ASSIGNMENT ONE: Multithreaded Battleship Game - Server and Client

Description

This assignment will utilize your programming skills in socket programming, threading, and use of the Observer pattern. These skills will be used to develop a multithreaded socket program of the Battleship game implemented with a server and game client. Both the client and the server must have a GUI interface. The server's GUI will be used to display and monitor game traffic; the client's GUI will contain a fleet map, a target map, and any other components required to play the game and chat during gameplay.

Notes

- This is your first assignment and is worth **6%** of your final grade in this course.
- See documentation below on submission and evaluation of assignment.
- Below you will find the specifications for your application, along with some hints on how to proceed.
- Read the specifications very carefully. If you are uncertain about any of the requirements, bring your code to your instructors and they will discuss it with you, or e-mail them.

Specifications

You will be required to write a:

1. Multithreaded server capable of supporting multiple pairs of Battleship players logged on to chat and play the game concurrently.
2. Client that will allow the user to register on the server and play the game of Battleship with the next available player logged on to the server. Communications between the 2 players are handled using the observer pattern as discussed in class.
3. The game server will wait until two players have logged on to the server. It will pair these players off under a separate thread of control which, when created, will notify each player that they can start playing the game. The game will randomly select which player goes first.
4. At the end of each game, the players should be given a choice if they wish to continue playing.
   a) If they both choose yes to quitting, they are logged off the system and their sockets are dropped.
   b) If one responds yes and the other no, then the player responding yes should be paired up with the next available player on the server. The chat history should remain on-screen.
   c) If both players wish to keep playing, then the gameboard will reset and the player who won the last game will go first in the next game. The chat history should remain on-screen.

Battleship Rules:

The map grids are to be set up as a minimum 10 X 10 grid.

The ships in a fleet are:

1. Aircraft carrier occupies 5 squares
2. Battleship occupies 4 squares
3. Cruiser occupies 3 squares
4. Submarine occupies 3 squares
5. Destroyer occupies 2 squares

Note: Ships are placed only horizontally or vertically (not diagonally).

Once players are selected for a game to begin, they must be allowed to place their ships on their tactical map before game play can resume.

Submission Deliverables

1. Assignment zip file will be uploaded into **D2L** by the specified due date and time, also, named as follows: your first initial followed by last name i.e. JSmith.zip.

   The assignment zip file will include the following:
   a. An executable Java Archive file (.jar) for you client application and one for your server application. Your applications should have separate server and client driver classes i.e. **ClientDriver** and **ServerDriver**.
   b. Two readMe.txt files with instructions on how to install and use the server and client programs.
   c. The project should have completed javadoc using **"-private"** option when generated, with output placed in the doc directory of the project.
   d. A folder containing the complete Eclipse project directory. Your project must be in Eclipse format (i.e. can be imported into the specific development environment as specified by your instructor). Please do NOT compress (zip) this folder.
   e. At the root of the submission folder, include a readMe.txt to describe the completeness of the assignment (as a percentage) and a list of known deficiencies and/or missing functionalities.

2. Upload the executable Java Archive file (.jar) of your client to the instructor specified discussion board in D2L for other class members to download and install. This jar file should be placed in a zip file with a readme.txt detailing installation and operation instructions for your client software.

3. A Live Demonstration in class with support of functionality (i.e. on-screen logging of server activity) will be part of the assessment for this assignment. Student will plug their laptop into overhead projection system and start their server and allow classmates to log on to the server and play their game.

Submission Criteria

Three levels of acceptable functionality will be allowed for this project, submissions below these levels will result in an automatic zero grade. These levels are defined as:

1. **50% complete:** Server is multi-threaded (i.e., can support multiple clients) and clients can communicate with the server. The server must be able to return information to the client.

2. **75% complete:** Server can accept two clients and allow them to play a game of Battleship while logged onto the server.

3. **100% complete:** Server can maintain multiple games of Battleship being played concurrently. When a game ends, the players will be given the choice to play again or quit. In addition, the game will allow both players to chat before, during, and after each game.

Your program must execute. Code that doesn't run/compile will not be accepted.

No late Assignments will be accepted and will be assigned a mark of 0%.

You are expected not merely to solve the problems, but to make intelligent decisions about how to approach the problems. Remember, it is not enough to submit adequately coded programs that merely produce the expected output, but should be elegant, designed legibility, have maintainability, and other such important factors.

Please note that proficiency in using official Java API Doc can answer a lot of questions you may have about using some standard Java classes and methods, and many other types of programming which you will encounter, both at SAIT, and beyond.

Criteria for Marking Assignment:

Demonstration of implemented game (in-class)		/	**5**
Appealing, intuitive, user-friendly GUI for the client side		/	**5**
Readmes/Instructions, javadoc documentation		/	**2**
Proper system/project structure and naming		/	**2**
Server GUI shows real-time log of connections and game traffic		/	**2**
Use of streams		/	**1**
Use of sockets		/	**1**
Use of threads		/	**1**
Use of the Observer pattern		/	**1**
Exceptions are handled properly		/	**2**
Players can chat during game		/	**2**
Players can play game and game logic is correct		/	**2**
Players can be reinstated to play via the server		/	**2**
Synchronization of key method(s)		/	**2**
Total		/	**30**