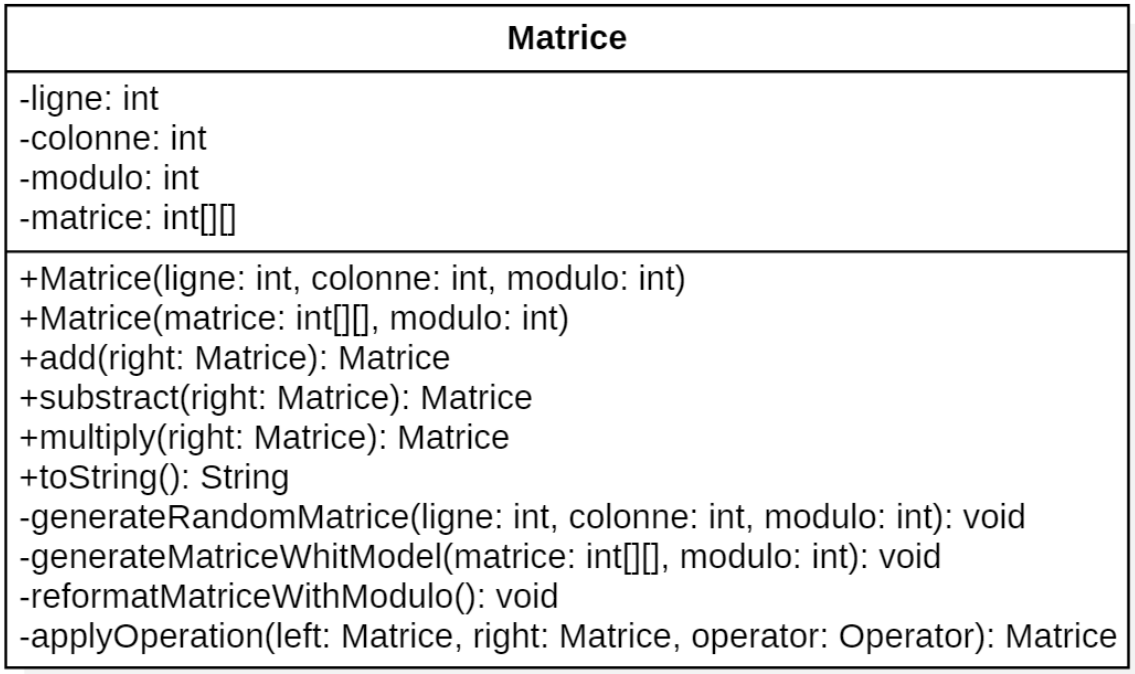
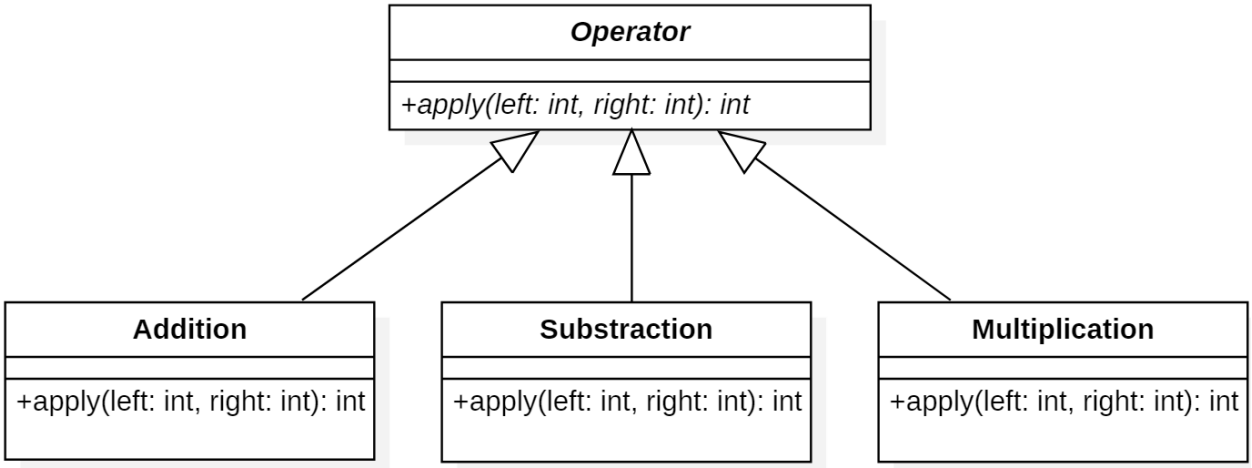
1) UML

1.1) Matrice



Il s'agit d'une classe publique permettant de créer une matrice d'entier.

1.2) Operator



La classe Operator est une classe abstraite car nous ne voulons pas qu'un Operator soit instanciable car il n'est pas concret. En effet, nous voulons instancier les classes concrètes (classes enfants) dans le but de pouvoir additionner, soustraire et multiplier.

2) Justification des choix

2.1) Matrice

Attributs : Nous avons décidé de mettre les propriétés en private afin de respecter le principe d'encapsulation et d'utiliser des getter et des setters si nécessaire (ce qui n'est pas le cas pour ce laboratoire).

Constructeur : Nous avons créé deux constructeurs. Le premier permet de créer une matrice en spécifiant le nombre de ligne, de colonne ainsi que le modulo. La matrice est générée de manière aléatoire à l'aide de la fonction privée « generateRandomMatrice() », qui utilise Math.random() ;

Le deuxième constructeur permet de créer une matrice identique à celle qui est passée en paramètre. Pour ce faire, le constructeur à besoins de deux choses, une matrice et le modulo. La construction se fait grâce à l'utilisation de la fonction privée « generateMatriceWhitModel() ». Cette fonction s'assure que la matrice qui est passée en paramètre d'entrée dans le constructeur ne comporte pas d'élément (entier) supérieur ou égal au modulo qui est également transmis en paramètre d'entrée du constructeur.

Méthodes add(), substract() et multiply() : Ces méthodes prennent une matrice en paramètre d'entrée et retournent une Matrice également. Lors d'une opération, il faut créer une nouvelle matrice afin de sauver le résultat de l'opération sans modifier les deux matrices qui sont additionnées, soustraites ou multipliées. Nous avons décidé de passer qu'une seule matrice en paramètre d'entrée de ces fonctions car la deuxième est la matrice qui appelle la méthode (this).

Ces méthodes sont codées de la façon suivante : il y a un try catch (RuntimeException) afin de remonter toute éventuelle exception. Dans le try nous appelons la méthode « applyOperation() ». C'est cette dernière qui va permettre de réaliser une opération quelconque entre deux matrices.

Méthode toString() : Permet d'afficher une matrice comme dans la donnée du laboratoire.

Méthode reformatMatriceWithModulo() : Cette fonction permet de mettre la matrice au bon format (selon le modulo) en utilisant la fonction Math.floorMod().

Méthode applyOperation() : Cette méthode permet de réaliser une des trois opérations entre deux matrices. En paramètre d'entrée de la fonction, un Operator doit être transmis. C'est ce dernier qui va définir l'opération effectuée. N'oublions pas que la classe Operator est une classe abstraite et donc non instanciable. De ce fait, nous devons transmettre en paramètre d'entrée une classe enfant (addition, soustraction, multiplication). De plus, il est important de noter que cette méthode retourne une

nouvelle matrice. Cette dernière est donc créée dans la méthode et le résultat de l'opération entre les deux matrices est écrit dans cette nouvelle matrice. Cela permet de ne pas modifier les matrices utilisées lors des différentes opérations.

2.2) Operator

Généralités : Comme expliqué ci-dessus, la classe Operator ne doit pas pouvoir être instancié, elle est donc abstraite. C'est pourquoi elle n'a ni attributs ni constructeurs.

Méthode apply() : Il s'agit d'une méthode publique et abstraite qu'il faut redéfinir (overrider) dans les classes enfants. Nous avons décidé de faire cette méthode en publique et non protégée car si un jour nous utilisons des paquetages, il suffit que le paquetage de la classe diffère du paquetage où la classe est utilisée (objet instancié) pour que ça ne fonctionne plus.

En effet, un champ protégé d'une classe A est toujours visible à l'intérieur de son paquetage. A l'extérieur de son paquetage, un champ protégé est hérité par une sous-classe B mais non visible au travers d'une instance de A ou de B (sauf pour une instance de B invoquée dans B ou dans une sous-classe de B).[1]

2.2.1) Addition, Subtraction, Multiplication

Généralités : Ces trois classes héritent de la classe abstraite Operator. Comme mentionné plus haut, la classe Operator contient une méthode abstraite. C'est pourquoi ces trois classes concrètes implémentent cette même méthode.

Constructeur : Nous n'avons pas écrit de constructeur car le constructeur par défaut implicite que Java met à disposition (celui qui est défini par le compilateur) suffit amplement pour instancier ces classes.

Méthode apply() : Cette méthode est overrider dans les trois classes enfants afin de pouvoir effectuer l'opération de l'opérateur instancier. Elle retourne un résultat entier et prend en paramètre deux entiers afin d'effectuer une des trois opérations élémentaires.

[1] Définition tirée du site https://perso.telecom-paristech.fr/hudry/coursJava/monPaquet/visibilite.html