Folder src/main

5 printable files

(file list disabled)

src/main/java/ch/heigvd/dai/Addition.java

```java
/**
 * @author Aubry Antoine
 * @author Faria dos Santos Dani Tiago
 */

package ch.heigvd.dai;

/**
 * @class Addition
 *
 * @brief Implémente l'opération d'addition entre deux éléments d'une matrice.
 *
 * Fournit une implémentation concrète de l'interface MatrixOperation,
 * en définissant la méthode apply qui réalise l'addition de deux entiers.
 */
public class Addition implements MatrixOperation {
    public int apply(int a, int b) {
        return a + b;
    }
}
```

src/main/java/ch/heigvd/dai/Matrix.java

```java
/**
 * @author Aubry Antoine
 * @author Faria dos Santos Dani Tiago
 */

package ch.heigvd.dai;
import java.util.Scanner;



/**
 * @class Matrix
 *
 * @brief Cette classe représente une matrice avec des opérations définies sous un modulo spécifique.
 *
 * La classe Matrix permet de créer une matrice de dimensions spécifiées (lignes et colonnes) et d'effectuer
 * des opérations élémentaires sur les matrices, comme l'addition, la soustraction, ou la multiplication,
 * en fonction de l'opération passée en paramètre. Chaque élément de la matrice est calculé sous un modulo
défini.
 *
 * Elle offre les fonctionnalités suivantes :
 * - Initialisation des valeurs de la matrice (aléatoirement ou manuellement).
 * - Réalisation d'opérations sur deux matrices à l'aide de l'interface MatrixOperation.
 * - Affichage du contenu de la matrice.
 * - Validation des indices et gestion des valeurs en tenant compte du modulo.
 *
 * @throws IllegalArgumentException Si les dimensions de la matrice ou le modulo sont invalides.
 * @throws RuntimeException Si les indices utilisés pour accéder aux éléments de la matrice sont hors des
limites.
 */
public class Matrix {
    final private int[][] data;
    final private int rows;
```

```
    final private int cols;
    final private int modulo;

    /**
     * @brief Constructeur de la classe Matrix.
     *
     * @param rows Le nombre de lignes de la matrice.
     * @param cols Le nombre de colonnes de la matrice.
     * @param modulo Le modulo à utiliser pour les opérations sur la matrice.
     *
     * @throws IllegalArgumentException Si le nombre de lignes, de colonnes ou le modulo est invalide (non
positif).
     *
     * Ce constructeur initialise une matrice vide avec les dimensions spécifiées,
     * et définit le modulo pour les opérations futures sur la matrice.
     */
    public Matrix(int rows, int cols, int modulo) {
        if (rows < 0 || cols < 0 || modulo <= 0) {
            throw new IllegalArgumentException("Valeurs incorrectes.");
        }
        this.rows = rows;
        this.cols = cols;
        this.modulo = modulo;
        this.data = new int[rows][cols];
    }

    /**
     * @brief Initialise la matrice soit avec des valeurs aléatoires, soit avec des valeurs saisies par l'
utilisateur.
     *
     * @param isRandom Si true, la matrice est remplie de valeurs aléatoires ; sinon, les valeurs sont saisies
manuellement par l'utilisateur.
     * @param scanner Scanner pour lire les entrées de l'utilisateur si l'initialisation n'est pas aléatoire.
     *
     * Cette méthode parcourt chaque élément de la matrice pour l'initialiser avec une valeur soit aléatoire,
soit
     * fournie par l'utilisateur. Les valeurs sont réduites modulo le paramètre défini.
     */
    public void randomMatrixInitiation(boolean isRandom, Scanner scanner) {
        for (int i = 0; i < rows; ++i) {
            for (int j = 0; j < cols; ++j) {
                if (isRandom){
                    setValue(i, j, (int) (Math.random() * modulo));
                } else {
                    System.out.print("Entrez une valeur pour la case [" + i + "]" + "[" + j + "]: ");
                    int value = scanner.nextInt();
                    setValue(i, j, value);
                }
            }
        }
    }

    /**
     * @brief Effectue une opération entre deux matrices et retourne le résultat.
     *
     * @param other La matrice avec laquelle l'opération sera effectuée.
     * @param operation L'opération à appliquer (par exemple, addition, soustraction).
     * @return Une nouvelle matrice résultant de l'opération.
     *
     * @throws RuntimeException Si les dimensions des matrices sont différentes.
     *
     * Cette méthode permet d'appliquer une opération donnée à deux matrices. Le résultat est retourné dans
     * une nouvelle matrice, et chaque élément est calculé sous le modulo défini pour cette matrice.
     */
    public Matrix operate(Matrix other, MatrixOperation operation) {
        int maxRows = rows, maxCols = rows;
```

```
        if (rows != other.rows || cols != other.cols) {
            maxRows = Math.max(rows, other.rows);
            maxCols = Math.max(cols, other.cols);
        }
        Matrix result = new Matrix(maxRows, maxCols, modulo);
        for (int i = 0; i < maxRows; ++i) {
            for (int j = 0; j < maxCols; ++j) {
                int newValue = (operation.apply(getValue(i, j), other.getValue(i, j)) % modulo + modulo) %
modulo;
                result.setValue(i, j, newValue);
            }
        }
        return result;
    }

    /**
     * @brief Définit la valeur d'un élément dans la matrice en tenant compte du modulo.
     *
     * @param row L'indice de la ligne de l'élément.
     * @param col L'indice de la colonne de l'élément.
     * @param value La valeur à définir pour l'élément.
     *
     * @throws RuntimeException Si les indices sont hors des limites de la matrice.
     *
     * Cette méthode met à jour un élément spécifique de la matrice. La valeur est réduite modulo le paramètre
     * modulo de la matrice avant d'être stockée.
     */
    public void setValue(int row, int col, int value) {
        if (row >= 0 && row < rows && col >= 0 && col < cols) {
            data[row][col] = value % modulo;
        } else {
            throw new RuntimeException("Index out of range");
        }
    }

    /**
     * @brief Retourne la valeur d'un élément à un indice donné dans la matrice.
     *
     * @param row L'indice de la ligne de l'élément.
     * @param col L'indice de la colonne de l'élément.
     * @return La valeur de l'élément à la position (row, col), ou 0 si l'indice est hors des limites.
     *
     * Cette méthode récupère la valeur d'un élément en vérifiant d'abord si les indices sont dans les
     * limites de la matrice. Si les indices sont valides, elle retourne la valeur ; sinon, elle retourne 0.
     */
    public int getValue(int row, int col) {
        if (row >= 0 && row < rows && col >= 0 && col < cols) {
            return data[row][col];
        } else {
            return 0;
        }
    }

    /**
     * @brief Retourne la valeur du modulo utilisé dans la matrice.
     *
     * @return Le modulo utilisé pour les opérations dans cette matrice.
     *
     * Cette méthode retourne simplement la valeur du modulo défini pour cette matrice, utilisé pour toutes
     * les opérations sur les éléments de la matrice.
     */
    public int getModulo() {
        return modulo;
    }

    /**
```

```
     * @brief Affiche le contenu de la matrice.
     *
     * Cette méthode parcourt chaque élément de la matrice et l'affiche dans la console sous forme de tableau.
     * Chaque ligne de la matrice est affichée sur une ligne séparée.
     */
    public void display() {
        for (int[] row : data) {
            for (int value : row) {
                System.out.print(value + " ");
            }
            System.out.println();
        }
    }
}
```

src/main/java/ch/heigvd/dai/MatrixOperation.java

```
/**
 * @author Aubry Antoine
 * @author Faria dos Santos Dani Tiago
 */

package ch.heigvd.dai;

/**
 * @interface MatrixOperation
 *
 * @brief Interface définissant une opération sur deux éléments d'une matrice.
 *
 * Cette interface sert de base pour implémenter différentes opérations sur les matrices
 * (par exemple, addition, soustraction, multiplication). Elle définit une méthode
 * apply qui doit être implémentée pour spécifier le comportement de l'opération entre deux entiers.
 */
public interface MatrixOperation {
    int apply(int a, int b);
}
```

src/main/java/ch/heigvd/dai/Multiplication.java

```
/**
 * @author Aubry Antoine
 * @author Faria dos Santos Dani Tiago
 */

package ch.heigvd.dai;

/**
 * @class Addition
 *
 * @brief Implémente l'opération de multiplication entre deux éléments d'une matrice.
 *
 * Fournit une implémentation concrète de l'interface MatrixOperation,
 * en définissant la méthode apply qui réalise la multiplication de deux entiers.
 */
public class Multiplication implements MatrixOperation {
    public int apply(int a, int b) {
        return a * b;
    }
}
```

src/main/java/ch/heigvd/dai/Subtraction.java

```
/**
 * @author Aubry Antoine
 * @author Faria dos Santos Dani Tiago
 */
```

```java
package ch.heigvd.dai;

/**
 * @class Addition
 *
 * @brief Implémente l'opération de soustraction entre deux éléments d'une matrice.
 *
 * Fournit une implémentation concrète de l'interface MatrixOperation,
 * en définissant la méthode apply qui réalise la soustraction de deux entiers.
 */
public class Subtraction implements MatrixOperation {
    public int apply(int a, int b) {
        return a - b;
    }
}
```