

```
1 /**
2  * This tests roughly the functions of the class Matrix. {@link Matrix}
3  *
4  * @author      Arthur Junod
5  * @author      Edwin Haeffner
6  * Date :       13/11/2023
7  */
8
9 public class Main {
10     public static void main(String[] args) {
11         int modulo = 5;
12         System.out.println("The modulus is " + modulo);
13
14         Matrix one = new Matrix(new int[][]{{1,3,1,1},
15                                             {3,2,4,2},
16                                             {1,0,1,0}}, modulo);
17
18         Matrix two = new Matrix(new int[][]{{1,4,2,3,2},
19                                             {0,1,0,4,2},
20                                             {0,0,2,0,2}}, modulo);
21
22         System.out.println("one");
23         System.out.println(one);
24
25         System.out.println("two");
26         System.out.println(two);
27
28         System.out.println("one + two");
29         System.out.println(one.add(two));
30
31         System.out.println("one - two");
32         System.out.println(one.sub(two));
33
34         System.out.println("one x two");
35         System.out.println(one.mult(two));
36
37         System.out.println("Test randomness: ");
38         System.out.println(new Matrix(4, 4, 7));
39         System.out.println(new Matrix(4, 4, 7));
40         System.out.println(new Matrix(4, 4, 7));
41     }
42 }
```

```

1  /**
2   * Class Matrix used to store a matrix, display this matrix and make
   * operations between two matrices.
3   * The matrix is made of integers
4   * This uses another class named "Operation" to operate each element of
   * the matrix : {@link Operation}
5   *
6   * @author      Arthur Junod
7   * @author      Edwin Haeffner
8   * Date :      13/11/2023
9   */
10
11 public class Matrix {
12     private final int m;
13     private int n;
14     private final int modValue;
15     private final int[][] matrix;
16
17     /**
18      * Create a randomly filled Matrix by giving its dimensions.
19      * @param m number of row
20      * @param n number of columns
21      * @param modValue value of the modulo
22      */
23     public Matrix(int m, int n, int modValue) {
24         verifyDim(m, n);
25         modPos(modValue);
26         this.n = n;
27         this.m = m;
28         this.modValue = modValue;
29
30         this.matrix = new int[m][n];
31
32         for (int i = 0; i < m; ++i) {
33             for (int j = 0; j < n; ++j) {
34                 matrix[i][j] = (int) Math.floor(Math.random()
35                     * modValue);
36             }
37         }
38     }
39
40     /**
41      * Create a Matrix from a given 2d array of int.
42      * @param matrix 2d array for the matrix
43      * @param modValue value of the modulo
44      */
45     public Matrix(int[][] matrix, int modValue){
46
47         arrayCheck(matrix);
48         modPos(modValue);
49

```

```

50
51     this.m = matrix.length;
52     this.n = matrix[0].length;
53
54     //Searching for the longest array in the given matrix
55     for(int i = 0; i < m;++i){
56         if (this.n < matrix[i].length)
57             this.n = matrix[i].length;
58     }
59
60     this.matrix = new int[m][n];
61     this.modValue = modValue;
62
63     for(int i = 0; i < m; ++i){
64         for(int j = 0; j < n; ++j){
65             if(j >= matrix[i].length) continue;
66             this.matrix[i][j] = Math.floorMod(matrix[i][j],
67                 modValue);
68         }
69     }
70 }
71
72 public String toString(){
73     StringBuilder out = new StringBuilder();
74     for(int i = 0; i < m; ++i){
75         for(int j = 0; j < n; ++j){
76             out.append(matrix[i][j]).append(" ");
77         }
78         out.append('\n');
79     }
80     return out.toString();
81 }
82
83 /**
84  * Applies an operation to all the elements of 2 matrix
85  * @param b Other Matrix for the operation
86  * @param operation Operation to apply
87  * @return A new Matrix from the application of
88  * the chosen operation.
89  */
90 private Matrix matrixOp(Matrix b, Operation operation) {
91
92     verifyMod(b);
93     int newM = Math.max(this.m,b.m);
94     int newN = Math.max(this.n,b.n);
95
96     int operandA;
97     int operandB;
98
99     int[][] newMatrix = new int[newM][newN];
100

```

```

101         for(int i = 0; i < newM; ++i){
102             for(int j = 0; j < newN; ++j){
103                 //If a matrix is bigger in the n or m dimension
104                 // than the other one, the smaller matrix sends
105                 // 0 to avoid out of range error.
106                 operandA = (i >= this.m || j >= this.n) ?
107                     0 : this.matrix[i][j];
108
109                 operandB = (i >= b.m || j >= b.n) ?
110                     0 : b.matrix[i][j];
111
112                 newMatrix[i][j] =
113                     Math.floorMod(operation.operate(operandA,
114                     operandB),modValue);
115             }
116         }
117
118         return new Matrix(newMatrix, modValue);
119     }
120
121     /**
122     * Add 2 Matrix together.
123     * @param b The other Matrix
124     * @return A new Matrix created from the addition
125     */
126     public Matrix add(Matrix b){
127         return matrixOp(b, new Addition());
128     }
129
130     /**
131     * Subtract 1 Matrix from another.
132     * @param b The other Matrix
133     * @return A new Matrix created from the subtraction
134     */
135     public Matrix sub(Matrix b){
136         return matrixOp(b, new Subtraction());
137     }
138
139     /**
140     * Multiply 2 matrix together.
141     * @param b The other Matrix
142     * @return A new Matrix created from the multiplication
143     */
144     public Matrix mult(Matrix b){
145         return matrixOp(b, new Multiplication());
146     }
147
148     /**
149     * Verify that the moduli of this Matrix and another
150     * are equals, else it throws and exception.
151     * @param b The other Matrix

```

```

152      * @throws RuntimeException if the moduli of the 2 matrices
153      * are not equal.
154      */
155      private void verifyMod(Matrix b) {
156          if(modValue != b.modValue)
157              throw new RuntimeException("The moduli of the 2 " +
158                                      "matrices are not equal");
159      }
160
161      /**
162       * Verify that a 2d array of int isn't null or empty,
163       * else it throws an exception.
164       * @param a The 2d array to check
165       * @throws RuntimeException if the array passed as parameter
166       * isn't valid.
167       */
168      private void arrayCheck(int[][] a){
169          if(a == null || a.length == 0 || a[0].length == 0)
170              throw new RuntimeException("The array passed as parameter" +
171                                      " isn't valid");
172      }
173
174      /**
175       * Verify that a modulo isn't negative or equal to 0,
176       * else it throws an exception.
177       * @param mod The modulo to check
178       * @throws RuntimeException if the modulo cannot be negative
179       * or equal to 0.
180       */
181      private void modPos(int mod){
182          if(mod <= 0)
183              throw new RuntimeException("The modulo cannot be negative" +
184                                      " or equal to 0");
185      }
186
187      /**
188       * Verify that two int aren't negative or equal to 0,
189       * else it throws an exception.
190       * @param m One of the int to check
191       * @param n The second int to check
192       * @throws RuntimeException if one of the dimension
193       * is lower or equal to 0.
194       */
195      private void verifyDim(int m, int n){
196          if (n <= 0 || m <= 0)
197              throw new RuntimeException("One of the dimension is" +
198                                      " lower or equal to 0");
199      }
200 }
201

```

```
1 /**
2  * This adds operand A and operand B.
3  * Children of the abstract class Operation : {@link Operation}
4  * Operand A and operand B must be integers
5  *
6  * @author      Arthur Junod
7  * @author      Edwin Haeffner
8  * Date :        13/11/2023
9  */
10
11 public class Addition extends Operation{
12
13     /**
14      * Add the second operand to the first
15      * @param opA first operand
16      * @param opB second operand
17      * @return the first plus the second operand
18      */
19     @Override
20     public int operate(int opA, int opB) {
21         return opA + opB;
22     }
23 }
24
```

```
1 /**
2  * Abstract class Operation used to operate two operands.
3  * Operand A and operand B must be integers
4  *
5  * @author      Arthur Junod
6  * @author      Edwin Haeffner
7  * Date :        13/11/2023
8  */
9 public abstract class Operation {
10
11     /**
12      * Applies the operation on the two operands.
13      * @param opA first operand
14      * @param opB second operand
15      * @return the result of the operation
16      */
17     abstract int operate(int opA,int opB);
18 }
19
```

```

1  /**
2   * This tests the functions of the class Matrix using junit. {@link
   Matrix}
3   *
4   * @author      Arthur Junod
5   * @author      Edwin Haeffner
6   * Date :       13/11/2023
7   */
8
9
10 import junit.extensions.RepeatedTest;
11 import org.junit.Test;
12
13 import java.lang.annotation.Repeatable;
14
15 import static org.junit.Assert.assertEquals;
16 import static org.junit.Assert.assertThrows;
17 public class MatrixTest {
18
19     private static final int[][] MATRIX_NOT_FULL = {{1},
20                                                     {3,2,4},
21                                                     {},
22                                                     {1,0,2,3,1,2,1}};
23
24     private static final int[][] MATRIX_NOT_FULL_RES = {{1,0,0,0,0,0,0},
25                                                         {3,2,4,0,0,0,0},
26                                                         {0,0,0,0,0,0,0},
27                                                         {1,0,2,3,1,2,1}};
28
29     private static final int[][] MATRIX_ONE = {{1,3,1,1},
30                                                 {3,2,4,2},
31                                                 {1,0,1,0}};
32
33     private static final int[][] MATRIX_TWO = {{1,4,2,3,2},
34                                                 {0,1,0,4,2},
35                                                 {0,0,2,0,2}};
36
37     private static final int[][] MATRIX_RESULT_ADD = {{2,2,3,4,2},
38                                                         {3,3,4,1,2},
39                                                         {1,0,3,0,2}};
40
41     private static final int[][] MATRIX_RESULT_SUB = {{0,4,4,3,3},
42                                                         {3,1,4,3,3},
43                                                         {1,0,4,0,3}};
44     private static final int[][] MATRIX_RESULT_MULT = {{1,2,2,3,0},
45                                                         {0,2,0,3,0},
46                                                         {0,0,2,0,0}};
47     private static final int[][] MATRIX_MOD = {{-1,2,-2,3,0},
48                                                  {0,2,0,3,-465},
49                                                  {-9,0,2,0,0},
50                                                  {-7,-8,345,0,1}};

```



```

51     private static final int[][] MATRIX_RESULT_MOD = {{7,2,6,3,0},
52                                                         {0,2,0,3,7},
53                                                         {7,0,2,0,0},
54                                                         {1,0,1,0,1}};
55     @Test
56     public void testAdd(){
57
58         Matrix matrixA = new Matrix(MATRIX_ONE,5);
59         Matrix matrixB = new Matrix(MATRIX_TWO,5);
60
61         Matrix matrixResAdd = new Matrix(MATRIX_RESULT_ADD,5);
62
63         assertEquals("Test: add Matrix FAILED",
64                     matrixResAdd.toString(), matrixA.add(matrixB).toString());
65     }
66
67     @Test
68     public void testSub(){
69
70         Matrix matrixA = new Matrix(MATRIX_ONE,5);
71         Matrix matrixB = new Matrix(MATRIX_TWO,5);
72
73         Matrix matrixResSub = new Matrix(MATRIX_RESULT_SUB,5);
74
75         assertEquals("Test: sub Matrix FAILED",
76                     matrixResSub.toString(),matrixA.sub(matrixB).toString());
77     }
78
79     @Test
80     public void testMult(){
81
82         Matrix matrixA = new Matrix(MATRIX_ONE,5);
83         Matrix matrixB = new Matrix(MATRIX_TWO,5);
84
85         Matrix matrixResMult = new Matrix(MATRIX_RESULT_MULT,5);
86
87         assertEquals("Test: mult Matrix FAILED",
88                     matrixResMult.toString(),matrixA.mult(matrixB).toString());
89     }
90     @Test
91     public void testMod(){
92         Matrix matrix = new Matrix(MATRIX_MOD,8);
93         Matrix matrixResMod = new Matrix(MATRIX_RESULT_MOD,8);
94
95         assertEquals("Test: mod negative Matrix FAILED",
96                     matrixResMod.toString(),matrix.toString());
97     }
98
99     @Test
100    public void testMatrixNotFull(){
101        Matrix matrix = new Matrix(MATRIX_NOT_FULL,5);

```

```

102         Matrix matrixRes = new Matrix(MATRIX_NOT_FULL_RES,5);
103
104         assertEquals("Test: matrix not full FAILED",
105             matrixRes.toString(),matrix.toString());
106     }
107
108
109     @Test
110     public void testInitDimMod(){
111         Exception e = assertThrows(RuntimeException.class,
112             ()-> new Matrix(1, 3, 0));
113         assertEquals("Test: mod equal to zero when init with dim FAILED"
114             , "The modulo cannot be negative or equal to 0",
115             e.getMessage());
116     }
117
118     @Test
119     public void testInitMatrixMod(){
120         Exception e = assertThrows(RuntimeException.class,
121             ()->new Matrix(new int[][] {{1,3}, {1,4,5,6}}, 0));
122         assertEquals("Test: mod equal to zero when init" +
123             " with matrix FAILED",
124             "The modulo cannot be negative or equal to 0",
125             e.getMessage());
126     }
127
128     @Test
129     public void testVerifyMod(){
130         Matrix matrixA = new Matrix(MATRIX_ONE,5);
131         Matrix matrixB = new Matrix(MATRIX_TWO,8);
132
133         Exception e = assertThrows(RuntimeException.class,
134             ()->matrixA.add(matrixB));
135         assertEquals("Test: same mod when operating FAILED",
136             "The moduli of the 2 matrices are not equal",
137             e.getMessage());
138     }
139
140     @Test
141     public void testInitDimZero(){
142         Exception em = assertThrows(RuntimeException.class,
143             ()->new Matrix(0, 3, 4));
144         assertEquals("Test: dim m not equal to zero FAILED",
145             "One of the dimension is lower or equal to 0",
146             em.getMessage());
147         Exception en = assertThrows(RuntimeException.class,
148             ()->new Matrix(4, 0, 4));
149         assertEquals("Test: dim n not equal to zero FAILED",
150             "One of the dimension is lower or equal to 0",
151             en.getMessage());
152     }

```

```
153
154     @Test
155     public void testInitEmptyMatrix(){
156         Exception e = assertThrows(RuntimeException.class,
157             ()->new Matrix(new int[][]{{}}, 9));
158         assertEquals("Test: init with an empty matrix FAILED",
159             "The array passed as parameter isn't valid",
160             e.getMessage());
161
162     }
163 }
164
```

```
1 /**
2  * This subtracts operand B from operand A.
3  * Children of the abstract class Operation : {@link Operation}
4  * Operand A and operand B must be integers
5  *
6  * @author      Arthur Junod
7  * @author      Edwin Haeffner
8  * Date :       13/11/2023
9  */
10 public class Subtraction extends Operation {
11
12     /**
13      * Subtract the second operand to the first.
14      * @param opA first operand
15      * @param opB second operand
16      * @return the first operand minus the second
17      */
18     @Override
19     public int operate(int opA, int opB) {
20         return opA - opB;
21     }
22 }
23
```

```
1 /**
2  * This multiplies operand A and operand B.
3  * Children of the abstract class Operation : {@link Operation}
4  * Operand A and operand B must be integers
5  *
6  * @author      Arthur Junod
7  * @author      Edwin Haeffner
8  * Date :       13/11/2023
9  */
10 public class Multiplication extends Operation{
11
12     /**
13      * Multiply the second and first operands.
14      * @param opA first operand
15      * @param opB second operand
16      * @return first * second
17      */
18     @Override
19     public int operate(int opA, int opB) {
20         return opA * opB;
21     }
22 }
23
```