**Battleship Project** - 200 Points

You are asked to build a Battleship program. This will be a one player game.

**Battleship Rules** - The goal of Battleship is to sink five different boats by guessing coordinates on a grid.

* Each row in the game is represented by the letters A through J. (10 rows).
* The columns are represented by the numbers 0 though 9. (10 columns).
* Thus, there are 100 locations on the board.
* The five boats have varying lengths to them. Each boat also has a symbol that defines it to the player.

|  |  |  |
| --- | --- | --- |
| **Boat** | **Length** | **Symbol** |
| Aircraft carrier | 5 | C |
| Battleship | 4 | B |
| Destroyer | 3 | D |
| Submarine | 3 | S |
| PT Boat | 2 | P |

* The board should be nicely formatted and outputted to the console. It should include the Row letters and Column numbers.
* The boats are randomly placed on the board and cannot overlap one another, nor can they extend outside the board itself.
* Once the ships are randomly placed, the user begins guessing coordinates on the board using the row letter and column number together. Ex: C7, E1, etc…
* The board will respond with either "Hit" or "Miss". That location on the board will be filled with either an "X" (hit) or "O" (miss)
* After each guess, the board should be updated and outputted to the console.
* If the hit sinks a ship, an appropriate message is outputted. "You sank the destroyer." And the ship's symbols are revealed on the board.
* Play continues until all ships are sunk, at which point the program will output that the player won and how many guesses it took them to do so.

**Program Design Requirements**

Required Classes

**Ship Class**

Data Fields: (all should be private)

***name*** - This String data field holds the name of the ship. (Aircraft carrier, sub, etc…)

***size*** - This integer data field holds the size of the ship.

***hits*** - This integer data field hold the number of hits the ship has taken.

***vertical*** - This boolean data field holds whether the ship is running vertically (true) or horizontally(false), which can be useful when placing the ships on the board.

**symbol** - This character data field controls the symbol used to determine the kind of ship.

**Constructor** - 2 constructors

Ship - no-arg constructor - Defaults to battleship

Ship (name, size, symbol)

**Ship Methods**

get and set methods for all data fields.

*public boolean hit()* – This method will increase the number of hits on a ship and check to see if the hit sinks the ship. An appropriate message should be outputted. (“You sank the battleship”). This method will return true if the ship is sunk, otherwise false. This method can be used to update the fleet ArrayList in the event a Ship is sunk.

**Coordinate Class**

**Data fields**

***row*** - This character data field holds the row letter for a given coordinate on the board.

***column*** - This character data field holds the column number for a given coordinate on the board.

***guessed*** - This boolean data field holds true or false based on whether the player has guessed this coordinate before.

***contents*** - This character data field holds the contents of the coordinate on the board. There are six possible characters:

|  |  |
| --- | --- |
| **Symbol** | **Situation** |
| ‘ ‘ (space) | This coordinate holds nothing |
| C | Holds part of Aircraft Carrier |
| B | Holds part of the Battleship |
| D | Holds part of the Destroyer |
| S | Holds part of the Submarine |
| P | Holds part of the PT Boat |

**Constructors**

Coordinate() - (no - arg) - Defaults to 0,0 (will be unused)

Coordinate(char r, char c) - Fills in the row and column character data fields. contents will be filled later when ships are randomly placed.

**Methods**

get and set methods for all data fields.

***public char display*** (ArrayList<Ship>) - This method receives the ArrayList of Ships currently on the board and will return the symbol to be outputted by the game. If the coordinate has not been guessed, a hyphen (-) should be returned. If the coordinate has been guessed, it should return "O" if the coordinate holds no ship, "X" if the coordinate holds a ship that hasn't been sunk, or the ship's symbol if the ship has been sunk.

**Board Class**

The Board class holds the 2D array of Coordinates as well as the locations of all the ships.

**Data Fields**

***board*** - This is a 2D array of Coordinates. It will use this to output the board and keep track of the game's state.

***fleet*** - This is an ArrayList of Ships. It will contain all five ships at the start of the game. When a ship is sunk, it is removed from the ArrayList.

**guesses** – This integer data field will hold the number of guesses it takes the player to find all five ships.

**Constructor**

Board() - (no-arg) - This is the only constructor for the game. This constructor is what triggers the creation of all Ships, Coordinates and populates the game board for the player.

**Methods**

***public void initializeBoard()***- This method will create all Coordinate objects and place them in the *board* 2D array.

***public void makeShips()*** - This method will create the five Ship objects and place them in the *fleet* ArrayList.

***public void placeShips()*** – This method will randomly place the five ships within the board 2D array. None of the five ships can overlap or extend off the board. \*You may end up creating other methods to help with the placement of the ships, such as ***isValid*** (boolean method to see if the random location works)

**public void place(Ship, row, col, vertical)** – This method will set the contents of the spots on the board that the ship will occupy.

***public void showBoard()*** – This method will output the board to the console. It should use the Coordinate ***display*** method to output the appropriate symbol within the board.

**public void check(char r, char c)** – This method will take the user’s guess and process it properly. Make sure it will handle the following situations:

* The String will have to be converted into integer values to check the location on the board.
* It will alert the user if this coordinate has already been guessed.
* It will alert the user of a hit on a ship and if a ship is sunk.
* It will alert the user of a miss.

**public void showShips()** – This will show the location of the ships on the board. (Cheat)

**public void showSolution** – This will simply show the contents of all Coordinate objects on the board.

**public void play()** – This will begin the gameplay phase. Here, you should create a while loop that will continue until the ***fleet*** ArrayList is empty. At that point the game is over and the number of guesses used by the player is displayed.

* Each iteration of the while loop should request a coordinate from the player. Be sure to check for invalid input and provide an appropriate message when this happens.
* Each guess should update the game board and redisplay the state of the game in the console.
* An input of ***??*** will reveal the solution of the board, without disrupting or changing play of the board.
* An input of **SS** will output the names of the ships remaining on the board.

**BattleshipRunner Class**

This is your runner class. The first and only line should be:

Board b = new Board();

 Sample Solution Same Board During Gameplay

User has partially sunk the battleship and aircraft carrier. Submarine is sunk. Player hasn't hit the destroyer or PT boat yet. Os are misses.

 0 1 2 3 4 5 6 7 8 9

A O - - O - - X - O -

B - - O - - - X - - -

C - O - - - - - - - O

D - - - - O - - X - -

E O - - - - - - - - -

F - - - O - - - - - O

G - - - - - S S S - -

H - - - - - - - - - -

I - - O - - - - - O -

J O - - - O - O - - -

 0 1 2 3 4 5 6 7 8 9

A - - - - - - B A - -

B - - - - - - B A - -

C - - - - - - B A - -

D - D D D - - B A - -

E - - - - - - - A - -

F - P - - - - - - - -

G - P - - - S S S - -

H - - - - - - - - - -

I - - - - - - - - - -

J - - - - - - - - - -

Name:

**Submitting Battleship**

1. Submit to repl.it your 4 Class files: Ship, Board, Coordinate, BattleshipRunner. Run your program and fill in the “Points Earned” Column of the rubric below.
2. Have a second student fill out the “Verification” column. They will sign their name at the bottom document.

**Battleship Rubric**

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective** | **Points Possible** | **Points Earned** | **Verification** |
| Does your program output a board representing a 10 x 10 Battleship board? | 20 |  |  |
| Does your output include row letters and column numbers, nicely formatted? | 10 |  |  |
| Does your program randomly place 5 ships without overlaps or any ships extending beyond the boundary of the Battleship board? | 50 |  |  |
| Does your program convert the user's input into a coordinate on the board? | 10 |  |  |
| Does your program handle invalid input from the user without crashing? | 10 |  |  |
| Does your program accurately determine misses based on the supplied coordinate? (“O” on board) | 10 |  |  |
| Does your program accurately determine hits based on the supplied coordinate? (“X” on board) | 10 |  |  |
| When a ship is sunk, does the board change all its Xs to the ships symbol? | 10 |  |  |
| Does your program accurately detect when a ship "sinks"? | 20 |  |  |
| Does your program detect when all ships are sunk? (Game over condition) | 20 |  |  |
| Does an input of "SS" show the names of the remaining ships on the board? | 10 |  |  |
| Does an input of "??" show the solution of the board, without disrupting game play? | 20 |  |  |
| **Total** | 200 |  |  |

Verified by: