Independent Programming Assignment 5

CHAPTER 8, 14 – Battleship (Multi-Dimensional Arrays and ArrayLists)

Assigned/Due: See zyBooks for exact due date

*All portions of this project due by 11:55pm on the due date.

YOU MAY NOT WORK WITH OR WRITE CODE TOGETHER WITH A PARTNER.

This assignment will be graded on the following criteria:

Compiles and runs	30%	(Your program **MUST** compile in order to be considered for grading)
Correctness	40%	(Your program must satisfy each requirement of the specifications)
Style	20%	(Your program must use comments and have user-friendly output)
Instructions	10%	(You must include any other materials requested in the lab)

Description

Write a Java application for a 1-player version of the popular game called Battleship. The main learning objectives of this lab are to demonstrate a practical understanding of maps, multi-dimension arrays and ArrayLists.

Your battleship game must conform to the following standards:

- Programming structure requirements:
  - Uses a map to store the number of previous guesses for any particular row-column spot (e.g., the key should be some kind of row-column representation and the value should be an integer representing the number of previous guesses for that row and column)
  - Uses a 2D array to store the state of the board (you choose what data type); the **board should be 7x7 cells**.
  - Uses one or more ArrayLists to store the coordinates of the players guesses in sequential order (you will be asked to print them all at the end)

- Game-making requirements:
  - A JAR file has been included to use as reference for output style and program flow.
  - The program begins by asking a game-making user to enter the **starting coordinate and direction to place the ship (rightward or downward)** of 3 different battleships (of **length 2, length 3 and length 4**). This can be done by asking 9 consecutive questions (row # (0-6), col # (0-6), and orientation for each ship ('r' or 'd')).
    - Upon finishing the placement of all pieces, the entire board is printed out to confirm placement (see below for example)
      - Assume all entries will place the ship properly on the board (i.e., **you do not need error handling to ensure your placement doesn't extend off the board or that there is not already a ship overlapping with the new ship's location**)
      - Row-column numbering must be included when printed
      - Show the location of all 3 ships (**use an 'S' to denote a ship on a cell, use a '-' to denote open water**).
      - You do **not** need to distinguish between the 3 ships when printing
      - Make sure to print a number of new-line characters after printing the board so it is off the screen for the user to begin guessing

Example of printed game-maker board

r\c	0	1	2	3	4	5	6
0	-	-	-	-	-	-	-
1	-	-	-	-	-	-	-
2	-	S	-	S	S	S	-
3	-	S	-	-	-	-	-
4	-	-	-	-	-	-	-
5	S	S	S	S	-	-	-
6	-	-	-	-	-	-	-

- Game-play requirements:
  - Once the board is set, the user is **repeatedly** asked to enter a row and column (can be done in two questions which expect integers from 0-6)
  - If the user guesses:
    - The location of a ship, print that it was a "HIT!"
    - An empty location, print that it was a "MISS!"
    - A cell that was already guessed, print "r\c = 0\0 has already been guessed 1 time", for example

- **After each guess, print the board for the player (see below)**
  - Row-column numbering must be included when printed
  - **Use an 'X' to denote a hit, a 'm' to denote a miss, and a '-' to denote an unguessed cell**

Example of printed player board after guessing R/C = 1/2, 5/3

r\c	0	1	2	3	4	5	6
0	-	-	-	-	-	-	-
1	-	-	m	-	-	-	-
2	-	-	-	-	-	-	-
3	-	-	-	-	-	-	-
4	-	-	-	-	-	-	-
5	-	-	-	X	-	-	-
6	-	-	-	-	-	-	-

- You do not need to notify the user of individual battleships being sunk, but, **when all ships have been sunk (i.e., all 9 locations have been hit) the game is over**
- When the game is over, print out the **total number of moves** and **all row-column guesses** that were made like so (i.e., a history of all the guesses):

```
Guess   | Row   Col
-------------------------
1       | 1     2
2       | 5     3
3       | 5     2
4       | 5     1
5       | 5     0
6       | 2     1
7       | 3     1
8       | 2     3
9       | 2     4
10      | 2     5
```

  - Again, you must use one or more ArrayLists for the history, although how you store the information in the ArrayList(s) is your choice. History should include repeat guesses.

NOTE: As a side note, a 2-player game can be imagined by running two 1-player games in parallel as two separate java applications…the winner is the one who took less turns to sink all the battle ships.

Submission Instructions

1.) Templates: Use the relevant template file(s) for this lab found in the Google Drive
   a. Use the specified .java source file template for each [CODE] problem and **DO NOT change the file or class names** (doing so will cause your code to receive a 0 by the autograder)
      i. Make sure to update the header and name *System.out.print* statements
   b. Submit the console results (showing input and output) for at least **2 different test cases** you created to convince yourself that your program is working properly; place test cases at the very end of your .java file in the space provided by the template for test cases.
2.) zyBook Submission: **Make sure your name is on all files you turn in.**
   a. Submit the relevant .java file(s) under the appropriate lab assignment
      i. Check to make sure all the files contain your latest work before submitting
   b. The final grade and grading feedback will be returned to you via the Canvas assignment section