```java
package structuredonnees.matrice;

import java.util.ArrayList;

/**
 * MatriceCreuse
 * permet de creer une matrice creuse
 * et de faire des operations sur les matrices creuses
 */
public class MatriceCreuse {

    private ArrayList<Coefficient> listeCoefficients;

    private static final int LONGEUR_DEFAULT = 5;

    private static final int LARGEUR_DEFAULT = 5;

    private int nbLignes;

    private int nbColonnes;

    /**
     * Constructeur
     */
    public MatriceCreuse() {
        this.nbLignes = this.LONGEUR_DEFAULT;
        this.nbColonnes = this.LARGEUR_DEFAULT;
        this.listeCoefficients = new ArrayList<>();
    }

    /**
     * Constructeur
     * @param nbLignes
     * @param nbColonnes
     * @throws IllegalArgumentException
     */
    public MatriceCreuse(int nbLignes, int nbColonnes) throws IllegalArgumentException {
        if (nbLignes < 0 || nbColonnes < 0) {
            throw new IllegalArgumentException("Nombre de lignes ou de colonnes invalide");
        }
        this.nbLignes = nbLignes;
        this.nbColonnes = nbColonnes;
        this.listeCoefficients = new ArrayList<>();
    }

    /**
     * setValeur
     * @param ligne
     * @param colonne
     * @param valeur
     * @throws IllegalArgumentException
     */
    public void setValeur(int ligne, int colonne, double valeur) throws IllegalArgumentException {
        if (ligne <= 0 || ligne > this.nbLignes || colonne <= 0 || colonne > this.nbColonnes) {
            throw new IllegalArgumentException("Numéro de valeur,  de ligne ou de colonne invalide \nigne utilistateur : " + ligne
                                               + "  | Lignes possibles : 0-" + this.nbLignes + "\nColonne utilistateur : " + colonne
                                               + "  | Colones possibles : 0-" + this.nbColonnes + "\nValeur utilisateur : " + valeur
                                               + "  | Valeurs possibles : n'importe quel entier non nul\n\n");
        } if (valeur == 0.0) {
            System.out.println("La valeur du coefficient ne peut pas etre nulle la valeur na pas \u00e9t\u00e9 modifi\u00e9e");
        } else {
            Coefficient coef = new Coefficient(ligne, colonne, valeur);
```

```
63                  this.listeCoefficients.add(coef);
64              }
65          }
66
67          /**
68           * getValeur
69           * @param ligne
70           * @param colonne
71           * @return
72           * @throws IllegalArgumentException
73           */
74          public double getValeur(int ligne, int colonne) throws IllegalArgumentException {
75              if (ligne <= 0 || ligne > this.nbLignes || colonne <= 0 || colonne > this.
                nbColonnes) {
76                  throw new IllegalArgumentException("Numero de ligne ou de colonne
                    invalide\nLigne utilistateur : " + ligne
77                                                     + "  | Lignes possibles : 0-" + this.
                                                       nbLignes + "\nColonne utilistateur : "

78                                                     + colonne + "  | Colones possibles :
                                                       0-" + this.nbColonnes + "\n\n");
79              }
80              double valeur = 0.0;
81              for (int i = 0; i < this.listeCoefficients.size() && valeur == 0.0; ++i) {
82                  if (this.listeCoefficients.get(i).getLigne() != ligne || this.
                    listeCoefficients.get(i).getColonne() != colonne) {
83                      valeur = this.listeCoefficients.get(i).getValeur();
84                  }
85              }
86              return valeur;
87          }
88
89
90
91          /**
92           * afficher la matrice
93           */
94          public void afficher() {
95              for (int i = 0; i < this.listeCoefficients.size(); ++i) {
96                  System.out.println(this.listeCoefficients.get(i).toString() + "\n");
97              }
98          }
99
100
101         /**
102          * multiplier deux matrices
103          * @param matrice1
104          * @param matrice2
105          * @return
106          */
107         public static MatriceCreuse multiplication(MatriceCreuse matrice1, MatriceCreuse
            matrice2) {
108             if (matrice1.getNBcolonnes() != matrice2.getNBlignes()) {
109                 throw new IllegalArgumentException("Le nombre de colonnes de la premiere
                    matrice doit etre égal au nombre de lignes de la deuxieme matrice");
110             }
111             MatriceCreuse matriceResultat = new MatriceCreuse(matrice1.getNBlignes(),
                matrice2.getNBcolonnes());
112             for (int i = 0; i < matrice1.listeCoefficients.size(); ++i) {
113                 for (int j = 0; j < matrice2.listeCoefficients.size(); ++j) {
114                     if (matrice1.listeCoefficients.get(i).getColonne() == matrice2.
                        listeCoefficients.get(j).getLigne()) {
115                         matriceResultat.setValeur(matrice1.listeCoefficients.get(i).
                            getLigne()
116                                                   , matrice2.listeCoefficients.get(j).
                                                   getColonne()
117                                                   , matriceResultat.getValeur(matrice1.
                                                   listeCoefficients.get(i).getLigne()
118                                                   , matrice2.listeCoefficients.get(j).
                                                   getColonne())
119                                                   + matrice1.listeCoefficients.get(i).
                                                   getValeur() * matrice2.listeCoefficients.get
                                                   (j).getValeur());
```

```
                };

            }
        }
        return matriceResultat;
    }

    /**
     * additionner deux matrices
     * @param matrice1
     * @param matrice2
     * @return
     */
    public static MatriceCreuse addition(MatriceCreuse matrice1, MatriceCreuse
    matrice2) {
        if (matrice1.getNBlignes() != matrice2.getNBlignes() || matrice1.getNBcolonnes
        () != matrice2.getNBcolonnes()) {
            throw new IllegalArgumentException("Les matrices ne sont pas de meme
            taille");
        }
        MatriceCreuse matriceResultat = new MatriceCreuse(matrice1.getNBlignes(),
        matrice1.getNBcolonnes());
        for (int i = 0; i < matrice1.listeCoefficients.size(); ++i) {
            matriceResultat.setValeur(matrice1.listeCoefficients.get(i).getLigne()
                                      , matrice1.listeCoefficients.get(i).getColonne()
                                      , matrice1.listeCoefficients.get(i).getValeur()
                                      + matrice2.getValeur(matrice1.listeCoefficients.
                                      get(i).getLigne()
                                      , matrice1.listeCoefficients.get(i).getColonne
                                      ()));
        }
        return matriceResultat;
    }

    /**
     * multiplier la matrice par un coefficient
     * @param coef
     * @return
     */
    public MatriceCreuse multiplier(int coef) {
        MatriceCreuse matrice = new MatriceCreuse(this.nbLignes, this.nbColonnes);
        for (int i = 0; i < this.listeCoefficients.size(); ++i) {
            matrice.setValeur(this.listeCoefficients.get(i).getLigne()
                              , this.listeCoefficients.get(i).getColonne()
                              , this.listeCoefficients.get(i).getValeur() * (double)
                              coef);
        }
        return matrice;
    }

    /**
     * getNBlignes
     * @return
     */
    public int getNBlignes() {
        return this.nbLignes;
    }

    /**
     * getNBcolonnes
     * @return
     */
    public int getNBcolonnes() {
        return this.nbColonnes;
    }
}
```