**Magic Square Assignment**

A magic square is an arrangement of distinct integers (i.e. each number is used once, in a [square](http://en.wikipedia.org/wiki/Square_%28geometry%29) grid, where the numbers in each row, and in each column, and both main diagonals sum up to the same value. For example, in a 3 x 3 grid, the following is a magic square since every column, row and diagonal sum up to 15.

***MagicSquare***: Your first program is to calculate and output a *random* 3 x 3 magic square. Add in something to keep track of how many attempts it took to find a "magic square".

|  |  |  |
| --- | --- | --- |
| **Variable** | **Type** | **Purpose** |
| attempts | int | Keeps track of how many attempts it took to find a magic square |
| magic | int [ ] [ ] | 2D Array to hold the numbers; This is the magic square |
| picked | boolean [ ] | This array will remember which numbers have been picked and put in magic square.This could be omitted with some clever thinking. |

**Main Method**

int[][] magic = new int [3][3];

initializeBoard(magic); //Sets the value of all elements in magic to zero.

while (!isMagic(magic))

 {

 initializeBoard(magic); //Reset the board to all 0s if the numbers didn't work

 populateMagic(magic); //Randomly assigns unique numbers (1 to 9) into the magic array

 }

outputMagic(magic); //Outputs the board after a solution has been found.

***MagicSquare2:***  Magic boards can also work with other n x n sized boards, where n is any odd number. Copy your MagicSquare program code into a new class called MagicSquare2. The user will input the size of the board (n). Your will work the same as the first program to find a magic square for the new sized board. Realize that the sum of columns, rows and diagonals changes with a larger board. The new number to check for is calculated with the following formula:

 *(n(n2 + 1)) / 2* (given an n x n sized board, and n is an odd positive integer.)

**MagicSquare3:** Create a new class called MagicSquare3. In this one, you brute force all possible magic square boards. Your goal is to find the total number of solutions possible. Reflections and rotations count as unique solutions. Store all solutions in an ArrayList called solutions. This gives us the ability to retrieve any of the solutions found.

***ArrayList<int[ ][ ]> solutions;***

***solutions = new ArrayList();***