RIT | Golisano College of Computing and Information Sciences
School of Information

ISTE-120 - Computational Problem Solving for the Information Domain I
Homework Assignment 10 (HW10)

DELIVERABLE
Zip the files together and submit it to the MyCourses Assignment folder for this homework. Homework MUST be submitted on time to receive full credit. Homework submitted to the "late" dropbox will receive 80% of your actual grade. Homework not submitted to either dropbox will receive no credit.

ASSIGNMENT: Magic Squares

PROBLEM STATEMENT

An `n×n` matrix that is filled with n^2 unique numbers (no duplicates) is a magic square if the sum of the elements in each **row**, in each **column**, and in the two **diagonals** is the same value.

Here is an example of a magic square:

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

Write a program that reads in the numbers from 1 to n^2 (for some value of n) from the keyboard and tests whether they form a magic square when arranged as a square matrix.

Program Organization Requirements:

Functional Class: `Square.java`

- Declare an ArrayList attribute, `numbers`, to store user input data
- Declare a two-D `int` array attribute, `square`, that will add the user inputs into a square
- Create a method
  ```
  public void add(int i)
  ```
  that adds a number to the ArrayList of `numbers`
- Create a method
  ```
  public boolean isSquare()
  ```
  that checks the size of the ArrayList is a perfect square. If the size is not a perfect square, this returns **false**, otherwise it returns **true**

- Create a method
  `public boolean isUnique()`

  that returns **true** if the ArrayList contains all of the numbers from 1 to n^2, returns **false** otherwise

- Create a method
  `public boolean isMagic()`

  that returns **true** if the array `square` is a magic square, **false** otherwise. This method should compute the row, column, and diagonal sums to test for a magic square

- More attributes and/or methods may be added as needed
- This functional class cannot use:
  - `Scanner`
  - `JOptionPane`
  - `System.out.print`
  - `System.out.println`

Test Class: `TestMagicSquare.java`

- Prompt user for a sequence of integers
- Use `Scanner` or `JOptionPane` class
- Validate user input for the integer values
- Use the `add(i)` method of `Square` to add each number to the ArrayList
- Check if the user input makes a square; terminate the program if not
- Test to see that all of the numbers from 1 to n^2 are in the input. If any problems, terminate the program
- Call `isMagic()` to see if it is a magic square

Sample Runs on following page

Command Prompt - java TestMagicSquare

```
Enter an integer (x to exit): 2
Enter an integer (x to exit): 3
Enter an integer (x to exit): 4
Enter an integer (x to exit): r
*** Invalid data entry ***
Enter an integer (x to exit): 5
Enter an integer (x to exit): d
*** Invalid data entry ***
Enter an integer (x to exit): 6
Enter an integer (x to exit): x

 Step 1. Numbers do not make a square: Program Stopped
Press any key to continue . . .
```

Command Prompt - java TestMagicSquare

```
Enter an integer (x to exit): 1
Enter an integer (x to exit): 2
Enter an integer (x to exit): 3
Enter an integer (x to exit): 4
Enter an integer (x to exit): 5
Enter an integer (x to exit): 6
Enter an integer (x to exit): 7
Enter an integer (x to exit): 8
Enter an integer (x to exit): 8
Enter an integer (x to exit): x

 Step 1. Numbers make a square ***

 Step 2. Numbers are not unique: Program Stopped
Press any key to continue . . .
```

Command Prompt - java TestMagicSquare

```
Enter an integer (x to exit): 1
Enter an integer (x to exit): 2
Enter an integer (x to exit): 3
Enter an integer (x to exit): 4
Enter an integer (x to exit): 5
Enter an integer (x to exit): 6
Enter an integer (x to exit): 7
Enter an integer (x to exit): 8
Enter an integer (x to exit): 9
Enter an integer (x to exit): x

 Step 1. Numbers make a square ***

 Step 2. Numbers are unique ***

 Step 3. But it is NOT a magic square.***
Press any key to continue . . .
```

```
Command Prompt - java TestMagicSquare
Enter an integer (x to exit): 16
Enter an integer (x to exit): 3
Enter an integer (x to exit): 2
Enter an integer (x to exit): 13
Enter an integer (x to exit): 5
Enter an integer (x to exit): 10
Enter an integer (x to exit): 11
Enter an integer (x to exit): 8
Enter an integer (x to exit): 9
Enter an integer (x to exit): 6
Enter an integer (x to exit): 7
Enter an integer (x to exit): 12
Enter an integer (x to exit): 4
Enter an integer (x to exit): 15
Enter an integer (x to exit): 14
Enter an integer (x to exit): 1
Enter an integer (x to exit): x

 Step 1. Numbers make a square ***

 Step 2. Numbers are unique ***

 Step 3. Yes, it is a MAGIC SQUARE! ***
Press any key to continue . . .
```

DETAILS

1. Submit files to the MyCourses Assignment folder
2. Proper data types should be used
3. `JOptionPane` or `Scanner` should be used for input only in the `TestMagicSquare`
4. `System.out.print()` or `System.out.println()` can only used in the `TestMagicSquare`
5. Header comments for all classes and methods must use Javadoc style. Do not generate the JAVADOC, just write the comments in the Javadoc style

Grade Sheet Name: _____

No points if programs do not compile.

Criteria	Max Pts	Pts Earned
Program meets stated requirements:		
Square Class:		
• ArrayList attribute, numbers, used properly	10	
• 2-dim array, square, used properly	10	
• add(int i) method defined correctly	10	
• isMagic() method defined correctly	10	
• Other attributes are applied properly	5	
• Other methods are applied properly	5	
• Did not use Scanner, JOptionPane, System.out	5	
TestMagicSquare Class:		
• Prompt user for a sequence of integers	5	
• Validate user input for int values	10	
• Use the add(i) method of the Square to add each number to the ArrayList	5	
• Check if the user input makes a square. If not, stop the program	5	
• If the numbers make a square, test for uniqueness. If there are any duplicate numbers, stop the program	5	
• Then call isMagic() method if it is a magic square.	5	
• Output matches the samples provided	10	
Points earned	100	
Points Deduction:		
• JavaDoc Comments	0-10	
• In-code Section/variable Comments	0-10	
• White Space, Alignment & Indentation	0-10	
• Variable/Method Names	0-10	
• Zip file should include java & class	0-10	
Grade if up to 1 day late (80% of Pts Earned):		

Note: If your program fails to compile, you will receive a zero for the assignment.

Additional Comments: