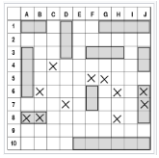


CS110 Project – Battleship

This project is presented to two phases to make it more manageable. After completing and testing Phase I, you will have the basis to move to Phase II. You will submit once – as much as you have completed of Phase II (or only Phase I if that's as far as you get). Zip all files, including text files for submission.

All submissions must be received by end of day Friday, May 6th. No late submissions will be accepted.

Background

Battleship is a classic two-person game, originating in pencil-paper form pre-dating World War I. More recently, released as a game consisting of plastic grids and pegs, and then an electronic version. Game play consists of a head to head battle between players. Each player has a 10x10 grid of locations and a fleet of 5 ships. Before the game begins, each player places their 5 ships on their board (hidden from the other player). Swapping turns, the players guess a location on their opponent's grid, in an attempt to locate a ship. When a ship has been hit in all locations it spans, the ship is sunk. The first player to sink their opponent's entire fleet, wins the game.

For our implementation, the user will play against the computer. The computer will make random (but not duplicate) guesses.

For Phase I, you are writing and testing some necessary classes. In Phase II you will complete the implementation of the game.

Phase I

Below, you'll find a UML of the Phase I required classes. I am giving you the code for the classes highlighted in yellow. You will write all other classes. The description of public methods is included in subsequent pages. You are encouraged to use additional private methods as needed, but the public interface of your classes must match the UML (and the descriptions provided).

Each player's fleet consists of 5 ships. [Different versions of this game have different ship names/sizes – be sure to use this configuration.]

Ship Name	Length
Destroyer	2
Cruiser	3
Submarine	3
Battleship	4
Aircraft Carrier	5

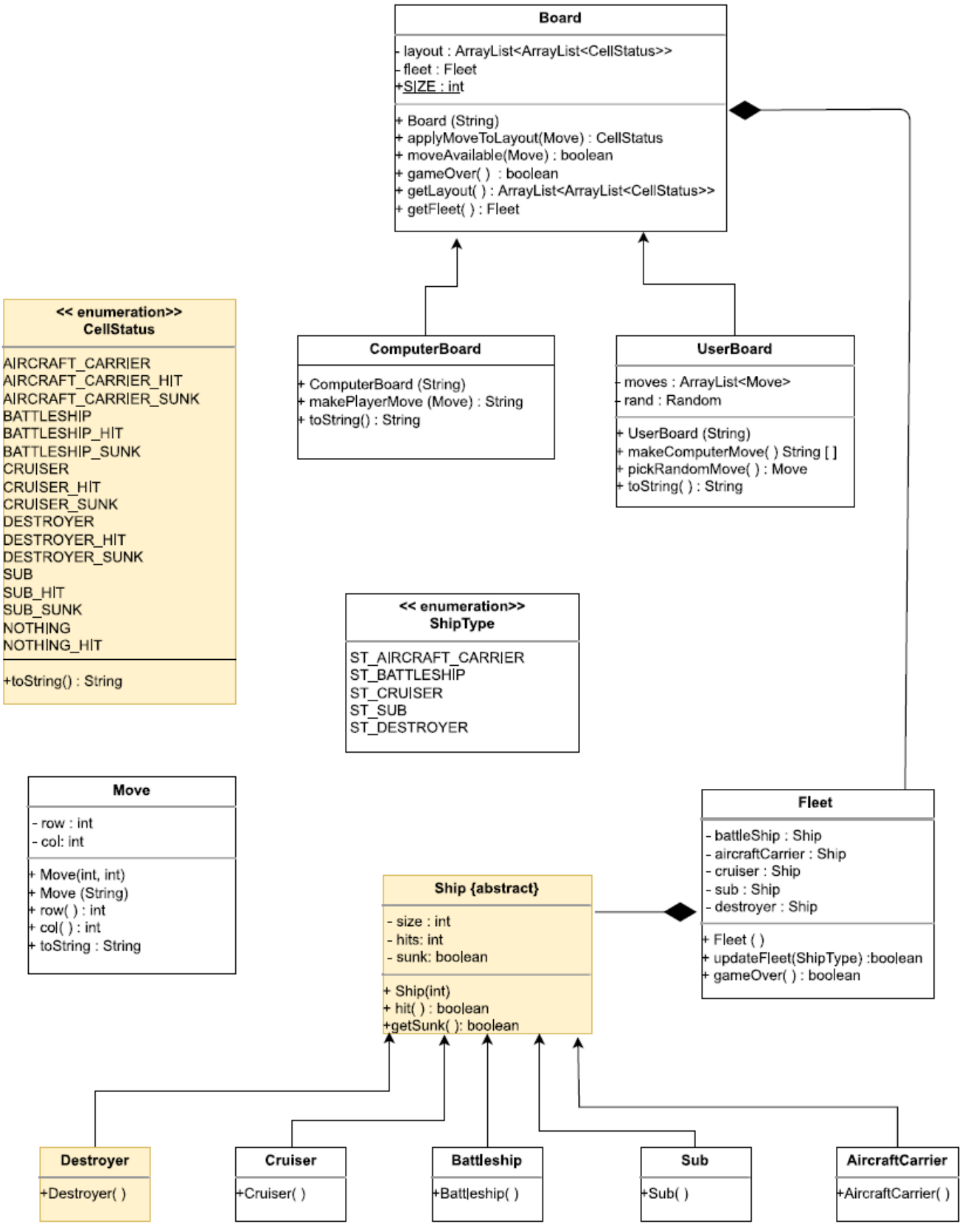
Both players will store their ship locations in a text file prior to the game start. Each line in the file will contain one letter (A, B, C, D, S) to indicate the ship type, followed by a start location and an end location. Here is a sample file:

```
C C1 C3
A A1 A5
B B1 B4
S D1 D3
D E1 E2
```

Note: You will need 2 files, one for the computer's board and one for the user's board. Order of lines in the file should not matter. If either file does not exist, issue and message and gracefully exit the program.

Posted with this assignment is a file named PhaseOneTester.java. Your Phase I classes should work with this tester. I would encourage you to comment out most of the tester and add in only the functionality you are currently testing. For example, only execute the "Test your Move class" segment after you've written the Move class. Only after you have ensured successful implement of this class, should you move on to the "Test your Fleet" code.

Battleship Phase I Classes



Enum ShipType

Enum Constants
ST_AIRCRAFT_CARRIER
ST_BATTLESHIP
ST_CRUISER
ST_DESTROYER
ST_SUB

Cruiser class “is-a” Ship

Calls Ship class constructor with size of 3.

Battleship class “is-a” Ship

Calls Ship class constructor with size of 4.

AircraftCarrier class “is-a” Ship

Calls Ship class constructor with size of 5.

Sub class “is-a” Ship

Calls Ship class constructor with size of 3.

Move

- **Move(int, int)**

  Creates a Move object from two integers representing the indices in a two-dimensional array.

- **Move(String)**

  Creates a move object from a String consisting of a letter and a number.

- **row**

  Accessor for row. Using 'row' rather than 'getRow' allows for more compact code when manipulating ArrayLists.

- **col**

  Accessor for col. Using 'col' rather than 'getCol' allows for more compact code when manipulating arrayLists.

- **toString**

  Returns a 2 to 3-character string consisting of a letter in the range A-J followed by a number in the range 1-10. Provides for ease of display of move values in an interface.

Fleet

- **Fleet**

  Initializes Ship fields.

- **updateFleet**

  Informs the appropriate ship that it has been hit, and returns true if this sank the ship, and false if it did not.

- **gameOver**

  Returns true if all ships have been sunk, returns false if not.

Board

- **Board**

  Initializes layout, initially setting all cells to CellStatus.NOTHING. Gets information from file and add ships to the layout. Initializes Fleet.

- **applyMoveToLayout**

  Applies a move to layout. If the targeted cell does not contain a ship, it is set to CellStatus.NOTHING_HIT. If it contains a ship, the cell is changed from, for instance, CellStatus.SUB to CellStatus.SUB_HIT. It returns the original CellStatus of the targeted cell.

- **moveAvailable**

  Takes a move as a parameter and determines if the spot is available

- **getLayout**

  Returns a reference to the layout (not a deep copy!)

- **getFleet**

  Returns a reference to the Fleet object

- **gameOver**

  Returns true if all ships have been sunk, false otherwise.

ComputerBoard "is-a" Board

- **ComputerBoard(String)**

  Passes the filename on to the Board constructor

- **makePlayerMove**

  Takes a move and makes it AGAINST this board. Takes in move to be applied. Returns either null, or, if the move sank a ship, a String along the lines of "You sank My Battleship!"

  if a ship was sunk, calls updateLayout to change _HIT values to _SUNK values.

- **toString**

  Returns a String representation of the ComputerBoard, displaying the first character of the String returned by the toString method overridden in CellStatus

UserBoard "is-a" Board

The UserBoard maintains a list of all possible moves. Initially, it will be all locations on the Board. When the computer takes a turn, it randomly selects an item from this list and removes it from the list.

- **UserBoard(String)**

  Passes the filename on to the Board constructor. Initialize the Random object and the ArrayList of all possible Moves.

- **makeComputerMove**

  Computer move against UserBoard. Selects and makes a move AGAINST this board. Returns an array of two Strings. The first is the move the computer made in user readable form. The second is either null, or, if the move resulted in a ship being sunk, a string along the lines of "You sunk my Battleship!"

- **toString**

  Returns a String representation of the ComputerBoard, displaying the second character of the String returned by the toString method overridden in CellStatus