    public Matrix performOperation(Matrix other, Operator operator) {
        try {
            checkIfModsEquals(other);
            int biggestN = Math.max(this.N, other.N);
            int biggestM = Math.max(this.M, other.M);

            Matrix tempMatrix = new Matrix(biggestN, biggestM, mod, this.values);
            Matrix tempMatrix2 = new Matrix(biggestN, biggestM, mod, other.values);

            for (int i = 0; i < biggestN; i++) {
                for (int j = 0; j < biggestM; j++) {
                    tempMatrix.values[i][j] = Math.floorMod(operator.doOperation(tempMatrix.values[i][j],
tempMatrix2.values[i][j]), mod);
                }
            }

            return tempMatrix;
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
        return null;
    }

    /**
     * Checks if the modulus of this matrix is equal to the modulus of another matrix.
     *
     * @param other The other matrix to compare with.
     * @throws RuntimeException If the modulus of the matrices are not equal.
     */
    private void checkIfModsEquals(Matrix other) {
        if (this.mod != other.mod) {
            throw new RuntimeException("The modulus are not equal");
        }
    }
}
```

src\main\java\ch\heigvd\poo\operators\Addition.java

```
package ch.heigvd.poo.operators;
```

```java
/**
 * @author Gruber Adam
 * @author Pittet Axel
 * Addition class implementing the Operator interface.
 * Provides an addition operation between two integers.
 */
public class Addition implements Operator {
    /**
     * Adds two integer operands.
     *
     * @param x First operand.
     * @param y Second operand.
     * @return Sum of x and y.
     */
    @Override
    public int doOperation(int x, int y) {
        return x + y;
    }
}
```

src\main\java\ch\heigvd\poo\operators\Multiplication.java

```java
package ch.heigvd.poo.operators;

/**
 * @author Gruber Adam
 * @author Pittet Axel
 * Multiplication class implementing the Operator interface.
 * Provides a multiplication operation between two integers.
 */
public class Multiplication implements Operator {
    /**
     * Multiplies two integer operands.
     *
     * @param x First operand.
     * @param y Second operand.
     * @return Product of x and y.
     */
    @Override
    public int doOperation(int x, int y) {
        return x * y;
    }
}
```

src\main\java\ch\heigvd\poo\operators\Operator.java

```java
package ch.heigvd.poo.operators;

/**
 * @author Gruber Adam
 * @author Pittet Axel
 * Operator interface for defining a mathematical operation between two integers.
 */
public interface Operator {
    /**
     * Executes a specific operation on two integer operands.
     *
     * @param x First operand.
     * @param y Second operand.
     * @return Result of the operation between x and y.
     */
    int doOperation(int x, int y);
}
```

src\main\java\ch\heigvd\poo\operators\Subtraction.java

```java
package ch.heigvd.poo.operators;

/**
 * @author Gruber Adam
 * @author Pittet Axel
 * Subtraction class implementing the Operator interface.
 * Provides a subtraction operation between two integers.
 */
public class Subtraction implements Operator {
    /**
     * Subtracts the second operand from the first operand.
     *
     * @param x First operand.
     * @param y Second operand.
     * @return Difference between x and y.
     */
    @Override
    public int doOperation(int x, int y) {
        return x - y;
    }
}
```