INTRODUCTION TO COMPUTER SCIENCE I
LAB 7
Magic!

1 Magic Squares

A *magic square* is an $n \times n$ grid filled with the numbers $1, 2, \ldots, n^2$ such that each number appears exactly once, and the sum of every row, column, and main diagonal[1] in the grid adds up to the same value. For example, the 3×3 grid…

4	9	2
3	5	7
8	1	6

…is a magic square because every row, column, and main diagonal sums up to 15.

Magic squares have been around for a long time. The first known recordings of magic squares date back to the 7th century BC. The particular magic square shown above is called the *Lo Shu* square. An old Chinese legend says that during a great flood of the Lo River, a turtle walked out of the river with a shell pattern depicting the Lo Shu. This pattern helped the people determine the right sacrifice to make to the river gods, thereby ending the flood.

Your work in this lab probably won't stop any floods (but who knows?), but by the end of the lab you will be able to generate magic squares. It is not possible to construct a magic square of size $n = 2$, but all other sizes are possible. There's a nice, straightforward algorithm for constructing magic squares of odd size (see Section 2.3 below), so that will be our focus.

2 Your Job

2.1 Setup

Create a directory for this project, change into it, and grab source code:
`bit.ly/COSC-111-lab-7-source`

The file `MagicSquare.java` currently contains a `main()` method that reads in an odd number from the keyboard and calls the (not yet written) `createMagicSquare()` method. There is also a method called `print2D()` that prints a 2-dimensional array. You can use this method to help test the methods you will write.

[1]The main diagonals of a grid are the two diagonals that both start and end at corners.

2.2 Checking if a square is magic

Your first job is to fill in the `checkSquare()` method. This method takes as input a 2-dimensional array, and returns a `boolean`. The method should return `true` if the array is a magic square (i.e., all rows, columns, and main diagonals add up to the same value), and it should return `false` if the array is not a magic square.

2.3 Generating magic squares

Your second job is to fill in the `createMagicSquare()` method. This method takes as input an `int` representing the desired size of the square (you can assume that the input will be odd), and should return an `int[][]` storing the magic square that you generate and fill in.

Here is an algorithm to fill in a magic square of odd size n:

1. Start by putting the number 1 in the middle column of the top row.
2. Move up 1 and to the right 1 from the last position you filled. If that position already has a number written in it, move down 1 from the last position you filled instead. If you run over the edge of the square, "wrap around" to the other side. Write the next number into this new position.
3. Repeat step 2 until you have filled in all numbers from 2 to n^2.

For example, to fill a 3×3 magic square, the sequence of steps would be:

Step 1:

	1	

Step 2:

	1	
		2

Step 3:

	1	
3		
		2

Step 4:

	1	
3		
4		2

Step 5:

	1	
3	5	
4		2

Step 6:

	1	6
3	5	
4		2

Step 7:

	1	6
3	5	7
4		2

Step 8:

8	1	6
3	5	7
4		2

Step 9:

8	1	6
3	5	7
4	9	2

3 Submitting your work

Submit your modified `MagicSquare.java` using either the CS submission web site.

This assignment is due on Monday, Apr-22, 11:59 pm.