# Lab 2

## *Description*

The purpose of this lab is to write a program in stages as specified below.

In this lab you will write a program to emulate the game of tic-tac-toe between two people at the same console. That is to say that in this lab your program will emulate pencil and paper (a tic-tac-toe "board").

When you are done, a typical session will be as follows.

```
 Welcome to tic-tac-toe.

 Enter coordinates for your move following the X and O prompts.

       1 2 3
     A  | |
        -----
     B  | |
        -----
     C  | |

X:A2

       1 2 3
     A  |X|
        -----
     B  | |
        -----
     C  | |

O:B3

       1 2 3
     A  |X|
        -----
     B  | |O
        -----
     C  | |
```

and so on. Illegal moves will prompt the user again for a new move. A win or a stalemate will be announced, mentioning the winning side if any. The program will terminate whenever a single game is complete.

For this lab, you will be provided with a base file to work with. The base file can be downloaded from:
https://github.com/victoryu/CIS35A-Labs

You will add and/or modify the code, as instructed below. Do not change the overall structure of the program. Just fill in with your code at TODO and Step #.

This file has the general framework of the `TicTacToe` class. An object of this class will represent a tic-tac-toe "board". The board will be represented internally by a two dimensional array of characters (three by three), and by a character indicating who's turn it is ('X' or 'O'). These are stored in the class instance variables as follows.

```
private char[][] board;
private char player; // 'X' or 'O'
```

Step1: Fill in the constructor to make an unused "board". The array should be filled with nine space characters and the player should be initialized to 'X'.

Write some test code in your main method to create a `TicTacToe` object and print it using the print method given, so as to test your code. Your print method should produce the first board in the example above.

Step 2. Now we will fill in the play method. The String s should be of the form "A1", "B2" etcetera. If the play is legal (i.e. has not been taken by either player yet), then the method should write the appropriate character in the right place of the board array, and return true. Whether 'X' or 'O' is written depends upon the value of the player variable. If the play is illegal, the method should simply return false, including the case where the String s is not of the correct form, (A, B or C followed by 1, 2 or 3).

Step 3. Fill in the method `switchTurn` to toggle the player variable between 'X' and 'O'.

Modify your main program to call play and `switchTurn` a few times, printing the board between each to verify that it works. Simply hard code the moves into your java program for now.

Step 4. Fill in the code for the stalemate method. It should return true if there are no more places to move on the board. Otherwise, return false.

Step 5. Fill in the code for the won method. This method should return true if the current player has 3 in-a-row in any row, column or diagonal. Otherwise, return false.

Step 6. Comment out or delete your main method and uncomment ours at the bottom of the file. Follow the comments in the method to finish the program. Each comment corresponds to either one or a few lines of code. You will need to call the methods you wrote on the `TicTacToe` object called game we created in our main method.

## *Lab Submission*

- Your program should follow the coding guideline
- Make a directory named LastnameFirstname_lab2 and create all your files there:
    - A test plan that includes at least 5 test cases (scenarios, input, expected result, and actual reports) and a brief explanation of the limitations

    - All the source code
    - Execution of the test cases (sample output)
- Copy the directory to G:\\vyu\CIS35A\DropBox
- Click the "Submit" button at the course website.

## *Grading*

The total point for this lab is 10.

| | |
|---|---|
| 10: | A completely functional game that is neatly laid out. Coding standards are followed. All the required documents are submitted (test plan with 5+ test cases, sample output) |
| 9.0-9.9: | A completely functional game, but the code does not meet the coding standard. All requirement documents are not submitted. |
| 8.0-8.9 | The game works but messy and poorly designed. Some or all documents are missing |
| 7.0-7.9 | The game is mostly working but has some functional issues or glitches. |
| 6.0-6.9 | The game is partially working, with only some function implemented. |
| 5.9 or below | The game is mostly not working. |