AP Computer Science Java
Units 6, 8 Lab - Battleship

Objectives

The objectives of this lab are to practice creating classes, linking classes with constructors, writing getter and setter methods (accessors and mutators), and reinforcing concepts of object-oriented program. The goal is not to create a final product, but to take time to explore the big ideas of OOP and Java classes.

Instructions

In this assignment, you will create a set of simple classes for playing the game Battleship. Battleship is played on a square grid of 10 rows and 10 columns (Labelled w letters down the left side and numbers across the top), which you will represent using a 2-D array of **chars** in a class named **Board**. A separate class, **Battleship**, will allow you to try playing your game and see whether the methods work as planned. Throughout build and testing phases, a runner or driver class is recommended to test each method as you develop the larger program.

Throughout the project you may be building and testing components from each individual class you create, hence you'll need at least the following files:

- `Board.java`
- `Battleship.java`
- `Runner.java`

Preamble/Setup

The challenge of this project is to decompose the game into many smaller steps, normally abstracted with simpler instructions. Consider that the game is really an **object** with a set of instructions sitting on a shelf in a box. To play the game, you find another player, take down the box, open the box and begin the setup of the game. First (if it was anything like playing board games at my house), you check the pieces are all there.

You'll need:

1. 2 Boards (each board is made up of 2 playing grids, 10x10 with a little hole for pegs in each of the 100 squares
2. 2 sets of Ships – Ships are: Carrier (5 holes), Battleship (4), Destroyer (3), Submarine (3), and the Patrol Boat (2).
3. About 50 pegs, of two different colors, one for marking a 'miss' (usually white pegs) and one for marking 'hits' (usually red pegs). On some versions of the game, 5 peg holes sit atop of the flip up lid to indicate number of boats sunk.
4. A set of instructions are needed to guide users with objectives, setup, and rules for play.

Playing Battleship

It is important to understand HOW the game is played using paper (original version) or using the actual board game elements. The setup of the game includes setting up the boards (shaped like a laptop the boards serve to hide the players placement of ships and pegs for both the ships and guesses of the other players' board.

Players start the game by placing their 5 ships on the lower board, being sure that all boats are positioned within the 10x010 grid, and not overlapping another ship of hanging off the edge of the board. Once completed, players take turns launching missiles at the opponent's board, by calling out coordinates, like 'A3'. The other player will report 'hit' or 'miss'. (A variation suggests reporting which ship was hit, but we will not do that in this version of our game, but it certainly could be added as a hacker edition.)

Coding Battleship

As we covert each of these elements from the board game, to a simulated computer game, we'll talk about each element and it's role in the game as well as if the part of the game should be a variable, a method(), or a class.

Each player's 'game unit' holds the ships (lower) and guesses (upper) boards that are a 10x010 grid of holes, it also contains storage for the two colored sets of pegs, and a place to hold the 5 ships that are placed on the lower board during setup.

Let's begin with the `Board.java` file. This class will hold the two boards, the constructor for the setup of the game, and the methods to interact with the board and allow game play.

A ship may be placed horizontally or vertically on the board. However, placing a board on the ship must involve prompting the user for the starting coordinates, the length of the ship and the orientation. A placeShips( ) in the Board class should provide feedback if the placement worked or not, and if not a re-prompt to   method In your program, you will represent blank squares on the board using the String "-", and squares where ships have been added with the String "b". For example, the board below contains 3 boats, 2 of length 5 and one of length 3:

```
- - - - - - - - - -
- b b b - - - b - -
- - - - - - - b - -
- - - - - - - b - -
- - - - - - - b - -
- - - - - - - b - -
- - - - - - - - - -
- - b b b b b - - -
- - - - - - - - - -
- - - - - - - - - -
```

Players then take turns to try and shoot the ships on each other's boards by choosing a row and column reference for each attempted shot. When a player chooses a square containing a ship, that square is marked as a "hit": in your program, this will be done using the String "x". When the player "misses" by choosing a blank square this will be marked with the String "m" in your program. The game is over when one player has shot every square containing part of a ship on their opponent's board. The board below is from a game in progress. One ship has already been completely destroyed and 2 squares of one of the other ships have been shot.

```
- - - - - - - m - -
- x x x - - m x - -
- - - - - - - x - -
- - - m - - - b - -
- - - - - - - b - -
- m - - - - - b - -
- - - - m - - - - -
- - b b b b b - m -
- - - - - - - - - -
- - - - - m - - - -
```

The specification for the Board class is as follows:

Variables

private String[][] squares

- An array which represents the board on which a player places their battleship and records shots.

Constructors

public Board()

- This constructor initializes the squares array with every value set to "-", which is the String used to represent a blank square.

Methods

- `public String toString()`
  - Returns a multi-line representation of the board by concatenating all the values in squares with a new line for each row and spaces separating the Strings in each row.
- `public boolean addShip(int row, int col, int len, boolean horizontal)`
  - Attempts to add a ship of length len to the grid, starting at the row and column specified and proceeding either rightwards (if horizontal is true), or downwards (if horizontal is false). If the ship can be placed in the place specified, each square making up the ship should be set to "b", and the method should return true. A ship may not be placed if it would go off the grid, or would intersect another ship on the grid. If the ship cannot be placed, no values in squares should be changed and the method should return false.
- `public boolean foundShip(int len)`
  - Search the board for any possible ships of length `len`. If there are exactly `len` consecutive squares (either horizontal or vertical) containing a "b" String somewhere in the grid, then return true, otherwise, return false.
- `public int shoot(int row, int col)`
  - If row and col specify a square which is out of bounds, the method should return -1. If the square at the specified row and column contains "-" (i.e. is blank), the square should be changed to "m" to signify a miss, and the method should return 0. If the square contains "b" (i.e. a battleship which hasn"t been hit yet) this square should be changed to an "x" to signify a hit, and the method should return 1. If the square contains either "x" or "m" the method should return 2 (these are squares which have already been “shot”).
- `public boolean gameOver()`
  - If the String "b" does not appear at any location in squares, then there are no unsunk ships remaining on the board, so return true to indicate that the game is over. Otherwise return false.

Along with your Board class there is another class in a separate file named Battleship. This class is a simple implementation of a game which allows you to add ships to a single Battleship board using user input and shoot at these ships. A sample run of the main method from this class is shown below.

Sample Run

```
Welcome to Battleship!
Type "a" to add new ship, "b" to see the board, "p" to play or "q" to
quit.
a
Starting in which row?
2
Starting in which column?
3
How long?
3
Horizontal (h) or vertical (v)?
h
New ship added!
Type "a" to add new ship, "b" to see the board, "p" to play or "q" to
quit.
p
You need ships of length 3 and 4 to play!
Type "a" to add new ship, "b" to see the board, "p" to play or "q" to
quit.
a
Starting in which row?
4
Starting in which column?
1
How long?
4
Horizontal (h) or vertical (v)?
v
New ship added!
Type "a" to add new ship, "b" to see the board, "p" to play or "q" to
quit.
p
Ok, let's play!
```

```
Press "s" to shoot at a square, "b" to see the board, "q" to quit
s
Input row
5
Input column
6
Miss!
Press "s" to shoot at a square, "b" to see the board, "q" to quit


b
- - - - - - - - - -
- - - - - - - - - -
- - - b b b - - - -
- - - - - - - - - -
- b - - - - - - - -
- b - - - - m - - -
- b - - - - - - - -
- b - - - - - - - -
- - - - - - - - - -
- - - - - - - - - -
Press "s" to shoot at a square, "b" to see the board, "q" to quit
s
Input row
2
Input column
3
Hit!
Press "s" to shoot at a square, "b" to see the board, "q" to quit
s
Input row
2
Input column
4
Hit!
Press "s" to shoot at a square, "b" to see the board, "q" to quit
s
Input row
2
Input column
5
Hit!
Press "s" to shoot at a square, "b" to see the board, "q" to quit
s
Input row
2
```

```
Input column
5
You already tried that
Press "s" to shoot at a square, "b" to see the board, "q" to quit
b
- - - - - - - - - -
- - - - - - - - - -
- - - x x x - - - -
- - - - - - - - - -
- b - - - - - - - -
- b - - - - m - - -
- b - - - - - - - -
- b - - - - - - - -
- - - - - - - - - -
- - - - - - - - - -
Press "s" to shoot at a square, "b" to see the board, "q" to quit
s
Input row
4
Input column
1
Hit!
Press "s" to shoot at a square, "b" to see the board, "q" to quit
s
Input row
5
Input column
1
Hit!
Press "s" to shoot at a square, "b" to see the board, "q" to quit
s
Input row
6
Input column
1
Hit!
Press "s" to shoot at a square, "b" to see the board, "q" to quit
s
Input row
7
Input column
1
Hit!
Game over!
```

You should run the game several times and try different inputs - this uses the methods from your Board class, so these will all need to produce the correct results and outputs if the program is to work as desired. You can also add, edit, or isolate code to help you with testing specific features if you wish. Don't however add a main method to your Board class to test methods or your code won't be scored correctly.

Milestones
As you work on this assignment, you can use the milestones below to inform your development process:

Milestone 1: Write the constructor code: you need to initialize the squares variable, and then set every cell in squares to "-" by using nested loops. Write the toString method, again using nested loops to print all values separated by spaces and with line breaks in appropriate positions. Test these methods by running the Battleship class (throughout this program you can type "b" to print the toString representation of the Board being used).

Milestone 2: Write the addShip method. It may be helpful to split this method into two parts: one where the parameter horizontal is true, the other where it is false. In each case you will need to use if statements to determine whether all parts of the ship are on the board, then iterate through the intended squares for the ship to determine whether there is already a ship in the way. If the ship can be placed you can iterate through the relevant squares again, changing them to "b". Don't forget to add returns to indicate what the outcome of placing the ship is. Test your method thoroughly using the runner class.

Milestone 3: Write the foundShip method. Again you will probably need to think about horizontal and vertical cases separately: you will need to check for both types of ship in this method. When iterating through consecutive squares (either horizontal or vertical) you will need a variable to keep track of the length of a continuous row of squares containing "b". Remember to check the value of this and reset it when there is a non-"b" square or the end of a row/column is reached.

Milestone 4: Write and test the shoot method. This method just requires testing the value of that square using if statements, and returning the appropriate value, and changing the value of that square if necessary. Write the gameOver method for which you just need to iterate through all values in squares again, and return false if any is a "b", returning true if all are not equal to "b". Run the full Battleship program multiple times with different inputs to test all the different features.