Programmation Orientée Objet Avancée
– TP 3 –

Exercice :

Une matrice creuse est une matrice qui comprend une grande quantité de valeurs nulles (égales à 0) et seulement quelques valeurs non nulles éparses. On rencontre fréquemment de telles matrices en informatique, en maths, en physique, etc.

La matrice suivante est un exemple de matrice creuse :

$$\begin{pmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & \mathbf{1.0} & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & \mathbf{2.5} & 0.0 & 0.0 & 0.0 \\ 0.0 & \mathbf{0.3} & 0.0 & 0.0 & 0.0 & \mathbf{1.6} \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & \mathbf{0.9} & 0.0 & \mathbf{3.2} & 0.0 \end{pmatrix}$$

On désire réaliser une implémentation simple et efficace des matrices creuses. Pour cela on choisit d'utiliser une représentation chaînée telle que chaque valeur non nulle est indiquée par le triplet (i,j,v) où i et j sont les indices ligne et colonne et v la valeur correspondante.

On obtient alors pour l'exemple ci-dessus la liste suivante :

((1, 2, 1.0), (2, 2, 2.5), (3, 1, 0.3), (3, 5, 1.6), (5, 2, 0.9), (5, 4, 3.2)).

On remarque que la liste n'a pas besoin d'être triée et que la liste suivante représente de façon non ambiguë la même matrice :

((5, 2, 0.9), (2, 2, 2.5), (5, 4, 3.2), (3, 1, 0.3), (1, 2, 1.0), (3, 5, 1.6)).

En ne considérant que le nombre de valeurs numériques stockées en mémoire, et donc en négligeant le coût de gestion de la liste, on constate qu'avec une représentation contiguë on aurait besoin de stocker 36 valeurs tandis qu'avec une représentation chainée 18 (3×6) valeurs seraient suffisantes. L'écart devient vite très important lorsqu'il s'agit de matrices creuses de grande taille.

Pour simplifier l'exercice, on se limitera aux matrices creuses carrées correspondant à l'interface suivante :

```
interface MatriceCarree {
    int taille () ;
    // retourne la taille N de la matrice carrée NxN
    double getValeur (int i, int j) ;
    // retourne la valeur d'indice (i, j)
```

```
    void setValeur (int i, int j, double v) ;
    // affecte v à la position (i, j)
    void add (MatriceCarree m) ;
    // effectue l'addition entre la matrice courante et la matrice m
}
```

On remarque que telle que cette interface est définie, on peut ajouter une matrice à une autre quel que soit son type de réalisation.

Ecrire une classe **MatriceCreuse** qui implémente cette interface selon la seconde méthode, en conservant la liste des triplets (i, j, v) dans un objet LinkedList.

En plus des méthodes nécessaires pour l'interface, on demande de redéfinir et/ou surcharger les méthodes toString () et equals () héritée de la classe Object.

La solution proposée devra être simple et efficace.