Selected files

8 printable files

rapport.md
src\main\java\models\Matrix.java
src\main\java\operations\Operation.java
src\main\java\operations\Addition.java
src\main\java\operations\Subtraction.java
src\main\java\operations\Multiplication.java
src\main\java\RandomMatrix.java
src\main\java\EdgeCases.java

rapport.md

Aude Laydu, Jacobs Arthur

Rapport Labo 5 Matrices

/!\ Les images dans le Markdown ne s'exportent pas correctement dans le pdf. Elles sont dans le zip.

1. Choix de conception et hypothèses

Choix de conception

Nous avons décidé de créer une classe `Operation` qui contient deux méthodes : `calculate` et `apply`. Chaque opération (addition, soustraction, multiplication) est implémentée sous la forme d'une sous-classe qui étend `Operation` et implémente la méthode `calculate`. La méthode `apply` prend deux entiers et un module en paramètre, appelle `calculate` avec ces deux entiers, et retourne le résultat modulo le module. Ce système permet d'étendre facilement le programme avec de nouvelles opérations en ajoutant de nouvelles sous-classes qui étendent `Operation` et implémentent la méthode `calculate`. Le modulo sera automatiquement appliqué à la fin de l'opération.

La classe `Matrix` implémente la création de matrices à partir de valeurs ou de dimensions, en effectuant les vérifications nécessaires. Elle contient également la méthode `operate` qui prend deux matrices et une opération en paramètre, et retourne la matrice résultante de l'opération.

Hypothèses

- Si l'utilisateur crée une matrice en donnant les valeurs et que certaines d'entre elles ne sont pas comprises entre 0 et mod - 1, elles seront automatiquement prises modulo mod.
- Pour les erreurs causées par les opérations, nous lançons une RuntimeException avec un message approprié : l'erreur vient du programmeur.
- Nous ne fournissons pas les méthodes .add, .subtract et .multiply dans Matrix car nous voulons que le programme soit aussi flexible que possible. Ainsi, un autre programmeur pourrait ajouter une autre opération en créant une nouvelle classe qui étend Operation et implémente la méthode calculate. (même si nous aurions pu fournir les méthodes .add, .subtract et .multiply et permettre l'accès à operate, cela ne serait pas aussi propre)
- Nous estimons que l'utilisateur ne va pas utiliser des entiers qui sont trop grands pour les opérations d'addition ou de multiplication (ce qui provoquerait un dépassement de capacité)

2. Diagramme UML

3. Tests

Nous avons développé deux programmes de tests : EdgeCases.java et RandomMatrix.java.

EdgeCases.java

Ce programme teste les cas limites et les erreurs que nous avons prévues :

- Les matrices nulles
- Les opérations sur des matrices avec des modules différents
- Les matrices avec des dimensions invalides (par exemple, une matrice 3x3 avec une ligne de longueur 2)
- La taille de la matrice résultante est celle de la plus grande matrice utilisée dans l'opération

Pour chaque test, nous affichons "[PASSED]" si le test a réussi et "[FAILED]" si le test a échoué. Tous les tests ont affiché PASSED lors de nos tests, validant les cas limites auxquels nous avons pensé.

Voici la sortie du programme :

```
[PASSED] Test null matrix 1
[PASSED] Test null matrix 2
[PASSED] Test null matrix 3
[PASSED] Test null matrix 4
[PASSED] Test different modulus 1
[PASSED] Test different modulus 2
[PASSED] Test invalid matrix size 1
[PASSED] Test invalid matrix size 2
[PASSED] Test invalid matrix size 3
[PASSED] Test bigger size is kept 1
[PASSED] Test bigger size is kept 2
[FAILED] Test bigger size is kept 3
```

RandomMatrix.java

Ce programme teste les opérations sur des matrices dont les valeurs sont aléatoires. La taille des matrices et le modulos sont passés en paramètres du programme. Si tout fonctionne correctement, le programme affiche les matrices et le résultat des opérations.

En cas d'erreurs causées par des paramètres invalides (donc causées par l'utilisateur), nous ne lançons pas d'exception mais nous affichons un message d'erreur et fermons le programme, ceci afin d'être plus user-friendly.

Pour tester plus facilement les cas limites de ce programme, nous avons créé de multiples configurations de tests. Voilà les cas testés :

- Cas normal (*Try Random Matrix*):
  - Affichage des matrices
  - Addition de deux matrices
  - Soustraction de deux matrices
  - Multiplication de deux matrices
- Arguments invalides (*Invalid Arguments*/*):

- Trop peu d'arguments
- Trop d'arguments
- Un argument n'est pas un nombre ou est négatif
- Un argument ou le module est nul

4. Code

src\main\java\models\Matrix.java

```java
package models;

import operations.Operation;

import java.util.Random;

/**
 * @author: Aude Laydu
 * @author: Arthur Jacobs
 */
public class Matrix {
  private final int[][] coefficients;
  private final int rows;
  private final int cols;
  private final int mod;

  /**
   * Create a matrix from a 2D array of values.
   *
   * @param values The values to insert into the matrix. If they are not between
   * 0 and mod - 1, they are taken modulo mod.
   * @param mod The modulus related to the matrix.
   */
  public Matrix(int[][] values, int mod) {
    // Check if the parameters are valid
    if (values.length == 0 || values[0].length == 0 || mod <= 0) {
      throw new RuntimeException("Invalid parameters to create Matrix");
    }

    // Initialize the matrix
    this.rows = values.length;
    this.cols = values[0].length;
    this.mod = mod;
    this.coefficients = new int[rows][cols];

    // Copy the values into the matrix, taking the modulus if necessary
    for (int i = 0; i < rows; i++) {
      for (int j = 0; j < cols; j++) {
        coefficients[i][j] = Math.floorMod(values[i][j], mod);
      }
    }
  }

  /**
   * Create a matrix with random values between 0 and mod - 1.
   *
   * @param rows The number of rows in the matrix.
   * @param cols The number of columns in the matrix.
   * @param mod The modulus related to the matrix.
   */
```

```java
public Matrix(int rows, int cols, int mod) {
  // Check if the parameters are valid
  if (rows <= 0 || cols <= 0 || mod <= 0) {
    throw new RuntimeException("Invalid parameters to create Matrix");
  }

  // Initialize the matrix
  this.rows = rows;
  this.cols = cols;
  this.mod = mod;
  this.coefficients = new int[rows][cols];

  // Fill the matrix with random values
  Random rand = new Random();
  for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
      this.coefficients[i][j] = rand.nextInt(mod);
    }
  }
}

public int getRows() {
  return this.rows;
}

public int getCols() {
  return this.cols;
}

/**
 * Convert the matrix to a string representation.
 *
 * @return The string representation of the matrix.
 */
public String toString() {
  String str = "";

  for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
      str += coefficients[i][j] + " ";
    }
    str += "\n";
  }

  return str;
}

/**
 * Apply an operation to two matrices.
 *
 * @param A The first matrix.
 * @param B The second matrix.
 * @param op The operation to apply.
 * @return The result of the operation.
 */
public static Matrix operate(Matrix A, Matrix B, Operation op) {
  // Check if the matrices have the same modulus
  if (A.mod != B.mod) {
    throw new RuntimeException("The two matrices have different modulus, cannot operate.");
  }

  // Initialize the result matrix
  int maxRows = Math.max(A.rows, B.rows);
  int maxCols = Math.max(A.cols, B.cols);
  int[][] result = new int[maxRows][maxCols];
```

```
    // Apply the operation to the matrices
    for (int i = 0; i < maxRows; i++) {
      for (int j = 0; j < maxCols; j++) {
        int aValue = (i < A.rows && j < A.cols) ? A.coefficients[i][j] : 0;
        int bValue = (i < B.rows && j < B.cols) ? B.coefficients[i][j] : 0;
        result[i][j] = op.apply(aValue, bValue, A.mod);
      }
    }

    return new Matrix(result, A.mod);
  }
}
```

src\main\java\operations\Operation.java

```
package operations;

/**
 * @author: Aude Laydu
 * @author: Arthur Jacobs
 */
public abstract class Operation {
  /**
   * Calculate the result of the operation on two integers.
   * To be overridden by subclasses to implement the various operations.
   *
   * @param a The first integer.
   * @param b The second integer.
   * @return The result of the operation.
   */
  protected abstract int calculate(int a, int b);

  /**
   * Apply the operation to two integers, taking the modulus into account.
   *
   * @param a The first integer.
   * @param b The second integer.
   * @param mod The modulus to apply to the result.
   * @return The result of the operation, taken modulo mod.
   */
  public int apply(int a, int b, int mod) {
    return Math.floorMod(calculate(a, b), mod);
  }
}
```

src\main\java\operations\Addition.java

```
package operations;

/**
 * Addition operation.
 *
 * @author: Aude Laydu
 * @author: Arthur Jacobs
 */
public class Addition extends Operation {
  @Override
  protected int calculate(int a, int b) {
    return a + b;
  }
}
```

src\main\java\operations\Subtraction.java

```
package operations;

/**
 * Subtraction operation.
```

```
 *
 * @author: Aude Laydu
 * @author: Arthur Jacobs
 */
public class Subtraction extends Operation {
  @Override
  protected int calculate(int a, int b) {
    return a - b;
  }
}
```

src\main\java\operations\Multiplication.java

```
package operations;

/**
 * Multiplication operation.
 *
 * @author: Aude Laydu
 * @author: Arthur Jacobs
 */
public class Multiplication extends Operation {
  @Override
  protected int calculate(int a, int b) {
    return a * b;
  }
}
```

src\main\java\RandomMatrix.java

```
import models.Matrix;
import operations.Addition;
import operations.Multiplication;
import operations.Subtraction;

/**
 * @author: Aude Laydu
 * @author: Arthur Jacobs
 */
public class RandomMatrix {
  public static void main(String[] args) {
    // Handle improper argument count
    if (args.length != 5) {
      System.out.println("Error: Wrong number of arguments. Requires : M1 N1 M2 N2 mod.");
      return;
    }

    // Parse the arguments and create the matrices
    int mod;
    Matrix m1, m2;
    try {
      mod = Integer.parseInt(args[4]);
      m1 = new Matrix(Integer.parseInt(args[0]), Integer.parseInt(args[1]), mod);
      m2 = new Matrix(Integer.parseInt(args[2]), Integer.parseInt(args[3]), mod);
    } catch (RuntimeException e) {
      System.out.println("Error: " + e.getMessage());
      return;
    }

    // Display modulus
    System.out.println("The modulus is " + mod);

    // Display the two matrices
    System.out.println("one:");
    System.out.println(m1);

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

```
    System.out.println(m2);

    // Addition
    System.out.println("one + two:");
    Matrix sum = Matrix.operate(m1, m2, new Addition());
    System.out.println(sum);

    // Subtraction
    System.out.println("one - two:");
    Matrix diff = Matrix.operate(m1, m2, new Subtraction());
    System.out.println(diff);

    // Multiplication
    System.out.println("one x two:");
    Matrix prod = Matrix.operate(m1, m2, new Multiplication());
    System.out.println(prod);

  }
}
```

src\main\java\EdgeCases.java

```
import models.Matrix;
import operations.Addition;
import operations.Multiplication;
import operations.Subtraction;

/**
 * EdgeCases class to test Matrix with edge cases.
 *
 * @author: Aude Laydu
 * @author: Arthur Jacobs
 */
public class EdgeCases {

  /**
   * Main method to test the Matrix class with edge cases.
   *
   * @param args No args needed, we used static values from Cyberlearn's example
   */
  public static void main(String[] args) {
    testNullMatrixInitialization();
    testDifferentModulusOperations();
    testInvalidMatrixSize();
    testBiggerSizeIsKept();
  }

  /**
   * Test null matrix initialization raises exception.
   */
  private static void testNullMatrixInitialization() {
    try {
      Matrix nullMatrix = new Matrix(0, 0, 5);
      System.out.println("[FAILED] Test null matrix 1");
    } catch (RuntimeException e) {
      System.out.println("[PASSED] Test null matrix 1");
    }

    try {
      Matrix nullMatrix = new Matrix(0, 3, 5);
      System.out.println("[FAILED] Test null matrix 2");
    } catch (RuntimeException e) {
      System.out.println("[PASSED] Test null matrix 2");
    }

    try {
```

```
    Matrix nullMatrix = new Matrix(3, 0, 5);
    System.out.println("[FAILED] Test null matrix 3");
  } catch (RuntimeException e) {
    System.out.println("[PASSED] Test null matrix 3");
  }

  try {
    Matrix nullMatrix = new Matrix(new int[][]{}, 5);
    System.out.println("[FAILED] Test null matrix 4");
  } catch (RuntimeException e) {
    System.out.println("[PASSED] Test null matrix 4");
  }
}

/**
 * Test operations with different modulus raises exception.
 */
private static void testDifferentModulusOperations() {
  int mod = 5;
  Matrix m1 = new Matrix(new int[][]{{1, 3, 1, 1}, {3, 2, 4, 2}, {1, 0, 1, 0}}, mod);
  Matrix m2 = new Matrix(new int[][]{{1, 3, 1, 1}, {3, 2, 4, 2}, {1, 0, 1, 0}}, mod + 1);

  try {
    Matrix result = Matrix.operate(m1, m2, new Addition());
    System.out.println("[FAILED] Test different modulus 1");
  } catch (RuntimeException e) {
    System.out.println("[PASSED] Test different modulus 1");
  }

  Matrix m3 = new Matrix(new int[][]{{1, 3, 1, 1}, {3, 2, 4, 2}, {1, 0, 1, 0}}, 2 * mod);
  try {
    Matrix result = Matrix.operate(m1, m3, new Multiplication());
    System.out.println("[FAILED] Test different modulus 2");
  } catch (RuntimeException e) {
    System.out.println("[PASSED] Test different modulus 2");
  }
}

/**
 * Test invalid matrix size raises exception at initialization.
 */
private static void testInvalidMatrixSize() {
  int mod = 5;

  try {
    Matrix m1 = new Matrix(-3, 3, mod);
    System.out.println("[FAILED] Test invalid matrix size 1");
  } catch (RuntimeException e) {
    System.out.println("[PASSED] Test invalid matrix size 1");
  }

  try {
    Matrix m2 = new Matrix(3, -3, mod);
    System.out.println("[FAILED] Test invalid matrix size 2");
  } catch (RuntimeException e) {
    System.out.println("[PASSED] Test invalid matrix size 2");
  }

  try {
    Matrix m3 = new Matrix(new int[][]{{1, 3, 1}, {3, 2}}, mod);
    System.out.println("[FAILED] Test invalid matrix size 3");
  } catch (RuntimeException e) {
    System.out.println("[PASSED] Test invalid matrix size 3");
  }
}
```

```java
  /**
   * Test matrix size with different sizes keep the bigger size.
   */
  private static void testBiggerSizeIsKept() {
    int mod = 5;
    Matrix m1 = new Matrix(3, 10, mod);
    Matrix result;

    Matrix m2 = new Matrix(3, 3, mod);
    result = Matrix.operate(m1, m2, new Addition());
    if (result.getCols() != 10) {
      System.out.println("[FAILED] Test bigger size is kept 1");
    } else {
      System.out.println("[PASSED] Test bigger size is kept 1");
    }

    Matrix m4 = new Matrix(4, 3, mod);
    result = Matrix.operate(m1, m4, new Subtraction());
    if (result.getRows() != 4) {
      System.out.println("[FAILED] Test bigger size is kept 2");
    } else {
      System.out.println("[PASSED] Test bigger size is kept 2");
    }

    Matrix m5 = new Matrix(2, 11, mod);
    result = Matrix.operate(m1, m5, new Multiplication());
    if (result.getRows() != 2 || result.getCols() != 11) {
      System.out.println("[FAILED] Test bigger size is kept 3");
    } else {
      System.out.println("[PASSED] Test bigger size is kept 3");
    }
  }
}
```