File - Test.java

```java
/**
 * Classe de test pour la classe Matrice.
 * Il est nécessaire d'entrer les arguments M1 N1 M2 N2 I
 * Les arguments correspondent respectivement à la heuteur largeur
 * de la première matrice, hauteur largeur de la seconde matrice et
 * le modulo qu'elles respectent.
 * Deux matrices de contenu aléatoire seront générées et testées.
 * D'autres tests sont effectué avec des matrices prédéterminées pour
 * valider les cas limites.
 *
 * @author Forestier Quentin, Herzig Melvyn
 * @version 14.10.2020
 */
public class Test
{


    public static void testStatic()
    {
        int modulus = 5;

        Matrice one = new Matrice(new int[][]{
                {1, 3, 1, 1},
                {3, 2, 4, 2},
                {1, 0, 1, 0}
        }, modulus);

        Matrice two = new Matrice(new int[][]{
                {1, 4, 2, 3, 2},
                {0, 1, 0, 4, 2},
                {0, 0, 2, 0, 2}
        }, modulus);

        System.out.println("The modulus is " + modulus + "\n");

        System.out.println("one");
        System.out.println(one);

        System.out.println("\ntwo");
        System.out.println(two);

        System.out.println("\none + two");
        System.out.println(one.add(two));

        System.out.println("\none - two");
        System.out.println(one.sub(two));

        System.out.println("\none x two");
        System.out.println(one.prod(two));
    }

    public static void testUserInput(int M1, int N1, int M2, int N2,
                                     int modulus)
    {

        Matrice three = new Matrice(M1, N1, modulus);

```

```
58          Matrice four = new Matrice(M2, N2, modulus);
59
60          System.out.println("The modulus is " + modulus + "\n");
61
62          System.out.println("three");
63          System.out.println(three);
64
65          System.out.println("\nfour");
66          System.out.println(four);
67
68          System.out.println("\nthree + four");
69          System.out.println(three.add(four));
70
71          System.out.println("\nthree - four");
72          System.out.println(three.sub(four));
73
74          System.out.println("\nthree x four");
75          System.out.println(three.prod(four));
76      }
77
78      public static void testLimitCase()
79      {
80          try
81          {
82              System.out.println("Test de matrice avec une taille négative :");
83              Matrice negSize = new Matrice(-1, 4);
84
85          }
86          catch (RuntimeException e)
87          {
88              System.out.println(e);
89          }
90
91          System.out.println("\n");
92
93          try
94          {
95              System.out.println("Test de matrice avec un modulo incorrecte :");
96              Matrice badModulus = new Matrice( new int[][]{{1, 2, 3}, {2, 3, 4}},
97                                                0);
98          }
99          catch (RuntimeException e)
100         {
101             System.out.println(e);
102         }
103
104         System.out.println("\n");
105
106         try
107         {
108             System.out.println("Test d'une opération avec un modulo différent" +
109                                " :");
110             Matrice m1 = new Matrice(2, 5);
111             Matrice m2 = new Matrice(5, 4);
112
113             m1.add(m2);
114         }
```

```
115         catch (RuntimeException e)
116         {
117             System.out.println(e);
118         }
119 
120         System.out.println("\n");
121 
122         try
123         {
124             System.out.println("Test création d'une matrice vide + addition :");
125             Matrice m1 = new Matrice(0, 0, 4);
126             Matrice m2 = new Matrice(3, 4, 4);
127 
128             System.out.println(m1);
129             System.out.println(m2);
130 
131             System.out.println(m2.sub(m1));
132         }
133         catch (RuntimeException e)
134         {
135             System.out.println(e);
136         }
137 
138         System.out.println("\n");
139 
140         try
141         {
142             System.out.println("Test création d'une matrice vide à partir " +
143                     "d'un tableau vide");
144             int[][] tbl = {};
145             Matrice vide = new Matrice(tbl, 6);
146 
147             System.out.println(vide);
148         }
149         catch (RuntimeException e)
150         {
151             System.out.println(e);
152         }
153     }
154 
155     /**
156      * Fonction principale de test et entrée du programme.
157      *
158      * @param args Dimensions et module des matrices à créer
159      * @throws IllegalArgumentException Si un argument manque.
160      * @throws NumberFormatException Si les argument ne sont pas des entiers.
161      */
162     public static void main(String[] args)
163     {
164         if (args.length != 5)
165             throw new IllegalArgumentException("Args: M1 N1 M2 N2 module");
166         int M1 = Integer.parseInt(args[0]);
167         int N1 = Integer.parseInt(args[1]);
168         int M2 = Integer.parseInt(args[2]);
169         int N2 = Integer.parseInt(args[3]);
170 
171         int modulus = Integer.parseInt(args[4]);
```

File - Test.java

```
172
173         // Tests selon donnée.
174
175         System.out.println("------------ Test de la donnée ------------");
176         testStatic();
177
178         System.out.println("\n------------ Test de l'entrée utilisateur " +
179                 "------------");
180         testUserInput(M1, N1, M2, N2, modulus);
181
182         System.out.println("\n------------ Test de cas limite ------------");
183         testLimitCase();
184     }
185 }
186
```

File - Matrice.java

```java
/**
 * Class implémentant des matrices de taille personnalisable.
 * Le contenu est défini modulo N.
 * Les matrices peuvent être additionnées, soustraintes et multipliées.
 *
 * @author Forestier Quentin, Herzig Melvyn
 * @version 14.10.2020
 */
public class Matrice
{
    private int modulus;
    private int matrice[][];

    /**
     * Constructeur, génère la matrice d'après les dimensions en paramètre
     * ainsi que le contenu d'après le modulo.
     *
     * @param aHeight  Heuteur de la matrice.
     * @param aWidth   Largeur de la matrice.
     * @param aModulus Modulo du contenu.
     * @throws RuntimeException Dans le cas d'une dimension négative où
     *                          d'un modulo inférieur à 1.
     */
    public Matrice(int aHeight, int aWidth, int aModulus)
    {
        if (aHeight < 0 || aWidth < 0 || aModulus < 1)
            throw new RuntimeException("Argument(s) invalide(s)");

        modulus = aModulus;

        matrice = new int[aHeight][aWidth];

        for (int M = 0; M < aHeight; ++M)
        {
            for (int N = 0; N < aWidth; ++N)
            {
                matrice[M][N] = (int) (Math.random() * modulus);
            }
        }
    }

    /**
     * Constructeur de matrice carrée d'après la taille en paramètre et
     * génère le contenu d'après le modulo.
     *
     * @param size     Hauteur et largeur de la matrice.
     * @param aModulus Modulo du contenu.
     */
    public Matrice(int size, int aModulus)
    {
        this(size, size, aModulus);
    }

    /**
     * Constructeur crée l'objet matrice d'après le tableau bidimensionnel
     * en paramètre. Corrige son contenu d'après le module.
     *
```

```java
     * @param aMatrice Matrice à intégrer.
     * @param aModulus Module à appliquer.
     * @throws RuntimeException Dans le cas d'un modulo inférieur à 1.
     */
    public Matrice(int aMatrice[][], int aModulus)
    {
        if (aModulus < 1)
            throw new RuntimeException("Le modulo est < 1");

        modulus = aModulus;
        matrice = aMatrice;

        // Corrige le contenu pour respecter le modulo.
        for (int M = 0; M < this.getHeight(); ++M)
        {
            for (int N = 0; N < this.getWidth(); ++N)
            {
                matrice[M][N] = Math.floorMod(matrice[M][N], modulus);
            }
        }
    }

    /**
     * Retourne la hauteur de la matrice.
     *
     * @return Hauteur de la matrice.
     */
    public int getHeight()
    {
        return matrice.length;
    }

    /**
     * Retourne la largeur de la matrice
     *
     * @return Largeur de la matrice.
     */
    public int getWidth()
    {

        return matrice.length == 0 ? 0 : matrice[0].length;
    }


    /**
     * Retourne le modulo de la matrice
     *
     * @return Modulo de la matrice
     */
    public int getModulus()
    {
        return this.modulus;
    }

    /**
     * Affiche le contenu de la matrice.
     *
```

```java
     * @return String de la matrice.
     */
    @Override
    public String toString()
    {
        if (getHeight() == 0 || getWidth() == 0) return "Matrice vide";
        StringBuilder result = new StringBuilder();
        for (int M = 0; M < this.getHeight(); ++M)
        {
            for (int N = 0; N < this.getWidth(); ++N)
            {
                result.append(matrice[M][N]);
            }
            result.append('\n');
        }
        return result.toString();

    }

    /**
     * Additionne la matrice courante avec la matrice m.
     *
     * @param m Matrice, seconde opérande.
     * @return Matrice résultante de la somme.
     */
    public Matrice add(Matrice m)
    {
        return operation(m, Operator.ADDITION);
    }

    /**
     * Soustrait la matrice courante avec la matrice m.
     *
     * @param m Matrice, seconde opérande.
     * @return Matrice résultante de la différence.
     */
    public Matrice sub(Matrice m)
    {
        return operation(m, Operator.SUBTRACTION);
    }

    /**
     * Multiple la matrice courante avec la matrice m.
     *
     * @param m Matrice, seconde opérande.
     * @return Matrice résultante du produit.
     */
    public Matrice prod(Matrice m)
    {
        return operation(m, Operator.PRODUCT);
    }

    /**
     * Fonction principale de génération d'une matrice issue de deux matrices
     * combinées, la matrice courante et la matrice m.
     *
     * @param m        Matrice, seconde opérande.
```

```java
     * @param operator Opération à appliquer entre les éléments Amn Bmn.
     * @return Retourne la matrice résultante de l'opération.
     */
    private Matrice operation(Matrice m, Operator operator)
    {
        if (this.modulus != m.modulus)
            throw new RuntimeException("Les modulos ne sont pas égaux");

        Matrice result = new Matrice(Math.max(this.getHeight(), m.getHeight()),
                Math.max(this.getWidth(), m.getWidth()),
                this.modulus);

        int v1, v2;
        for (int M = 0; M < result.getHeight(); ++M)
        {
            for (int N = 0; N < result.getWidth(); ++N)
            {
                // Vérification si la case MxN est dans les tableaux.
                v1 = (M < this.getHeight() && N < this.getWidth()) ?
                        this.matrice[M][N] : 0;
                v2 = (M < m.getHeight() && N < m.getWidth()) ?
                        m.matrice[M][N] : 0;

                result.matrice[M][N] = Math.floorMod(operator.apply(v1, v2),
                        modulus);
            }
        }
        return result;
    }
}

/**
 * Enum des opérations applicables sur les Matrices
 *
 * @author Forestier Quentin, Herzig Melvyn
 * @version 14.10.2020
 *
 * Inspiré de: https://stackoverflow.com/a/2902471
 */
enum Operator
{
    ADDITION
            {
                @Override
                public int apply(int x1, int x2)
                {
                    return x1 + x2;
                }
            },
    SUBTRACTION
            {
                @Override
                public int apply(int x1, int x2)
                {
                    return x1 - x2;
                }
            },
```

File - Matrice.java

```
229     PRODUCT
230             {
231                 @Override
232                 public int apply(int x1, int x2)
233                 {
234                     return x1 * x2;
235                 }
236             };
237 
238 
239     /**
240      * Applique une opération sur x1 et x2.
241      *
242      * @param x1 Premier nombre, élément de la première matrice.
243      * @param x2 Second nombre, élément de la seconde matrice.
244      * @return Résultat de l'opération entre x1 et x2.
245      */
246     public abstract int apply(int x1, int x2);
247 }
248 
```