

```

1  /**
2   * Classe de test pour la classe Matrice.
3   * Il est nécessaire d'entrer les arguments M1 N1 M2 N2 P
4   * Les arguments correspondent respectivement à la hauteur largeur
5   * de la première matrice, hauteur largeur de la seconde matrice et
6   * le modulo qu'elles respectent.
7   * Deux matrices de contenu aléatoire seront générées et testées.
8   * D'autres tests sont effectués avec des matrices prédéterminées pour
9   * valider les cas limites.
10  *
11  * @author Forestier Quentin, Herzig Melvyn
12  * @version 14.10.2020
13  */
14  public class Test
15  {
16
17
18      public static void testStatic()
19      {
20          int modulus = 5;
21
22          Matrice one = new Matrice(new int[][]{
23              {1, 3, 1, 1},
24              {3, 2, 4, 2},
25              {1, 0, 1, 0}
26          }, modulus);
27
28          Matrice two = new Matrice(new int[][]{
29              {1, 4, 2, 3, 2},
30              {0, 1, 0, 4, 2},
31              {0, 0, 2, 0, 2}
32          }, modulus);
33
34          System.out.println("The modulus is " + modulus + "\n");
35
36          System.out.println("one");
37          System.out.println(one);
38
39          System.out.println("\ntwo");
40          System.out.println(two);
41
42          System.out.println("\none + two");
43          System.out.println(one.add(two));
44
45          System.out.println("\none - two");
46          System.out.println(one.sub(two));
47
48          System.out.println("\none x two");
49          System.out.println(one.prod(two));
50      }
51
52      public static void testUserInput(int M1, int N1, int M2, int N2,
53                                      int modulus)
54      {
55
56          Matrice three = new Matrice(M1, N1, modulus);
57

```

```
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     catch (RuntimeException e)
86     {
87         System.out.println(e);
88     }
89
90     System.out.println("\n");
91
92     try
93     {
94         System.out.println("Test de matrice avec un modulo incorrecte :");
95         Matrice badModulus = new Matrice( new int[][]{{1, 2, 3}, {2, 3, 4}},
96                                           0);
97     }
98     catch (RuntimeException e)
99     {
100         System.out.println(e);
101     }
102
103     System.out.println("\n");
104
105     try
106     {
107         System.out.println("Test d'une opération avec un modulo différent" +
108                             " :");
109         Matrice m1 = new Matrice(2, 5);
110         Matrice m2 = new Matrice(5, 4);
111
112         m1.add(m2);
113     }
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     System.out.println("----- Test de la donnée -----");
174     testStatic();
175
176     System.out.println("\n----- Test de l'entrée utilisateur " +
177         "-----");
178     testUserInput(M1, N1, M2, N2, modulus);
179
180     System.out.println("\n----- Test de cas limite -----");
181     testLimitCase();
182 }
183 }
184
```

```

1  /**
2   * Class implémentant des matrices de taille personnalisable.
3   * Le contenu est défini modulo N.
4   * Les matrices peuvent être additionnées, soustraites et multipliées.
5   *
6   * @author Forestier Quentin, Herzig Melvyn
7   * @version 14.10.2020
8   */
9  public class Matrice
10 {
11     private int modulus;
12     private int matrice[][];
13
14     /**
15      * Constructeur, génère la matrice d'après les dimensions en paramètre
16      * ainsi que le contenu d'après le modulo.
17      *
18      * @param aHeight Hauteur de la matrice.
19      * @param aWidth Largeur de la matrice.
20      * @param aModulus Modulo du contenu.
21      * @throws RuntimeException Dans le cas d'une dimension négative ou
22      *                          d'un modulo inférieur à 1.
23      */
24     public Matrice(int aHeight, int aWidth, int aModulus)
25     {
26         if (aHeight < 0 || aWidth < 0 || aModulus < 1)
27             throw new RuntimeException("Argument(s) invalide(s)");
28
29         modulus = aModulus;
30
31         matrice = new int[aHeight][aWidth];
32
33         for (int M = 0; M < aHeight; ++M)
34         {
35             for (int N = 0; N < aWidth; ++N)
36             {
37                 matrice[M][N] = (int) (Math.random() * modulus);
38             }
39         }
40     }
41
42     /**
43      * Constructeur de matrice carrée d'après la taille en paramètre et
44      * génère le contenu d'après le modulo.
45      *
46      * @param size Hauteur et largeur de la matrice.
47      * @param aModulus Modulo du contenu.
48      */
49     public Matrice(int size, int aModulus)
50     {
51         this(size, size, aModulus);
52     }
53
54     /**
55      * Constructeur crée l'objet matrice d'après le tableau bidimensionnel
56      * en paramètre. Corrige son contenu d'après le module.
57      *

```

```

58     * @param aMatrice Matrice à intégrer.
59     * @param aModulus Module à appliquer.
60     * @throws RuntimeException Dans le cas d'un modulo inférieur à 1.
61     */
62     public Matrice(int aMatrice[][], int aModulus)
63     {
64         if (aModulus < 1)
65             throw new RuntimeException("Le modulo est < 1");
66
67         modulus = aModulus;
68         matrice = aMatrice;
69
70         // Corrige le contenu pour respecter le modulo.
71         for (int M = 0; M < this.getHeight(); ++M)
72         {
73             for (int N = 0; N < this.getWidth(); ++N)
74             {
75                 matrice[M][N] = Math.floorMod(matrice[M][N], modulus);
76             }
77         }
78     }
79
80     /**
81     * Retourne la hauteur de la matrice.
82     *
83     * @return Hauteur de la matrice.
84     */
85     public int getHeight()
86     {
87         return matrice.length;
88     }
89
90     /**
91     * Retourne la largeur de la matrice
92     *
93     * @return Largeur de la matrice.
94     */
95     public int getWidth()
96     {
97
98         return matrice.length == 0 ? 0 : matrice[0].length;
99     }
100
101
102     /**
103     * Retourne le modulo de la matrice
104     *
105     * @return Modulo de la matrice
106     */
107     public int getModulus()
108     {
109         return this.modulus;
110     }
111
112     /**
113     * Affiche le contenu de la matrice.
114     *

```

```

115     * @return String de la matrice.
116     */
117     @Override
118     public String toString()
119     {
120         if (getHeight() == 0 || getWidth() == 0) return "Matrice vide";
121         StringBuilder result = new StringBuilder();
122         for (int M = 0; M < this.getHeight(); ++M)
123         {
124             for (int N = 0; N < this.getWidth(); ++N)
125             {
126                 result.append(matrice[M][N]);
127             }
128             result.append('\n');
129         }
130         return result.toString();
131     }
132 }
133
134 /**
135  * Additionne la matrice courante avec la matrice m.
136  *
137  * @param m Matrice, seconde opérande.
138  * @return Matrice résultante de la somme.
139  */
140 public Matrice add(Matrice m)
141 {
142     return operation(m, Operator.ADDITION);
143 }
144
145 /**
146  * Soustrait la matrice courante avec la matrice m.
147  *
148  * @param m Matrice, seconde opérande.
149  * @return Matrice résultante de la différence.
150  */
151 public Matrice sub(Matrice m)
152 {
153     return operation(m, Operator.SUBTRACTION);
154 }
155
156 /**
157  * Multiplie la matrice courante avec la matrice m.
158  *
159  * @param m Matrice, seconde opérande.
160  * @return Matrice résultante du produit.
161  */
162 public Matrice prod(Matrice m)
163 {
164     return operation(m, Operator.PRODUCT);
165 }
166
167 /**
168  * Fonction principale de génération d'une matrice issue de deux matrices
169  * combinées, la matrice courante et la matrice m.
170  *
171  * @param m Matrice (tableau d'entier 2D), seconde opérande.

```

```

172     * @param operator Opération à appliquer entre les éléments Amn Bmn.
173     * @return Retourne la matrice résultante de l'opération.
174     */
175     private Matrice operation(Matrice m, Operator operator)
176     {
177         if (this.modulus != m.modulus)
178             throw new RuntimeException("Les modulus ne sont pas égaux");
179
180         Matrice result = new Matrice(Math.max(this.getHeight(), m.getHeight()),
181                                     Math.max(this.getWidth(), m.getWidth()),
182                                     this.modulus);
183
184         int v1, v2;
185         for (int M = 0; M < result.getHeight(); ++M)
186         {
187             for (int N = 0; N < result.getWidth(); ++N)
188             {
189                 // Vérification si la case MxN est dans les tableaux.
190                 v1 = (M < this.getHeight() && N < this.getWidth()) ?
191                     this.matrice[M][N] : 0;
192                 v2 = (M < m.getHeight() && N < m.getWidth()) ?
193                     m.matrice[M][N] : 0;
194
195                 result.matrice[M][N] = Math.floorMod(operator.apply(v1, v2),
196                                                         modulus);
197             }
198         }
199         return result;
200     }
201 }
202
203 /**
204  * Enum des opérations applicables sur les Matrices
205  *
206  * @author Forestier Quentin, Herzig Melvyn
207  * @version 14.10.2020
208  *
209  * Inspiré de: https://stackoverflow.com/a/2902471
210  */
211 enum Operator
212 {
213     ADDITION
214     {
215         @Override
216         public int apply(int x1, int x2)
217         {
218             return x1 + x2;
219         }
220     },
221     SUBTRACTION
222     {
223         @Override
224         public int apply(int x1, int x2)
225         {
226             return x1 - x2;
227         }
228     },

```



```
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
```