Lab 11 (October 16 or October 17)

Instructions: Complete the exercises below. Be sure to show your code to one of the lab TAs before you leave, so that you can receive credit for this lab. You must also upload a copy of all your source code (.java) files to the link on Blackboard by 11:59 PM on Tuesday, October 17.

1. An *n* x *n* matrix is called a *positive Markov matrix* if each element is positive and the sum of the elements in each column is 1. Define the following method to check whether a matrix of `double` values is a positive Markov matrix:

   ```
   public static boolean isMarkovMatrix (double [ ] [ ] m)
   ```

   Write a test program that prompts the user to enter a 3 x 3 matrix of `double` values and reports whether it is a positive Markov matrix.

   Sample run #1:

   Enter a 3 by 3 matrix, row by row:

   ```
   0.15 0.875 0.375
   0.55 0.005 0.225
   0.30 0.12 0.4
   ```

   This is a positive Markov matrix

   Sample run #2:

   Enter a 3 by 3 matrix, row by row:

   ```
   0.95 -0.875 0.375
   0.65 0.005 0.225
   0.30 0.22 -0.4
   ```

   This is not a positive Markov matrix

2. A *magic square* is a *N* x *N* array of cells where each cell contains one of the first N^2 positive integers. The chief property of a magic square is that the sum of the numbers along any row, column, or major diagonal is a constant. The following algorithm can be used to generate magic squares for odd values of N:

   a. Start by writing 1 in the middle cell of the top row.

   b. Now keep writing 2, 3, 4, ..., moving up and to the left (meaning that the row and column value decrease by 1 for each new number that you add to the magic

square). If you reach the edge of the square, "come out" on its opposite side. For example, if moving to the left means that you cross over the left edge of the square, "wrap around" to the right side of the square and use that cell instead.

c. If you have to write on a square that has already been used, go down one cell from your starting position instead.

For example, here is a 3 by 3 magic square:

```
6   1   8
7   5   3
2   9   4
```

Note that, after placing the value 3, the next cell is already occupied by 1. As a result, 4 is placed in the cell below the starting position (3) instead.

Write a method `createSquare()` that takes an odd, positive integer as its argument and returns a filled-in magic square (a two-dimensional array of integers) of that size:

```
public static int [ ] [ ] createSquare (int size)
```

Hint: Begin by initializing each element of your two-dimensional array to a known invalid or "unused" value like -1. This will simplify step/rule (c) above.

Use this method in a program that prompts the user to enter a size for a magic square, and then prints the resulting magic square. You may assume that the user will always enter an odd, positive integer for the size of the square.

Grading Guidelines: This lab is graded on a scale of 0-3 points, assigned as follows:

0 points: Student is absent or does not appear to have completed any work for the lab

1 point: Student has written only one program, but it does not compile or run at all due to errors.

2 points: Student has written (or attempted to write) both programs, but only one compiles and runs without error.

3 points: Student has written both programs, and they both compile and run correctly, without any apparent errors.