420-201-VA | Assignment 2-A | Due by 11 March 2020

Modeling a Two-Dimensional Table

1 Purpose

The purpose of this assignment is to provide programming exercises for you to practice using two-dimensional arrays.

2 Background

Java rectangular array is an array of arrays, arranged in rows, where each row, in turn, is an array, and all rows have the same length. You can think of a rectangular array as a *grid* of variables.

Java ragged (or jagged) array is also an array of arrays, arranged in rows, but the rows can be of different lengths.

Most people consider the following 2D arrays a grid of integers and a ragged array of integers:

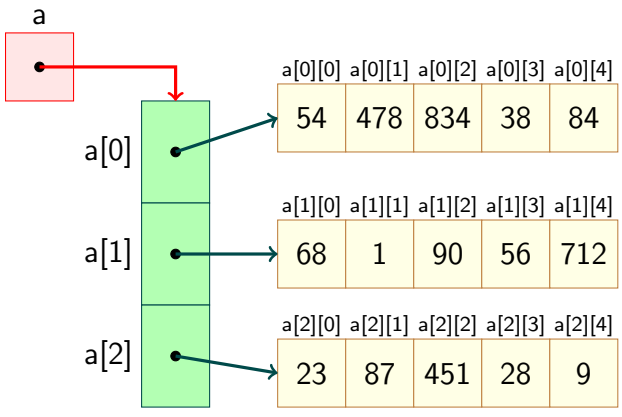
54	478	834	38	84
68	1	90	56	712
23	87	451	28	9

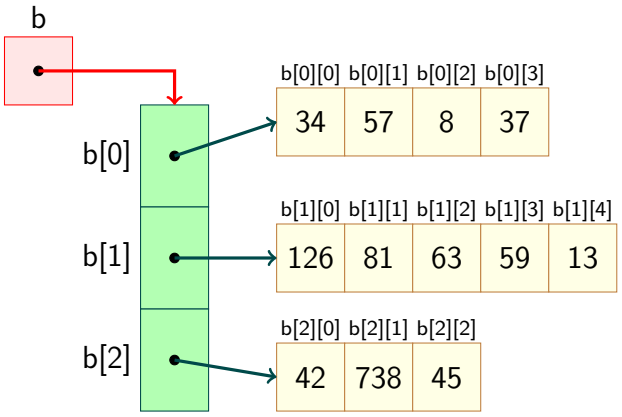
34	57	8	37	
126	81	63	59	13
42	738	45		

Programmers, however, note that in Java such 2D arrays are represented as illustrated below:

```
// A grid of integers
int[][] a = {{54, 478, 834, 38, 84},
             {68, 1, 90, 56, 712},
             {23, 87, 451, 28, 9}};
```

```
// A ragged array of integers
int[][] b = {{34, 57, 8, 37},
             {126, 81, 63, 59, 13},
             {42, 738, 45}};
```





3 Your Task

Having experienced that Java arrays are "dumb" objects in the sense that they offer no "array operations", you set out to create objects that model a "smart" grid of integers. You name your class `IntGrid` and specify it as follows:

IntGrid	The name of this class
− grid : int[ ][ ]	stores a reference to a 2D array of integers and is referred to as the grid in *this* `IntGrid` object
+ IntGrid(int[][] initialArray) : + IntGrid(int size) : + IntGrid(rows : int , cols : int): + IntGrid(it : IntGrid):	These four constructors delegate initialization of *this* grid to another constructor or a **setGrid** method below
+ setGrid(int[][] initialArray) : void	setting the number of columns in *this* grid to the length of the longest row of initArray, deep copies initialArray to *this* grid, filling new elements with zeros
+ setGrid(int size) : void	sets *this* grid to a zero-filled square array of integers of the specified size; throws an illegal argument exception if size< 0
+ setGrid(int rows, int cols) : void	sets *this* grid to a zero-filled array of the given rows and cols[a]
+ setElement(int r, int c, int value) : void	sets *this* grid's cell at row r and column c to value[a]
+ getElement(int row, int col) : int	returns value at the specified row and column[a]
+ getRow(k : int) : int[]	Returns a deep copy of the k'th row of *this* grid[a]
+ setRow(k : int, array : int []) : void	deep copies array to the k'th row of *this* grid[a]
+ numRows() : int	returns the number of rows in *this* grid
+ numColumns() : int	returns the number of columns in *this* grid
+ isSquare(): boolean	determines whether numRows() = numColumns()
+ isTall() : boolean	determines whether numRows() > numColumns()
+ isWide() : boolean	determines whether numRows() < numColumns()
+ max() : int	returns the largest value in *this* grid
+ min() : int	returns the smallest value in *this* grid
+ rowSum(int row) : int	returns the sum of elements in given row[a]
+ columnSum(int column) : int	returns the sum of elements in given column[a]
+ majorDiagonalSum() : int	returns the sum of elements on the major diagonal(s)
+ minorDiagonalSums() : int	returns the sum of elements on the minor diagonal(s)
+ allRowSums() : int[]	returns an array of all row sums
+ allColumnSums() : int[]	returns an array of all column sums
+ swapRows(r1 : int , r2 : int) : void	swaps rows r1 and r2[a]
+ swapColumns(c1 : int , c2 : int) :void	swaps columns c1 and c2[a]
+ lookup(int key) : boolean	determines whether *this* grid contains the given key
+ isMagicSquare() : boolean	determines whether *this* grid forms a normal magic square
+ printGridFeatures(): void	prints rows, columns, sums, min, max, etc. See lines 8-20, page 8.
+ equals(obj : Object) : boolean	compares *this* and obj for equality
+ toString() : String	returns a string representing *this* grid formatted neatly

[a] must throw an illegal argument exception if one or both of row and col indices are out of bounds

3.1 Where are the major and minor diagonal elements in a grid?

In this assignment, we define major diagonals and minor diagonals as follows:

3.1.1 Non-Square (Wide and Tall) Grids

There are two major diagonals running parallel to each other: one includes the top left cell and the other includes the bottom right cell.

There are two minor diagonals running parallel to each other: one includes the top right cell and the other includes the bottom left cell.

3.1.2 Square Grids

The major diagonal runs from the top left cell to the bottom right cell.

The minor diagonal runs from the top right cell to the bottom left cell.

3.1.3 Examples

A wide grid

2	11	16	19	22	15	8	5
21	23	13	7	20	18	12	1
3	4	24	10	17	14	9	6

A tall grid

1	24	6
19	21	12
16	2	14
13	18	10
5	7	17
15	9	23
8	3	22
4	11	20

A square grid

8	1	6
3	5	7
4	9	2

4 Test Driver Code

```
import java.util.Arrays;
/*
Important: you must enable assertion for your project as follows:
   1) Right click on the project in the Project Explorer
   2) Choose Properties (at the bottom of pop up menu)
   3) Choose Run (under Categories)
   4) In the VM Options field, enter -ea
   5) Click OK
*/
public class IntGridTestDriver
{
   public static void main(String[] args)
   {
      System.out.println("Testing constructor that takes a ragged array");
      int[][] a_ragged_array =
      {  {17, 18, 25,  2, 15}, {16,  5,  7, 14},
         {22,  6, 13, 20},
         { 3, 12, 19, 21, 10},
         {11, 24}
      };
      // instantiate a new IntGrid object using a_ragged_array above
      IntGrid it1 = new IntGrid(a_ragged_array);
      System.out.println(it1);
      it1.printGridFeatures();

      assert(it1.numRows() == 5); // test numRows()
      assert(it1.numColumns() == 5); // test numColumns()

      // test getRow
      assert(Arrays.equals(it1.getRow(0) , new int[]{17, 18, 25,  2, 15}));
      assert(Arrays.equals(it1.getRow(1) , new int[]{16,  5,  7, 14,  0}));
      assert(Arrays.equals(it1.getRow(2) , new int[]{22,  6, 13, 20,  0}));
      assert(Arrays.equals(it1.getRow(3) , new int[]{ 3, 12, 19, 21, 10}));
      assert(Arrays.equals(it1.getRow(4) , new int[]{11, 24,  0,  0,  0}));
      System.out.println("Constructor taking a ragged array: " + "OK");

      System.out.println("\nTesting getElement+ setElement");
      assert(it1.getElement(4, 1) - it1.getElement(4, 0) == 13);
      assert(it1.getElement(1, 4) == it1.getElement(4, 4));
      it1.setElement(1, 4, 23);
      it1.setElement(4, 2, 1);
      it1.setElement(4, 3, 8);
      it1.setElement(4, 4, 9);
      it1.setElement(2, 4, 4);

      assert(Arrays.equals(it1.getRow(0) , new int[]{17, 18, 25,  2, 15}));
      assert(Arrays.equals(it1.getRow(1) , new int[]{16,  5,  7, 14, 23}));
      assert(Arrays.equals(it1.getRow(2) , new int[]{22,  6, 13, 20,  4}));
      assert(Arrays.equals(it1.getRow(3) , new int[]{ 3, 12, 19, 21, 10}));
      assert(Arrays.equals(it1.getRow(4) , new int[]{11, 24,  1,  8,  9}));
      System.out.println("\nTested  getElement+ setElement: OK");
```

```
60
61      System.out.println("\nTesting copy constructor");
62      IntGrid it2 = new IntGrid(it1);
63      assert(it1.equals(it2));
64      System.out.println("Tested  copy constructor: " + "OK");
65
66      System.out.println("\nTesting set and get element methods");
67      // swap the elements at the two ends of the main diagonal
68      int temp = it1.getElement(0,0);
69      it1.setElement(0,0, it1.getElement(4,4));
70      it1.setElement(4, 4, temp);
71
72      // swap the elements at the two ends of the sub-diagonal
73      int temp2 = it1.getElement(0,4);
74      it1.setElement(0,4, it1.getElement(4,0));
75      it1.setElement(4, 0, temp2);
76
77      assert(Arrays.equals(it1.getRow(0) , new int[]{ 9, 18, 25,  2, 11}));
78      assert(Arrays.equals(it1.getRow(1) , new int[]{16,  5,  7, 14, 23}));
79      assert(Arrays.equals(it1.getRow(2) , new int[]{22,  6, 13, 20,  4}));
80      assert(Arrays.equals(it1.getRow(3) , new int[]{ 3, 12, 19, 21, 10}));
81      assert(Arrays.equals(it1.getRow(4) , new int[]{15, 24,  1,  8, 17}));
82      System.out.println("\nTested  set and get element methods: OK");
83
84      System.out.println("\nTesting swapRows");
85      // swap top and bottom rows
86      it2.swapRows(0, 4);
87      assert(Arrays.equals(it1.getRow(4) , new int[]{15, 24,  1,  8, 17}));
88      assert(Arrays.equals(it1.getRow(1) , new int[]{16,  5,  7, 14, 23}));
89      assert(Arrays.equals(it1.getRow(2) , new int[]{22,  6, 13, 20,  4}));
90      assert(Arrays.equals(it1.getRow(3) , new int[]{ 3, 12, 19, 21, 10}));
91      assert(Arrays.equals(it1.getRow(0) , new int[]{ 9, 18, 25,  2, 11}));
92
93      System.out.println("\nTested  swapRows: OK");
94
95      System.out.println("\nTesting swapColumns");
96      // swap elements of first column and last column,
97      it1.swapColumns(0, it1.numColumns()-1);
98      assert(Arrays.equals(it1.getRow(4) , new int[]{17, 24,  1,  8, 15}));
99      assert(Arrays.equals(it1.getRow(1) , new int[]{23,  5,  7, 14, 16}));
100     assert(Arrays.equals(it1.getRow(2) , new int[]{ 4,  6, 13, 20, 22}));
101     assert(Arrays.equals(it1.getRow(3) , new int[]{10, 12, 19, 21,  3}));
102     assert(Arrays.equals(it1.getRow(0) , new int[]{11, 18, 25,  2,  9}));
103     System.out.println("\nTested  swapColumns: OK");
104     it1.printGridFeatures();
```

```
105
106         System.out.println("\nTesting setTable passing a 2D array");
107         int[][] array2d =  {{1, 2, 3, 4, 5},{2, 3, 4, 5, 1},{3, 4, 5, 1, 2},
108                 {4, 5, 1, 2, 3},{5, 1, 2, 3, 4}};
109         it1.setGrid(array2d);
110         System.out.println(it1);
111         it1.printGridFeatures();
112         assert(Arrays.equals(it1.getRow(0) , new int[]{1, 2, 3, 4, 5 }));
113         assert(Arrays.equals(it1.getRow(1) , new int[]{2, 3, 4, 5, 1 }));
114         assert(Arrays.equals(it1.getRow(2) , new int[]{3, 4, 5, 1, 2 }));
115         assert(Arrays.equals(it1.getRow(3) , new int[]{4, 5, 1, 2, 3 }));
116         assert(Arrays.equals(it1.getRow(4) , new int[]{5, 1, 2, 3, 4 }));
117         System.out.println("Tested  setTable passing a 2D array: " + "OK");
118
119         System.out.println("\nTesting row, column, and diagonal sums, and min, max");
120         assert(Arrays.equals(it1.allRowSums() , new int[]{15, 15, 15, 15, 15}));
121         assert(Arrays.equals(it1.allColumnSums() , new int[]{15, 15, 15, 15, 15}));
122         assert(it1.majorDiagonalSums() == 15);
123         assert(it1.minorDiagonalSums() == 25);
124         assert(it1.min() == 1);
125         assert(it1.max() == 5);
126         assert(it1.isMagicSquare() == false);
127         System.out.println("Tested  row, column, and diagonal sums, and min, max: " + "
    OK");
128
129         System.out.println("\nTesting constructor taking two int arguments - wide grid"
    );
130         IntGrid it3 = new IntGrid(3,8);
131         System.out.println(it3);
132         it3.printGridFeatures();
133
134         assert(Arrays.equals(it3.getRow(0) , new int[]{0, 0, 0, 0, 0, 0, 0, 0 }));
135         assert(Arrays.equals(it3.getRow(1) , new int[]{0, 0, 0, 0, 0, 0, 0, 0 }));
136         assert(Arrays.equals(it3.getRow(2) , new int[]{0, 0, 0, 0, 0, 0, 0, 0 }));
137         assert(Arrays.equals(it3.allRowSums() , new int[]{0, 0, 0}));
138         assert(Arrays.equals(it3.allColumnSums() , new int[]{0, 0, 0, 0, 0, 0, 0, 0}));
139         assert(it3.majorDiagonalSums() == 0);
140         assert(it3.minorDiagonalSums() == 0);
141         assert(it3.numRows() == 3);
142         assert(it3.numColumns() == 8);
143         System.out.println("Tested  constructor taking two int arguments - wide grid: "
     + "OK");
144
145         System.out.println("\nTesting setRow - wide case");
146         it3.setRow(0, new int[]{2,11,16,19,22,15,8,5});
147         it3.setRow(1, new int[]{21,23,13,7,20,18,12,1});
148         it3.setRow(2, new int[]{3,4,24,10,17,14,9,6});
149
150         System.out.println(it3);
151         it3.printGridFeatures();
```

```
152
153         assert(Arrays.equals(it3.getRow(0) , new int[]{2,11,16,19,22,15,8,5}));
154         assert(Arrays.equals(it3.getRow(1) , new int[]{21,23,13,7,20,18,12,1}));
155         assert(Arrays.equals(it3.getRow(2) , new int[]{3,4,24,10,17,14,9,6}));
156         assert(Arrays.equals(it3.allRowSums() , new int[]{98, 115, 87}));
157         assert(Arrays.equals(it3.allColumnSums() , new int[]{26, 38, 53, 36, 59, 47,
    29, 12}));
158         assert(it3.majorDiagonalSums() == 82);
159         assert(it3.minorDiagonalSums() == 73);
160         assert(it3.numRows() == 3);
161         assert(it3.numColumns() == 8);
162         System.out.println("Tested  setRow - wide case: " + "OK");
163
164         System.out.println("\nTesting constructor taking two int arguments - Tall grid"
    );
165         IntGrid it4 = new IntGrid(8,3);
166         System.out.println(it4);
167         it4.printGridFeatures();
168
169         assert(Arrays.equals(it4.getRow(0) , new int[]{0, 0, 0}));
170         assert(Arrays.equals(it4.getRow(1) , new int[]{0, 0, 0}));
171         assert(Arrays.equals(it4.getRow(2) , new int[]{0, 0, 0}));
172         assert(Arrays.equals(it4.getRow(3) , new int[]{0, 0, 0}));
173         assert(Arrays.equals(it4.getRow(4) , new int[]{0, 0, 0}));
174         assert(Arrays.equals(it4.getRow(5) , new int[]{0, 0, 0}));
175         assert(Arrays.equals(it4.getRow(6) , new int[]{0, 0, 0}));
176         assert(Arrays.equals(it4.getRow(7) , new int[]{0, 0, 0}));
177
178         assert(Arrays.equals(it4.allRowSums() , new int[]{0, 0, 0, 0, 0, 0, 0, 0}));
179         assert(Arrays.equals(it4.allColumnSums() , new int[]{0, 0, 0}));
180         assert(it4.majorDiagonalSums() == 0);
181         assert(it4.minorDiagonalSums() == 0);
182         assert(it4.numRows() == 8);
183         assert(it4.numColumns() == 3);
184         System.out.println("Tested  constructor taking two int arguments - Tall grid: "
     + "OK");
185
186         System.out.println("\nTesting setRow - Tall case");
187         it4.setRow(0, new int[]{ 1, 24, 6 });
188         it4.setRow(1, new int[]{19, 21,12 });
189         it4.setRow(2, new int[]{16,  2,14 });
190         it4.setRow(3, new int[]{13, 18,10 });
191         it4.setRow(4, new int[]{ 5,  7,17 });
192         it4.setRow(5, new int[]{15,  9,23 });
193         it4.setRow(6, new int[]{ 8,  3,22 });
194         it4.setRow(7, new int[]{ 4, 11,20 });
195
196         System.out.println(it4);
197         it4.printGridFeatures();
```

```
198
199         assert(Arrays.equals(it4.getRow(0) , new int[]{ 1, 24, 6}));
200         assert(Arrays.equals(it4.getRow(1) , new int[]{19, 21,12}));
201         assert(Arrays.equals(it4.getRow(2) , new int[]{16,  2,14}));
202         assert(Arrays.equals(it4.getRow(3) , new int[]{13, 18,10}));
203         assert(Arrays.equals(it4.getRow(4) , new int[]{ 5,  7,17}));
204         assert(Arrays.equals(it4.getRow(5) , new int[]{15,  9,23}));
205         assert(Arrays.equals(it4.getRow(6) , new int[]{ 8,  3,22}));
206         assert(Arrays.equals(it4.getRow(7) , new int[]{ 4, 11,20}));
207
208         assert(Arrays.equals(it4.allRowSums() , new int[]{31, 52, 32, 41, 29, 47, 33,
    35}));
209         assert(Arrays.equals(it4.allColumnSums() , new int[]{81, 95, 124}));
210         assert(it4.majorDiagonalSums() == 74);
211         assert(it4.minorDiagonalSums() == 73);
212         assert(it4.numRows() == 8);
213         assert(it4.numColumns() == 3);
214         System.out.println("Tested  setRow - Tall case: " + "OK");
215
216         System.out.println("\nTesting isMagicSquare");
217         it2.setRow(0, new int[]{1, 1, 1, 1, 1});
218         it2.setRow(1, new int[]{1, 1, 1, 1, 1});
219         it2.setRow(2, new int[]{1, 1, 1, 1, 1});
220         it2.setRow(3, new int[]{1, 1, 1, 1, 1});
221         it2.setRow(4, new int[]{1, 1, 1, 1, 1});
222         System.out.println(it2);
223         it2.printGridFeatures();
224         assert(it2.isMagicSquare() == false);
225
226         it2.setRow(0, new int[]{17, 24, 1, 8, 15});
227         it2.setRow(1, new int[]{23, 5, 7, 14, 16});
228         it2.setRow(2, new int[]{4, 6, 13, 20, 22});
229         it2.setRow(3, new int[]{10, 12, 19, 21, 3});
230         it2.setRow(4, new int[]{11, 18, 25, 2, 9});
231         System.out.println(it2);
232         it2.printGridFeatures();
233         assert(it2.isColumnMagic() == true);
234         System.out.println("Tested isMagicSquare: " + "OK");
235         System.out.println("\nCongratulations! your program seems to be working
    correctly!");
236     }
237 }
```

5 Output

```
Testing constructor that takes a ragged array
 17  18  25   2  15
 16   5   7  14   0
 22   6  13  20   0
  3  12  19  21  10
 11  24   0   0   0

Dimensions: 5 rows by 5 columns (a square grid)
Row 1:              [17, 18, 25, 2, 15]
Row 2:              [16, 5, 7, 14, 0]
Row 3:              [22, 6, 13, 20, 0]
Row 4:              [3, 12, 19, 21, 10]
Row 5:              [11, 24, 0, 0, 0]
Row sums:           [77, 42, 61, 65, 35]
col sums:           [69, 65, 64, 57, 25]
major diagonal sum: 56
minor diagonal sum: 65
max element:        25
min element:        0
is magic square?    false

Constructor taking a ragged array: OK

Testing getElement+ setElement

Tested  getElement+ setElement: OK

Testing copy constructor
Tested  copy constructor: OK

Testing set and get element methods

Tested  set and get element methods: OK

Testing swapRows

Tested  swapRows: OK

Testing swapColumns

Tested  swapColumns: OK
Dimensions: 5 rows by 5 columns (a square grid)
Row 1:              [11, 18, 25, 2, 9]
Row 2:              [23, 5, 7, 14, 16]
Row 3:              [4, 6, 13, 20, 22]
```

```
Row 4:              [10, 12, 19, 21, 3]
Row 5:              [17, 24, 1, 8, 15]
Row sums:           [65, 65, 65, 65, 65]
col sums:           [65, 65, 65, 65, 65]
major diagonal sum: 65
minor diagonal sum: 65
max element:        25
min element:        1
is magic square?    true


Testing setTable passing a 2D array
  1   2   3   4   5
  2   3   4   5   1
  3   4   5   1   2
  4   5   1   2   3
  5   1   2   3   4

Dimensions: 5 rows by 5 columns (a square grid)
Row 1:              [1, 2, 3, 4, 5]
Row 2:              [2, 3, 4, 5, 1]
Row 3:              [3, 4, 5, 1, 2]
Row 4:              [4, 5, 1, 2, 3]
Row 5:              [5, 1, 2, 3, 4]
Row sums:           [15, 15, 15, 15, 15]
col sums:           [15, 15, 15, 15, 15]
major diagonal sum: 15
minor diagonal sum: 25
max element:        5
min element:        1
is magic square?    false

Tested  setTable passing a 2D array: OK

Testing row, column, and diagonal sums, and min, max
Tested  row, column, and diagonal sums, and min, max: OK

Testing constructor taking two int arguments - wide grid
  0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0

Dimensions: 3 rows by 8 columns (a wide grid)
Row 1:              [0, 0, 0, 0, 0, 0, 0, 0]
Row 2:              [0, 0, 0, 0, 0, 0, 0, 0]
Row 3:              [0, 0, 0, 0, 0, 0, 0, 0]
Row sums:           [0, 0, 0]
```

```
col sums:           [0, 0, 0, 0, 0, 0, 0, 0]
major diagonal sum: 0
minor diagonal sum: 0
max element:        0
min element:        0
is magic square?    false

Tested  constructor taking two int arguments - wide grid: OK

Testing setRow - wide case
  2  11  16  19  22  15   8   5
 21  23  13   7  20  18  12   1
  3   4  24  10  17  14   9   6

Dimensions: 3 rows by 8 columns (a wide grid)
Row 1:              [2, 11, 16, 19, 22, 15, 8, 5]
Row 2:              [21, 23, 13, 7, 20, 18, 12, 1]
Row 3:              [3, 4, 24, 10, 17, 14, 9, 6]
Row sums:           [98, 115, 87]
col sums:           [26, 38, 53, 36, 59, 47, 29, 12]
major diagonal sum: 82
minor diagonal sum: 73
max element:        24
min element:        1
is magic square?    false

Tested  setRow - wide case: OK

Testing constructor taking two int arguments - Tall grid
  0   0   0
  0   0   0
  0   0   0
  0   0   0
  0   0   0
  0   0   0
  0   0   0
  0   0   0

Dimensions: 8 rows by 3 columns (a tall grid)
Row 1:              [0, 0, 0]
Row 2:              [0, 0, 0]
Row 3:              [0, 0, 0]
Row 4:              [0, 0, 0]
Row 5:              [0, 0, 0]
Row 6:              [0, 0, 0]
Row 7:              [0, 0, 0]
Row 8:              [0, 0, 0]
```

```
Row sums:           [0, 0, 0, 0, 0, 0, 0, 0]
col sums:           [0, 0, 0]
major diagonal sum: 0
minor diagonal sum: 0
max element:        0
min element:        0
is magic square?    false

Tested  constructor taking two int arguments - Tall grid: OK

Testing setRow - Tall case
  1  24   6
 19  21  12
 16   2  14
 13  18  10
  5   7  17
 15   9  23
  8   3  22
  4  11  20

Dimensions: 8 rows by 3 columns (a tall grid)
Row 1:              [1, 24, 6]
Row 2:              [19, 21, 12]
Row 3:              [16, 2, 14]
Row 4:              [13, 18, 10]
Row 5:              [5, 7, 17]
Row 6:              [15, 9, 23]
Row 7:              [8, 3, 22]
Row 8:              [4, 11, 20]
Row sums:           [31, 52, 32, 41, 29, 47, 33, 35]
col sums:           [81, 95, 124]
major diagonal sum: 74
minor diagonal sum: 73
max element:        24
min element:        1
is magic square?    false

Tested  setRow - Tall case: OK

Testing isMagicSquare
  1   1   1   1   1
  1   1   1   1   1
  1   1   1   1   1
  1   1   1   1   1
  1   1   1   1   1

Dimensions: 5 rows by 5 columns (a square grid)
```

```
Row 1:              [1, 1, 1, 1, 1]
Row 2:              [1, 1, 1, 1, 1]
Row 3:              [1, 1, 1, 1, 1]
Row 4:              [1, 1, 1, 1, 1]
Row 5:              [1, 1, 1, 1, 1]
Row sums:           [5, 5, 5, 5, 5]
col sums:           [5, 5, 5, 5, 5]
major diagonal sum: 5
minor diagonal sum: 5
max element:        1
min element:        1
is magic square?    false

 17  24   1   8  15
 23   5   7  14  16
  4   6  13  20  22
 10  12  19  21   3
 11  18  25   2   9

Dimensions: 5 rows by 5 columns (a square grid)
Row 1:              [17, 24, 1, 8, 15]
Row 2:              [23, 5, 7, 14, 16]
Row 3:              [4, 6, 13, 20, 22]
Row 4:              [10, 12, 19, 21, 3]
Row 5:              [11, 18, 25, 2, 9]
Row sums:           [65, 65, 65, 65, 65]
col sums:           [65, 65, 65, 65, 65]
major diagonal sum: 65
minor diagonal sum: 65
max element:        25
min element:        1
is magic square?    true

Tested isMagicSquare: OK

Congratulations! your program seems to be working correctly!
```

6 FYI

6.1 What's a magic square?

A normal magic square of order n ($n \geq 3$) is an $n \times n$ square grid filled with *all* of the n^2 distinct integer numbers $1, 2, 3, \cdots, n^2$ such that the sum of the numbers on every row, every column, and every diagonal is equal to the magic constant $\dfrac{n^3+n}{2}$

6.2 How do you figure out the formula for the magic constant?

By definition, every distinct integer from 1 to n^2 appears exactly once in a normal magic square of order n, so the sum of all numbers in the square grid is

$$1+2+3+\cdots+n^2$$

Applying Gauss's formula[1]will result in the following simple way of computing the sum:

$$1+2+3+\cdots+n^2 = \frac{n^2(n^2+1)}{2}$$

Distributing the sum equally between the n rows (or the n columns), we conclude:

$$\begin{aligned} \text{Magic Constant of } n \times n \text{ Magic Square} &= \frac{1+2+3+\cdots+n^2}{n} \\ &= \frac{n^2(n^2+1)}{2n} \\ &= \frac{n(n^2+1)}{2} \\ &= \frac{n^3+n}{2} \end{aligned}$$

[1]Thanks to Carl Friedrich Gauss, we know that for any positive integer m, the sum from 1 to m is

$$1+2+3+\cdots+m = \frac{m(m+1)}{2}$$

For example, $1+2+3+4+5+6+7+8+9+10 = \dfrac{10(10+1)}{2} = \dfrac{110}{2}$

7 Evaluation Criteria

Evaluation Criteria

Functionality	Ability to perform as required, producing correct output for any set of input data, Proper implementation of all specified requirements, Efficiency	60%
Robustness	Ability to handle input data of wrong type or invalid value	10%
OOP style	Encapsulating only the necessary data inside objects, Information hiding, Proper use of Java constructs and facilities.	10%
Documentation	Description of purpose of program, Javadoc comment style for all methods and fields, comments on non-trivial steps in all methods	10%
Presentation	Format, clarity, completeness of output, user friendly interface	5%
Code readability	Meaningful identifiers, indentation, spacing, localizing variables	5%