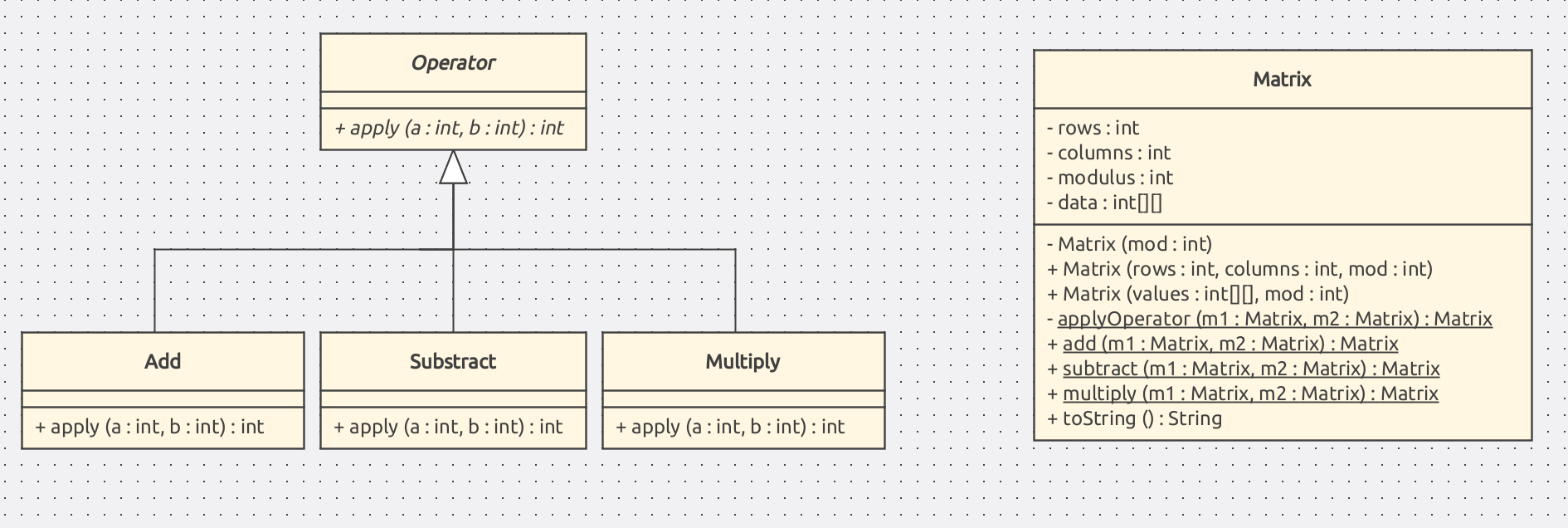
Rapport Laboratoire 5 : Matrices

# Introduction

Dans le cadre ce laboratoire n°5, nous avons dû implémenter une classe permettant la modélisation de matrices à deux dimensions de taille quelconque. De plus, il doit être possible d’effectuer diverses opérations arithmétiques sur deux matrices, composante par composante.

# Diagramme UML



# Choix de modélisation et d’implémentation

## Matrix

#### Divers

Les divers attributs de la classe ont été implémentés en *private* car leur accès au dehors de la classe-même n’est pas nécessaire et casserait l’encapsulation. Nous n’avons pas créé de getters car l’accès des différents attributs n’est pas nécessaire en dehors de la classe pour les besoins du projet. C’est une fonctionnalité qu’il serait facile et rapide d’ajouter en cas de besoin.

L’attribut représentant le modulo de la classe a été noté comme *final* pour spécifié qu’il sera défini qu’une fois via le constructeur.

#### Constructeurs

Nous avons créé un constructeur privé prenant uniquement un modulo afin de factoriser le test que sa valeur corresponde à nos attentes dans les deux autres constructeurs. Ce constructeur est privé car il ne fait pas sens de le rendre accessible aux utilisateurs de la classe étant donné qu’il n’instancie pas une matrice complète, utilisable.

Dans le constructeur contenant des valeurs déjà définies pour la matrice, nous avions tout d’abord réalisé une copie du tableau via la méthode *Arrays.copyOf* afin d’éviter que le contenu de la matrice soit modifié en dehors de la classe, ce qui casserait encore une fois l’encapsulation. Par la suite nous avons remarqué que chaque valeur devait être vérifiée afin d’être sûr qu’elle soit dans l’intervalle [0, modulo - 1]. Nous avons donc remplacé le *Arrays.copyOf* par une simple double boucle.

#### Appliquer les opérations

Les différentes méthodes des opérations entre deux matrices ont été créées comme static car elles retournent toutes une nouvelle instance de matrice contenant le résultat sans toucher aux deux matrices initiales. Il nous a donc semblé plus naturelle de passer les deux matrices en paramètres et de ce fait la méthode pouvait passer static.

Afin de ne pas avoir à créer un constructeur supplémentaire, nous vérifions que la composante des différentes matrices existe avant de l’utiliser dans l’opération. Dans le cas contraire, la valeur 0 est utilisée par défaut. Ceci nous permet de gérer les cas où les deux matrices ne sont pas de taille similaire.

Une nouvelle instance de l’*Operator* voulu est créée à chaque nouvelle opération entre deux matrices. Il aurait également été possible de stocker une instance unique en static dans la classe Matrix par exemple afin d’éviter ces instanciations inutiles. Mais étant donné le faible coût de l’opération (la classe *Operator* est très petite) et le fait que nous ne réalisons pas cette instanciation de nombreuses fois dans une boucle par exemple, nous sommes restés sur l’implémentation plus simple.

#### toString()

L’utilisation d’une instance de *StringBuilder* nous permet de ne pas créer une multitude de littéraux avant de retourner le résultat final.

## Operator

Nous n’avons pas grand-chose à dire à propos de ces classes *Operator*. Il aurait peut-être été possible d’ajouter un niveau d’abstraction en créant deux classes abstraites supplémentaires *UnaryOperator* et *BinaryOperator* dans un souci d’évolutivité (ex : opération d’inversion d’une matrice). Mais cela nous a semblé nous éloigner du cahier des charges initial.

# Tests effectués

Nous avons tout d’abord réalisé le test proposé dans la donnée, afin d’être sûr que nos différentes opérations fonctionnaient avec deux matrices de tailles différentes.

Par la suite, nous avons testé que les différents cas limites étaient bien pris en compte par notre programme. Voici la liste des tests réalisés :

|  |  |
| --- | --- |
| **Test** | **Résultat attendu** |
| Création d’une matrice avec un modulo nul ou négatif | Exception lancée dans le constructeur |
| Création d’une matrice avec un nombre de lignes ou de colonnes négatif | Exception lancée dans le constructeur |
| Opération entre deux matrices ne possédant pas le même modulo | Exception lancée lors de l’opération |
| Création d’une matrice avec des lignes de différentes tailles | Exception lancée dans le constructeur |
| Création d’une matrice avec des valeurs en dehors de l’intervalle [0, modulo - 1] | Exception lancée dans le constructeur |
| Création et opération sur une matrice vide | L’opération se déroule sans soucis. |
| Opération sur des matrices de tailles différentes | L’opération se déroule sans soucis, des 0 sont ajoutés au besoin. |