

Table of Contents

Table of Contents	1
Modélisation	2
Choix de conception	2
Considérations techniques	2
Classes du programme	3
Classe Main.....	3
Classe Matrix	3
Classe abstraite BinaryOperation	4
Classes de test	5
Classe MatrixTest.....	5
Classe BinaryOperationTest	5

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 $[0, \text{module} - 1]$, 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.

```
> java Main 2 4 4 2 4

The modulus is 4.

one:
3 0 3 2
3 2 3 3

two:
0 1
0 0
1 1
0 0

one + two:
3 1 3 2
3 2 3 3
1 1 0 0
0 0 0 0

one - two:
3 3 3 2
3 2 3 3
3 3 0 0
0 0 0 0

one x two:
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0

/mnt/c/U/g/Nex/D/I/H/Année 3/Semestre 1/P/L/P00_2023_Lab5/t/classes main
```

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.

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

utilise>

1

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