CS 1331 Homework 2 - Battleship

Introduction

In this programming project you will practice:

- reading from the console
- reading from a file
- control structures
- using methods
- arrays (1D and 2D)

Problem Description

You are to code up a modified Battleship game (https://en.wikipedia.org/wiki/Battleship_(game)). This game of Battleship involves two players that are trying to sink each other's ships. Each player has a grid of their current hits, misses and unguessed squares. Players take turns in guessing locations on the opposing player's board on where they think a boat is. Depending on that guess and the locations of the ships, the attempt can be a hit or a miss. In our version of Battleship, each player has only one shot (guess) per turn and even if it's a successful hit, that player's turn ends. The game ends when a player has successfully hit all of their opponent's ships.

Solution Description

Download Battleship.java (/spring2018/hw2/Battleship.java). You will be modifying this file that has many stubbed methods that you will complete along with the main method. The board will have letters representing the columns and numbers representing the rows. So the coordinate 'a7' will mean the "a"th column and 7th row. Your code should treat a7 and A7 as the same spot. Rows will start from 1 not 0. Treat the top left corner as 'a1'.

You will be provided a .txt file that will provide the boards for each player. The first line will have the dimension of each player's board on the first line (e.g. 3 corresponds with a 3x3 board). The second line will be the positions of player 1's ships. The third line will be the positions of player 2 ships. The placements of the ships will be guaranteed to be valid, and there can be a varying number of ship locations (however player 1 and 2 will have the same number).

ex.

```
5                                    //board will be 5x5
b4 b3 a4 a6 a7 c3 c5 f3 //player 1's ship locations
c3 c4 f5 f3 a2 b3 d1 d2  //player 2' ship locations
```

The game should start by printing what player 1 sees (a grid of player 2's board) and how many hits they have remaining. '~' should represent unguessed squares, 'X' for hit squares and 'O' for missed squares. A Scanner should then record the user's guess through the command line. If the guess is a successful hit, print "Hit!" and change the board location from a '~' to a 'X'. If the guess is a miss, print "Miss!" and change the board location from '~' to a 'O'. Re-print the board after each guess. Repeat the process for player 2. The game should continue shifting turns until one player has hit all of the other's ships.

At the end of the game, print which player (1 or 2) was the winner.

Methods

IMPORTANT: you may not change the method headers

- initBoard(): will take in parameter **n** and create a 2D char array of size n x n. Returns the initialized char[][].
- printBoard(): this method prints the **board** array, treat the top left corner as [0][0].
- fireMissle(): this method will take in a player's **board**, the **target** string of the location being fired at, a **shipLocations** string array of where the ships were originally placed, and **hitsLeft** representing the number of hits the player firing needs to hit left. If the selected target location has already been hit, print to the command line "**target**(e.g. a4) has already been chosen!" and should still be counted as a turn. Returns how many hits left the firing player has left after calling this fireMissle() method.
- isShip(): **target** is the string coordinate of where on the board to look, **shipLocations** is a string array of where the ships were originally placed. Returns whether the location is a ship or not.
- convertLocation(): takes in a string **coordinate** e.g. f1 and converts this to an int array of the corresponding indices of the board e.g. {0, 5} (represented by f1). Note that the row must be the first element and the column the second
- main(): the main method is where you will be writing the code to play the game. Here is where you will ask the players to make a console input of a coordinate. The input should rotate between player 1 and player 2. You must use all of the methods above inside your main method.

Sample Output

```
Player 1 (7 hits left):
~ ~ ~ ~
~ ~ ~ ~
~ ~ ~ ~
~ ~ ~ ~
Player 1 enter missile location: b2
Hit!
Player 1 (6 hits left):
~ ~ ~ ~
~ X ~ ~
~ ~ ~ ~
~ ~ ~ ~

----------

Player 2 (7 hits left):
~ ~ ~ ~
~ ~ ~ ~
~ ~ ~ ~
~ ~ ~ ~
Player 2 enter missile location: c3
Miss!
Player 2 (7 hits left):
~ ~ ~ ~
~ ~ ~ ~
~ ~ O ~
~ ~ ~ ~

...

Player 2 (1 hits left):
X X X ~
~ ~ ~ X
~ X O O
~ X O ~
Player 2 enter missile location: a4
Hit!
Player 2 (0 hits left):
X X X X
~ ~ ~ X
~ X O O
~ X O ~

----------

The winner is Player 2
```

see ex1.txt (/spring2018/hw2/ex1.txt) for the full example

Solution Constraints

- **IMPORTANT**
- You must use a Scanner (https://docs.oracle.com/javase/9/docs/api/java/util/Scanner.html) reading from `System.in` to get the player's input and to read the given game files.
- User and file input can be assumed to always be valid
- The size of the board will never be more than 9x9
- You cannot create your own classes for this assignment (other than the Battleship class), or import any classes other than the ones provided in the template

Tips and Considerations

- the String split() method will be useful for splitting a String into a String[] using a delimiter
- the String toLowerCase() and toUpperCase() methods might be useful
- make sure you know the difference between comparing the equality of primitives and Objects
- **char** is an integral type. That means you can do arithmetic with **char**s like location..charAt(0) - 'a' to translate a file letter to an integer index.
- make sure that you know how to index correctly into a 2D array, taking care to ensure the letters correspond with the columns and numbers with the rows

Grading

- [5] initializes ship locations from file correctly
- [10] initBoard method
- [10] printBoard method
- [5] convertLocations method
- [10] isShip method
- [5] fireMissile doesn't redo previous hits/misses
- [10] fireMissile registers hit correctly
- [10] fireMissile registers miss correctly
- [10] hits left counts are properly updated

- [10] game ends when all ship locations are hit
- [5] prints the correct winner
- [10] switches players after every turn

Javadocs

You will need to write Javadoc comments and look for checkstyle errors with your submission.

- Every class should have a class level Javadoc that includes `@author <GT Username>`.
- Every public method should have a Javadoc explaining what the method does and includes any of the following tags if applicable:
    - `@param <parameter name> <brief description of parameter>`
    - `@returns <brief description of what is returned>`
    - `@throws <Exception> <brief explanation of when the given exception is thrown>`

See the CS 1331 Style Guide (http://cs1331.gatech.edu/cs1331-style-guide.html) for details.

Checkstyle

For each of your homework assignments we will run checkstyle and deduct one point for every checkstyle error.

For this homework the **checkstyle cap is 20**, meaning you can lose up to 20 points on this assignment due to style errors. This limit will increase with each homework.

- If you encounter trouble running checkstyle, check Piazza for a solution and/or ask a TA as soon as you can!
- You can run checkstyle on your code by using the jar file found on the course website that includes xml configuration file specifying our checks. To check the style of your code run `java -jar checkstyle-6.2.2.jar *.java`.
- To check your Javadocs run `java -jar checkstyle-6.2.2.jar -j *.java`.
- Note that the command for checking code and the command for checking Javadocs are different. You will have to run both commands to fully test for style errors.
- Javadoc errors are the same as checkstyle errors, as in each one is worth a single point and they are counted towards the checkstyle cap.
- **You will be responsible for running checkstyle on *ALL* of your code.**
- Depending on your editor, you might be able to change some settings to make it easier to write style-compliant code. See the customization tips (http://cs1331.gatech.edu/customization-tips.html) page for more information.

Collaboration

When completing homeworks for CS1331 you may talk with other students about:

_ What general strategies or algorithms you used to solve problems in the homeworks _ Parts of the homework specification you are unsure of and need more explanation _ Online resources that helped you find a solution _ Key course concepts and Java language features used in your solution _ **You may not discuss, show, or share by other means the specifics of your code, including screenshots, file sharing, or showing someone else the code on your computer, or use code shared by others.**

Examples of approved/disapproved collaboration:

OKAY: "Hey, I'm really confused on how we are supposed to implement this part of the homework. What strategies/resources did you use to solve it?"

BY NO MEANS OKAY: "Hey… the homework is due in like 20 minutes… Can I see your code? I *promise* won't copy it directly!"

In addition to the above rules, note that it is not allowed to upload your code to any sort of public repository. This could be considered an Honor Code violation, even if it is after the homework is due.

Submission

- Submit your `Battleship.java` file as an attachment to the `hw2` assignment on Canvas. You can submit as many times as you want, so feel free to submit as you make substantial progress on the homework. We only grade your **last** submission, meaning we will ignore any previous submissions.
- As always, late submissions will not be accepted and non-compiling code will be given a score of 0. For this reason, we recommend submitting early and then confirming that you submitted ALL of the necessary files by re-downloading your file(s) and compiling/running them.

Have fun!