CS 3800 Computer Networks

Assignment # 3

Points: 25

- **Submission**
  - **Zip the documents for your solutions. Name the file Lastname_firstname_HW2.zip**
  - **Submit your zip on Blackboard under Assignments->Assignment3.**
- **Submission deadline**
  - **Deadline: 10/28 11:59 PM**
- **Late Submission Policy**
  - **You can do a late submission until 10/31 11:59PM with a 5% penalty on the total points for this assignment.**
  - **After that no submission will be accepted**
- **Early Submission**
  - **You can also get 5% extra point (on your score on the assignment) for early submission if you submit by 10/25 11:59PM.**
- **Getting help**
  - **From instructor during office hours or by email.**
- **Academic Dishonesty Policy**
  - **Submission will be checked for plagiarism. This is individual work.**

Note: Please post your questions about the assignment to the discussion list on BB. Also make sure to "subscribe" to the discussion list.

1. 2- Player Battleship game [25pts]

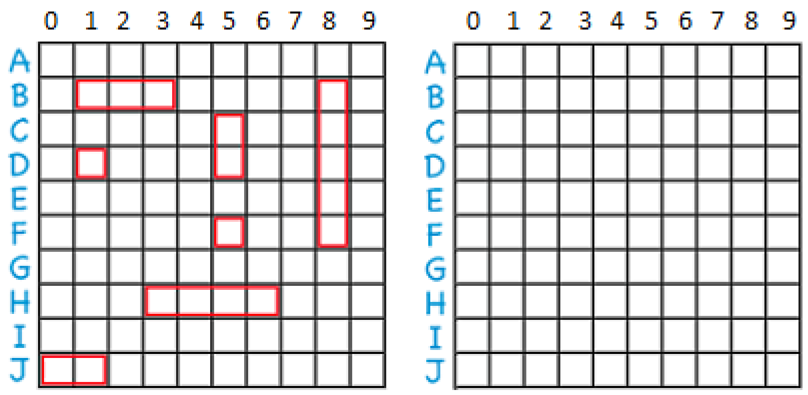
You will implement an online version of the two player Battleship board game. You will design and implement a multithreaded server and clients in JAVA using sockets that can handle two players. The players will use the clients to play with each other. The server will control the communications between the clients.

Description of the Battleship game:

Players: Two

Players take turns in trying to guess the locations of the other player's ships on a grid.

Each player draws two 10 x 10 grids, labelled along the sides with letters and numbers. On the left-hand grid the player secretly draws rectangles representing their fleet of ships:



The fleet

Each player's fleet consists of the following ships:

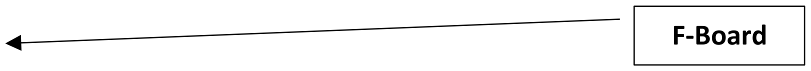
- 2 x Aircraft carrier - 5 squares
- 2 x Destroyers - 3 squares each
- 2 x Submarines - 1 square each

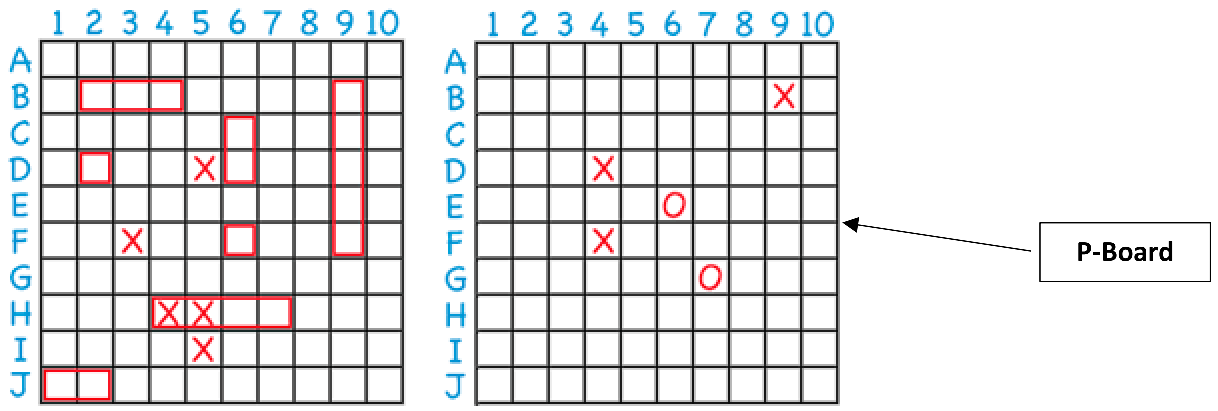
Each ship occupies a number of adjacent squares on the grid, horizontally or vertically.

Play

During play the players take turns is making a shot at the opponent, by calling out the coordinates of a square (eg D5). The opponent responds with "hit" if it hits a ship or "miss" if it misses. If the player has hit the last remaining square of a ship the opponent must announce the name of the ship; eg "You sank my battleship".

During play each player should record their opponent's shots on the left-hand grid, and their shots on the right-hand grid as "X" for a hit and "O" for a miss:





The first player to lose all their ships loses the game.

Implementation

You will implement and submit the following classes. You may implement additional JAVA classes as you see fit.

Client.java: The clients will connect to the server using TCP sockets. The user will provide the following input using std input on the command window.

1. Initial Input: Each player will select the grid blocks for his or her fleet. Each grid block is labelled as “A1” where the first letter is for the row and the subsequent digits are for the column. As shown in the figures above, each board has 10x10 grid blocks and the rows have values ranging from “A” to “J” and columns have values ranging from “0” to “9”. The player will need to only type in the labels of the first two blocks of a ship (except for submarine which requires just one grid block) in the fleet and the client will mark the grid blocks on the board for the entire ship. The board (See BattleShipTable.java) has already been implemented for you. The class uses a 2D array. A grid block are addressed by both its label and X-Y coordinates in the array. For example, the grid block A1 has the X-Y coordinates (0,1) in the 2D array. The BattleShipTable class also has the appropriate methods for marking the blocks for the ships on the board.
2. Play input: Over the course of the game, the player will select a grid block on the opponent’s board to “bomb”, and type in its label.

The client will display the following outputs:

1. Player’s board: After each round, the client will display the player’s board with his or her ships indicted by appropriate letters (e.g. “A” for Aircraft carrier”, “D” for destroyer and “S” for submarine. The grid blocks that have been bombed by the opponent will also be shown in the board.
2. Opponent’s board: After each round, the client will display the opponent’s board with the player’s hits and misses until that point. If a ship has been successfully hit, then that information is also displayed on the board. During the player’s turn, he/she will pick a grid block to bomb.

Server.java: The server will use a *ServerSocket* object (this is done by the main thread) to listen for connections from the clients on port 5000.

When a connection from a player is received, it can lead to two possible actions:

1. If the player is the first to arrive for a game, a new thread (let’s call it the game thread) will be created to initiate the game and do all the actions related to the game. However, it will first need to wait for the second player before the game can be initiated.
2. When the second player arrives, the main thead (with the ServerSocket) will pass the information of the player to the game thread. The game is now initiated.

Once the two players are connected to the server, the game will proceed forward in rounds. During each round of the play, a player will be presented with two boards - **F-Board**: the board with his/her fleet marked and the hits/misses of the opponent, and **P-Board**: the board with the players’ hits and misses on the opponent’s fleet. See the figure.

1. Initial round: The player will provide the location of the ships in their fleet to the server

2. Game rounds: The game starts with the first player asked to select a grid block on the second player’s board to bomb. The second player’s board is updated with a hit or a miss and displayed to the first player. A ship is said to be sunk if all the grid blocks occupied by the ship have been bombed by the opponent. The information on the ships that have been sunk and their locations are updated on the F-Board of the second player and presented. The second player is asked for his input (i.e. bomb a grid block on the first player’s board). The game proceeds in this manner until one of the player is successful in sinking all the ships.

Remember, each player's fleet consists of the following ships:

- 2 x Aircraft carrier - 5 squares each
- 2 x Destroyers - 3 squares each
- 2 x Submarines - 1 square each

Here is a rough pseudocode that you may use to implement the game

Step	**Server**		**Client (Player 1)**	**Client (Player 2)**
	Main thread	*Game Thread*		
1	Listen for connection using *ServerSocket*			
2			Send out connection request to Server on Port 5000	
3	Accept request and spawn a game thread			
4	Listen for connection using *ServerSocket*	Send MSG_REQUEST_INIT to player 1		
5			Prompt user to provide fleet information	

6			Send MSG_RESPONSE_INIT to server	
7		Receive MSG_RESPONSE_INIT from player 1.		
8		Wait for player 2 to Join		
9				Send out connection request to Server on Port 5000
10	Accept request from player 2 and share the socket with the game thread			
11	Wait for the game to end	Send MSG_REQUEST_INIT to player 2		
12				Prompt user to provide fleet information
13				Send MSG_RESPONSE_INIT to server
14		Receive MSG_RESPONSE_INIT from player 2.		
15		Send MSG_REQUEST_PLAY to player 1		
16			Display the F-Board and P-Board to the player. The player selects the grid block on the P-Board to bomb	
17			Send MSG_RESPONSE_PLAY to server	
18		Server updates the F-board of player 2 and P-board of player 1 using the response from player 1. It will note any ships that get sunk.		
19		Send MSG_REQUEST_PLAY to player 2		
20				Display the F-Board and P-Board to the player.

				The player selects the grid block on the P-Board to bomb
21				Send MSG_RESPONSE_PLAY to server
22		Server updates the F-board of player 1 and P-board of player 2 using the response from player 2. It will note any ships that get sunk.		
23 to N	Do steps 15- 22 until all the ships of a player has been sunk			
N+1		Server sends out MSG_REQUEST_GAME_OVER to both players		
N+2	Main thread exits		Print result. Close socket and exit	Print result. Close socket and exit

MSG_REQUEST_INIT: Request the initial board of the player from the client
MSG_RESPONSE_INIT: Response to the server with the initial board (with the fleet information) of the player
MSG_REQUEST_PLAY: Send the F-Board and P-Board of the Player. The message contains information on what ships has been sunk on both the player's and the opponents' board.
MSG_RESPONSE_PLAY: Response with the grid block of the opponent that the player selects to bomb.
MSG_REQUEST_GAME_OVER: server send the final F-Board to both players and a message announcing the winner

A Message class with the information on the message types is provided as attachment that you may use.

Note the following:

1. Use the code stubs BattleShipTable.java and Message.java provided as attachments. You can add additional variables, methods and user defined classes as needed.
2. You can send and receive the messages in the form of serialized objects (see how object serialization is done in JAVA) through the sockets.
3. Your application need only handle two player at a time.
4. Test your code before submission

Submission: Provide all the .java and .class files in sub-folder called "Battleship-lite". Provide screen shots of your client with responses as a MS Word file.

2. Extra-credit: N- Player Battleship game [5pts]

Now extend your implementation to handle any number of players at the same time. Your application should pair up the players and support multiple games simultaneously. Test your code before submission.
Submission: Provide all the .java and .class files in sub-folder called “Battleship-full”. Provide screen shots of your client with responses as a MS Word file.

Zip the sub-folder(s) and submit the zip file on BB.