```java
package structuredonnees.matrice;
import java.util.ArrayList;

public class MatriceCreuse {

    private final int NB_COLONNES_DEFAULT = 5;

    private final int NB_LIGNES_DEFAULT = 5;

    private int ligne;

    private int colonne;

    private ArrayList<Coefficient> matrice;

    /**
     * Constructeur initialisant par default la matrice 5 lignes par 5 colonnes
     */
    public MatriceCreuse() throws IllegalArgumentException {
        this.ligne = NB_LIGNES_DEFAULT;
        this.colonne = NB_COLONNES_DEFAULT;
        matrice = new ArrayList<Coefficient>();
    }

    /**
     * Constructeur de la matrice
     * @param ligne
     * @param colonne
     */
    public MatriceCreuse(int ligne, int colonne) throws IllegalArgumentException {
        if (ligne < 0 || colonne < 0) {
            throw new IllegalArgumentException("lignes ou colonne invalide");
        }
        this.ligne = ligne;
        this.colonne = colonne;
        matrice = new ArrayList<Coefficient>();
    }


    /**
     * renvoie la valeur du coeficient situé a la position (ligne;colonne)
     * @param ligne
     * @param colonne
     * @return
     */
    public double getValeur(int ligne, int colonne) throws IllegalArgumentException
{
        if (ligne > this.ligne || colonne > this.colonne ||
                    ligne < 0 || colonne < 0) {
            throw new IllegalArgumentException("lignes ou colonne invalide");
        }

        for (Coefficient coefficient : matrice) {
            if (coefficient.getLigne() == ligne && coefficient.getColonne() ==
  colonne) {
                return coefficient.getValeur();
            }
        }

        return 0;
    }

    /**
     * Modifie la valeur d'un coefficent
```

```
64      * @param numLigne numéro de la ligne du coefficient à modifier
65      * @param numColonne numéro de la colonne
66      * @param valeur  nouvelle valeur du coefficient
67      * @throws IllegalArgumentException  levée si un argument est incohérent
68      */
69     public void setValeur(int numLigne, int numColonne, double valeur)
70                                                     throws IllegalArgumentException
71     {
72         if (ligne > this.ligne || colonne > this.colonne ||
73                     ligne < 0 || colonne < 0) {
74             throw new IllegalArgumentException("lignes ou colonne invalide");
75         }
76
77         // TODO : gérer le cas des coordonnées correctes
78
79         this.matrice.add(new Coefficient(numLigne, numColonne, valeur));
80
81
82     }
83
84     /**
85      * Supprime un coefficient de la liste des coefficients.
86      * Si dans la liste des coefficients, il n'y a pas de coefficient situé à la
   ligne
87      * et colonne argument, la méthode est sans effet.
88      * De même si les coordonnées argument sont invalides : méthode sans effet
89      *  (pas d'exception levée dans ce cas, car méthode privée : ce sont les
   méthodes
90      *   appelantes qui vérifieront la validité de la ligne et de la colonne)
91      * @param numLigne numéro de la ligne du coefficient
92      * @param numColonne numéro de la colonne du coefficient
93      * @return un booléen égal à vrai ssi la suppression a été effectuée
94      */
95     private boolean supprimer(int numLigne, int numColonne) {
96         int i;      // indice de parcours de la liste des coefficients
97
98         // parcours de la liste à la recherche d'un coefficient situé à la position
   argument
99         for (i = 0; i < matrice.size()
100                     && ! matrice.get(i).estSitue(numLigne, numColonne); i++);
101
102        if (i < matrice.size()) {
103
104            // coefficient trouvé en position i : il faut le supprimer
105            /* TODO : écrire l'instruction pour supprimer le coefficient en
   position i*/
106
107
108            return true;
109        } else {
110
111            // pas de suppression possible
112            return false;
113        }
114    }
115
116    public static MatriceCreuse multiplication(MatriceCreuse a, MatriceCreuse b) {
117        // TODO Auto-generated method stub
118        return null;
119    }
120
121
122
```

```
123     /**
124      * Affiche les coefficients de la liste
125      */
126     public void afficher() {
127
128
129         if (true /* TODO : remplacer true par la condition qui dit que la liste des
    coefficients est vide */) {
130
131             // cas particulier de la matrice nulle
132             System.out.println("La matrice est nulle.");
133         } else {
134
135             // parcours de la liste dans le but d'afficher tous les coefficients
136             /* TODO : écrire la boucle de parcours de la liste des coefficients
137                       bien remarquer que dans la classe Coefficient, il y a une
    méthode toString */
138
139
140         }
141     }
142
143     public static MatriceCreuse addition MatriceCreuse mat1, MatriceCreuse mat2) {
144         // TODO Auto-generated method stub
145         return null;
146     }
147
148     public MatriceCreuse multiplier(int i) {
149         // TODO Auto-generated method stub
150         return null;
151     }
152
153
154
155
156 }
```