APCS **Cool Set 7**

1. Complete class `MatrixMathOps` to work with `MathMatrix` shown below.

```
/**
 * this is a driver to test the MatrixMathOps class
 * dominguez 2010
 */
public class MathMatrix
{
    public static void main( String args[] )
    {
        int[][] A = MatrixMathOps.generateIntMatrix(2,4,-5,5);
        int[][] B = MatrixMathOps.generateIntMatrix(4,3,-5,5);
        int[][] C = MatrixMathOps.generateIntMatrix(4,3,-5,5);
        System.out.print("A:");
        MatrixMathOps.showMatrix(A);
        System.out.print("\nB:");
        MatrixMathOps.showMatrix(B);
        System.out.print("\nC:");
        MatrixMathOps.showMatrix(C);

        int n = 2;
        //int[][] D = MatrixMathOps.scale(n,A);
        System.out.print("\n*********************************\n" +
                         n + "*A is " );
        MatrixMathOps.showMatrix(MatrixMathOps.scale(n,A));
        //int[][] E = MatrixMathOps.add(B,C);
        System.out.print("\n*********************************\n" +
                         "B + C  is " );
        MatrixMathOps.showMatrix(MatrixMathOps.add(B,C));
        //int[][] F = MatrixMathOps.multiply(A,B);
        System.out.print("\n*********************************\n" +
                         "A * B  is " );
        MatrixMathOps.showMatrix(MatrixMathOps.multiply(A,B));
    }
}
```

OUTPUT

A:

-4 5 -3 -3
5 -3 5 -5

B:

2 5 2
-2 -1 5
-1 4 -5
4 -1 2

C:

3 1 2
3 3 4
-4 0 3
1 -5 1

****

2*A is

-8 10 -6 -6
10 -6 10 -10

****

B + C is

5 6 4
1 2 9
-5 4 -2
5 -6 3

****

A * B is

-27 -34 26
-9 53 -40

```
/**
 * This class is designed to work with rectangular matrices.
 * dominguez 2010
 */
import java.util.Random;
public class MatrixMathOps
{
    public static int[][] generateIntMatrix(int rows, int cols, int lowerLimit, int
upperLimit)
    {
        Random ourRandNumGen = new Random();
        int[][] result = new int[rows][cols];
        int num;
        for(int r = 0; r < result.length; r++)
        {
            for(int c = 0; c < result[0].length; c++)
            {
                num = ourRandNumGen.nextInt(upperLimit - lowerLimit + 1) + lowerLimit;
                result[r][c] = num;
            }
        }
        return result;
    }


    public static int getNumRows( int[][] X )
    { return X.length; }
```

```
    //assume rectangular X is a rectangular matrix
    public static int getNumCols( int[][] X )
    { return X[0].length; }

    public static int[][] scale( int n, int[][] X )
    { ? }

    public static int[][] add( int[][]X, int[][]Y )
    { ? }

    public static int[][] multiply(int[][]X, int[][]Y)
    { ? }

    public static void showMatrix( int[][] X )
    {
        System.out.println();
        for( int k = 0; k < X.length; k++ )
        {
            for( int c = 0; c < X[k].length; c++ )
               System.out.printf( "%-7d", X[k][c] );
           System.out.println();
        }
    }

}
```

2. Write program `FancyWord` to work with the driver program shown below. Notice that each instance of `FancyWord`, called `goal`, is displayed by the line `out.println(goal)`. This means that your `FancyWord` class will have to have a `toString` method that converts a `String` matrix (or a matrix of `String` objects) into a single `String` object and returns this `String` object. Keep in mind that a `String` object can contain escape characters such as `"\n"`.

```
import java.io.IOException;
import static java.lang.System.*;

public class CoolSet7_1
{
  public static void main( String args[] ) throws IOException
  {
      String word;
        FileInput inFile = new FileInput( "coolset7-1.dat" );
        word = inFile.readLine();
        if( inFile.fileFound() )
        {
            try
            {
                while( !inFile.eof() )
                {
                    word = inFile.readLine();
                    FancyWord goal = new FancyWord(word);
                    out.println(goal);
                    out.println();
                }
            }catch( Exception n )
            { System.out.println( "\nEnd of file" ); }
        }
  }
}
```

"coolset7-1.dat"

```
HELLO
CAT
A
DOGHOUSE
PARANGARACUTIRIMICUARO
IT
NEWPORT
GOOGLE
```

Basically, when the driver reads a word that word is used to create a `FancyWord` object. The `FancyWord` constructor creates a square 2D array (using the number of letters in the word) and fills each `String` element in the array with a single-letter `String` object or a space (" "). The `toString` method uses the elements of the 2D array to form a single `String` object which is returned to `out.print()` in the driver and displays an hourglass using this word.

SAMPLE OUTPUT

```
PARANGARACUTIRIMICUARO
 A                   R
  R                 A
   A               U
    N             C
     G           I
      A         M
       R       I
        A     R
         C   I
          UT
          UT
         C   I
        A     R
       R       I
      A         M
     G           I
    N             C
   A               U
  R                 A
 A                   R
PARANGARACUTIRIMICUARO
```

```
NEWPORT
 E    R
  W  O
    P
  W  O
 E    R
NEWPORT
```

```
IT
IT
```

3. Write class TwoDArray to work with the driver shown below.

```
import static java.lang.System.*;
public class TwoDArrayDriver
{
  public static void main(String[] args
  {
    FileInput inFile = new FileInput( "matrix.txt" );
    int row = inFile.readInt();
    int col = inFile.readInt();
    int[][] matrix = new int[row][col];
    for( int i = 0; i < row; i++ )
    {
       for( int j = 0; j < col; j++ )
       { matrix[i][j] = inFile.readInt(); }
    }

    out.println();
    TwoDArray app = new TwoDArray( row, col, matrix );
    app.display();
    app.fun1();
    app.display();
    app.fun2();
    app.display();
  }
}
```

Method `fun1`

changes all even numbers in the array to zero.

Method `fun2`

A cell in any array can have up to four diagonal neighbors (i.e., in the North West, North East, South West and South East directions). The `fun2` method replaces the value of a cell with the number of diagonal neighbors that hold a value of zero.

SAMPLE DATA AND OUTPUT

"matrix.txt"

```
4 4
2 5 4 9
0 5 6 3
1 9 4 6
7 2 6 9
```

row = 4 col = 4

grid[0][0] = 2 grid[0][1] = 5 grid[0][2] = 4 grid[0][3] = 9
grid[1][0] = 0 grid[1][1] = 5 grid[1][2] = 6 grid[1][3] = 3
grid[2][0] = 1 grid[2][1] = 9 grid[2][2] = 4 grid[2][3] = 6
grid[3][0] = 7 grid[3][1] = 2 grid[3][2] = 6 grid[3][3] = 9

2 5 4 9
0 5 6 3
1 9 4 6
7 2 6 9

0 5 0 9
0 5 0 3
1 9 0 0
7 0 0 9

0 2 0 1
0 3 1 2
1 3 1 2
0 1 1 1

4. Write a program to work with the driver shown below that produces a Pascal Triangle of any size.

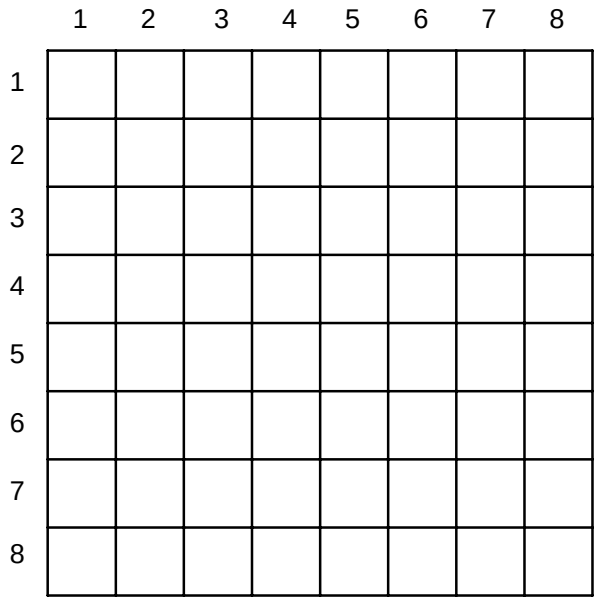
```
import static java.lang.System.*;
public class PascalsTriangleDriver
{
  public static void main( String args[] )
  {
      out.print("Enter number of rows in Pascal's Triangle . . . ");
      int size = SavitchIn.readInt();
      PascalsTriangle test = new PascalsTriangle(size);
      test.createTriangle();
    }
}
```

SAMPLE OUTPUT

```
Enter number of rows in Pascal's Triangle . . . 9
                    1
                  1   1
                1   2   1
              1   3   3   1
            1   4   6   4   1
          1   5   10  10  5   1
        1   6   15  20  15  6   1
      1   7   21  35  35  21  7   1
    1   8   28  56  70  56  28  8   1
```

5. The Swiss mathematician Leonhard Euler (1707 – 1783) proposed a problem regarding the movement of the knight chess piece on a chess board. The challenge that Euler proposed was to move the knight around an empty chessboard, and to touch each of the 64 squares *once and only once*.

   First, to understand what you have to do, move a knight from any position on the board using its standard L-shaped moves (two over in one direction, then one in a perpendicular direction). Number any position as 1 and then visit as many squares as possible, numbering each move as you go:



Assignment:

Your task in this lab is to write a program that will move a knight around an empty chess board, leaving behind a trail of increasing integers, ranging from 1 to, hopefully, 64. Here are the specifications for your assignment:

1. The knight will start in row 1, column 1.
2. The program will mark squares as they are visited, ranging from 1-64.
3. The program will continue until a complete tour is accomplished (all 64 squares) or the program gets stuck with nowhere to go.
4. The program will print the results, looking something like this:

```
      1    2    3    4    5    6    7    8

1     1    0   21    0    0   14   23   12
2    20    0    6    9   22   11    0    0
3     7    2   19   36   15   46   13   24
4     0    5    8   47   10   37    0   45
5     0   18    3   16   35   44   25   38
6     4   31   34    0   42   39   28    0
7     0    0   17   32   29   26   43   40
8     0   33   30    0    0   41    0   27

47 squares were visited
```

Now, once you get started on this assignment you will quickly discover that this is actually a pretty tough problem. There are two approaches. Adhering to the basic moving rules (two squares up/down, one over left/right; one square up/down, two over left/right) you can . . .

a) move your knight randomly. I discourage you from pursuing this approach because chances are slim to none that you will actually hit all 64 squares.

b) develop an algorithm that uses <u>logic</u>. If we analyze each square, we notice that some are more accessible than others. The squares on the edges are more difficult to visit than the squares not on an edge. By this logic, you can see that the corners are the most difficult, or inaccessible, squares of all.

Here is an analysis of the accessibility of each square.

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

A square with an accessibility of 8 means that it can be approached from 8 different other squares. A corner square is rated at 2, while the edges are rated at 3 or 4. It makes sense to try and visit squares with lower accessibility values first, leaving the more accessible middle squares for later in the algorithm.