TP – Matrice creuse
Utilisation d'une ArrayList

Fichiers disponibles sur Moodle :
- la classe ***Coefficient***
- *TestMatricePartie1.java*, *TestMatricePartie2.java*
- *indications.txt* : contient une partie du code des méthodes *supprimer*, *setValeur* et *afficher*

Le but du TP est d'écrire une classe Java pour représenter une matrice creuse. En mathématiques, une matrice creuse est une matrice généralement d'une grande taille, mais dont la plupart des coefficients sont égaux à 0. Pour optimiser l'occupation mémoire, l'idée est donc de ne stocker que les coefficients non nuls, dans une liste, plus précisément une collection *ArrayList*.

Plus précisément, les coefficients non nuls seront de type ***Coefficient*** (voir la classe disponible sur Moodle). Un objet de type coefficient réunira 3 informations : un numéro de ligne, un numéro de colonne et une valeur de type double. Tous les numéros de ligne et de colonne gérés dans ce TP seront des entiers strictement positifs.

Par exemple, supposons que nous souhaitions stocker une matrice de taille 5 lignes et 5 colonnes (pour simplifier la taille est ici réduite). Dans cette matrice, seuls 3 coefficients sont différents de 0 :

0	**10.5**	0	0	0
0	0	0	0	0
0	0	**6**	0	0
5.3	0	0	0	0
0	0	0	0	0

Plutôt que de stocker les valeurs dans un tableau à 2 dimensions, nous allons les stocker dans une liste de coefficients. Schématiquement, on obtient la représentation suivante :

(1 , 2 , 10.5) → (3 , 3 , 6) → (4 , 1 , 5.3)

Dans chaque triplet, on a indiqué : le numéro de la ligne, le numéro de la colonne et la valeur du coefficient. Ce triplet sera en fait une instance de la classe **Coefficient.**

La classe ***MatriceCreuse*** comportera les méthodes suivantes :

- Un **constructeur** sans argument qui créer par défaut une matrice de 5 lignes et 5 colonnes dont tous les coefficients seront égaux à 0.

- Un **constructeur** avec en argument le nombre de lignes et de colonnes de la matrice. Si l'un des arguments est invalide, le constructeur lèvera l'exception ***IllegalArgumentException***. Un

message d'erreur sera transmis au programme appelant en même temps que l'instance d'exception. A l'initialisation, tous les coefficients de la matrice seront égaux à 0.

- Une **méthode accesseur sur la valeur du coefficient** situé sur une ligne et une colonne données en paramètre. Cette méthode aussi est susceptible de lever l'exception ***IllegalArgumentException***. Egalement, un message d'erreur sera transmis à l'appelant. Il précisera que l'erreur s'est produite lors de l'appel à l'accesseur, et la ou les coordonnées incorrectes.

- Une **méthode permettant de modifier la valeur d'un coefficient** de la matrice. Elle aura en paramètre un numéro de ligne, un numéro de colonne et la nouvelle valeur du coefficient. Attention, il ne faut pas stocker dans la liste un coefficient égal à 0. Cette méthode aussi est susceptible de lever l'exception ***IllegalArgumentException***. Egalement un message d'erreur sera transmis à l'appelant. Il précisera que l'erreur s'est produite lors d'une tentative de modification d'un coefficient de la matrice et la ou les coordonnées incorrectes.

  Pour coder cette méthode, vous pouvez faire appel à une méthode privée de la classe permettant de supprimer un coefficient. Le code de cette méthode est écrit dans le fichier *indications.txt*. Toutefois il manque une instruction pour qu'il soit complet.

- Une méthode permettant **d'afficher les coefficients non nuls** de la matrice, ainsi que leurs coordonnées. Il s'agit bien d'afficher sur la console et pas de renvoyer une chaîne de caractères. Cette méthode vous sera utile pour tester le bon fonctionnement d'autres méthodes.

Partie 1

1) Ecrire la classe ***MatriceCreuse*** en commençant par définir les attributs et les constructeurs.

2) Tester le constructeur avec argument grâce à la méthode *testConstructeur* de la classe ***TestMatricePartie1***.

3) Ajouter à la classe les méthodes permettant de

- modifier la valeur d'un coefficient, ***setValeur***, voir le début de la méthode dans le fichier *indications.txt,* voir aussi la méthode *supprimer* qu'il faudra compléter

- d'afficher les coefficients non nuls de la matrice, ***afficher***, voir le début de la méthode dans le fichier *indications.txt*

4) Tester ces 2 méthodes grâce à la méthode :

- *testAfficherSetValeur* : la méthode effectue quelques tests basiques

- *testSetValeur* qui réalise plusieurs tests plus poussés

5) Ajouter à la classe la méthode permettant de consulter la valeur d'un coefficient, ***getValeur***.

6) Tester grâce à la méthode *testGetValeur* de la classe ***TestMatricePartie1.***

Partie 2

La classe permettant de tester les méthodes de la partie 2 est disponible sur Moodle : ***TestMatricePartie2***.

7) Ajouter à la classe une méthode permettant de construire une nouvelle matrice obtenue en **multipliant les coefficients** de la matrice courante par une valeur donnée en paramètre.

⇒ Tester cette méthode.

8) Ajouter à la classe une méthode permettant d'**additionner 2 matrices** pour en obtenir une troisième. Si l'addition des 2 matrices est impossible, l'exception ***IllegalArgumentException*** sera levée. Un message d'erreur sera transmis à l'appelant. Il précisera que l'erreur s'est produite lors de l'addition de 2 matrices, et indiquera la cause précise de l'erreur.

⇒ Tester cette méthode.

9) Ajouter à la classe une méthode permettant de **multiplier 2 matrices** pour en obtenir une troisième. Si la multiplication des 2 matrices est impossible, l'exception ***IllegalArgumentException*** sera levée. Un message d'erreur sera transmis à l'appelant. Il précisera que l'erreur s'est produite lors de la multiplication de 2 matrices, et indiquera la cause précise de l'erreur.

⇒ Tester cette méthode.

Partie 3 (facultatif)

Il s'agit dans cette partie de modifier la représentation interne de la matrice creuse de manière à stocker les coefficients dans l'ordre croissant de leurs coordonnées. Le but évidemment est d'améliorer l'efficacité des méthodes de la classe.

10) Modifier la classe **Coefficient** de manière à la doter d'une relation d'ordre. Il s'agit en fait d'ajouter une opération permettant de comparer la position de 2 coefficients. On écrira donc une méthode ***compareTo***. La valeur du coefficient n'intervient pas dans la comparaison. Seules les lignes et éventuellement les colonnes sont prises en compte.
Par exemple, on aura : $(1, 2, 10.5) < (3, 3, 6) < (4, 1, 5.3)$

⇒ Tester la méthode ***compareTo***.

11) Modifier la classe **MatriceCreuse** pour prendre en compte l'optimisation évoquée.

⇒ Tester les méthodes de la classe ainsi écrite (refaire passer les méthodes de test précédentes)