Homework 03 - Battleship

Problem Description

Hello, and welcome! Please make sure to read all parts carefully.

For homework 03, you will be creating a recreation of the game Battleship, where two players will choose the locations of ships on a board and attempt to sink the other player's ships by choosing coordinates to fire at.

Solution Description

Your program must be named `Battleship.java`. Please read all the steps and look at the Example Output carefully before you begin. It must work as follows:

1. Print out the message `Welcome to Battleship!`
2. Prompt each user to enter coordinates for five ships of length one. There must be five separate prompts for each ship (use the example below as a guide). You can expect the user input will be two ints separated by a space. The first int represents the row number and the second int represents the column number.
    a. If the user enters invalid integers, print

    ```
    Invalid coordinates. Choose different coordinates.
    ```

    b. If the user enters a coordinate that they had already entered, print

    ```
    You already have a ship there. Choose different coordinates.
    ```

    c. After each player enters their fifth coordinate, a board representing the player's ship locations must be printed to the console using the provided method. See step three on how to construct these Locations Boards.
    d. 100 new lines must follow the printed board so that the other player will not see the entered coordinates and board of their opponent.
3. Create two 5x5 grids in the form of 2D arrays using the coordinates entered by the players. These Location Boards store each player's ship locations and will be used to keep track of the damage states of each player's ships, as well as any misses. The corresponding Location Board must be printed to the console right after a player enters the coordinates of their ships.
    a. A `'-'` character must represent an empty space.
    b. An `'@'` character must represent a ship that is not hit. When the game begins, all ships will start fresh with no hits.
    c. An `'X'` character will represent a space with a ship that has been hit.
    d. An `'O'` character will represent a space that was fired upon, but since there is not ship at that location, the shot was a miss.
    e. Each player's board must have five ships of length one. Five of the 25 grid spaces will start with ships on them.
4. Additionally, you must generate two more 5x5 grids in the form of 2D arrays. These Target History Boards will allow each player to visually track their hits and misses. After each hit or miss by the player, their Target History Board must be printed to the console using the provided method.
    a. On this board, an `'X'` character must represent a hit by the player, an `'O'` character must represent a miss by the player, and a `'-'` character must represent a space that has not been attacked.

5. Prompt Player 1 to enter a coordinate to fire upon. You can expect the user input will be two ints separated by a space.
    a. If the user enters invalid integers, print

        ```
        Invalid coordinates. Choose different coordinates.
        ```

    b. If the user enters a coordinate that they had already entered, print out the following

        ```
        You already fired on this spot. Choose different coordinates.
        ```

    c. If the user enters a coordinate with no ship on it, print out the following and print the updated Target History Board, where `[NUM]` is replaced with the attacked player's ID.

        ```
        PLAYER [NUM] MISSED!
        ```

    d. If the user enters a coordinate with a ship on it, print out the following and print the updated Target History Board, where `[NUM A]` is replaced with the attacking player's ID and `[NUM B]` is replaced with the attacked player's ID.

        ```
        PLAYER [NUM A] HIT PLAYER [NUM B]'s SHIP!
        ```

6. Player 2 will get a turn after each turn that Player 1 takes, which will function in the same way as Player 1's turns.
7. When a ship is hit by a player, the Location board (which tracks the damage states) of the corresponding player's ships must be updated. Misses should be updated on the Location board as well.
8. The program must terminate gracefully after a player wins. This will occur when all of the `'@'` signs on their opponent's board have been replaced with `'X'` symbols.
    a. Immediately following the move which sinks the final ship location, print the following message, where `[NUM]` is replaced by the winning player's ID.:

        ```
        PLAYER [NUM] WINS! YOU SUNK ALL OF YOUR OPPONENT'S SHIPS!
        ```

    b. Using the provided method, print both players' Location Boards in order to verify the results of the game to the players. Player 1's Location Board should be printed first.

In your solution, you must use each of the following Java features at least once:

1. A for loop (not including those used in provided code)
2. A do-while loop.

Note: A premade Battleship.java file will be provided for this HW. Download the file from Canvas. Your code MUST be written in this file. The file simply includes a method, `printBattleShip(…)`, that MUST be used for printing 2D arrays to the console. Make sure not to alter the `printBattleShip(…)` method.

HINT: The method is used by passing in the 2D array you wish to print. For example:

```
printBattleShip(playerOneShotsBoard);
```

Example Outputs

User input is **bolded**. Please make sure to follow the exact formatting as shown in the pdf.

```
Welcome to Battleship!

PLAYER 1, ENTER YOUR SHIPS' COORDINATES.
Enter ship 1 location:
```

0 1
Enter ship 2 location:
1 3
Enter ship 3 location:
2 1
Enter ship 4 location:
3 0
Enter ship 5 location:
3 4
```
  0 1 2 3 4
0 - @ - - -
1 - - - @ -
2 - @ - - -
3 @ - - - @
4 - - - - -
```

PLAYER 2, ENTER YOUR SHIPS' COORDINATES.
Enter ship 1 location:
0 1
Enter ship 2 location:
0 4
Enter ship 3 location:
2 0
Enter ship 4 location:
5 10
Invalid coordinates. Choose different coordinates.
Enter ship 4 location:
3 1
Enter ship 5 location:
4 4
```
  0 1 2 3 4
0 - @ - - @
1 - - - - -
2 @ - - - -
3 - @ - - -
4 - - - - @
```

Player 1, enter hit row/column:
5 12
Invalid coordinates. Choose different coordinates.
Player 1, enter hit row/column:
3 2
PLAYER 1 MISSED!
```
  0 1 2 3 4
0 - - - - -
1 - - - - -
2 - - - - -
3 - - O - -
4 - - - - -
```

Player 2, enter hit row/column:
1 0
PLAYER 2 MISSED!
```
  0 1 2 3 4
0 - - - - -
1 O - - - -
2 - - - - -
```

```
3 - - - - -
4 - - - - -

Player 1, enter hit row/column:
3 2
You already fired on this spot. Choose different coordinates.
Player 1, enter hit row/column:
0 4
PLAYER 1 HIT PLAYER 2's SHIP!
  0 1 2 3 4
0 - - - - X
1 - - - - -
2 - - - - -
3 - - O - -
4 - - - - -

Player 2, enter hit row/column:
3 3
PLAYER 2 MISSED!
  0 1 2 3 4
0 - - - - -
1 O - - - -
2 - - - - -
3 - - - O -
4 - - - - -

Player 1, enter hit row/column:
2 0
PLAYER 1 HIT PLAYER 2's SHIP!
  0 1 2 3 4
0 - - - - X
1 - - - - -
2 X - - - -
3 - - O - -
4 - - - - -

Player 2, enter hit row/column:
3 4
PLAYER 2 HIT PLAYER 1's SHIP!
  0 1 2 3 4
0 - - - - -
1 O - - - -
2 - - - - -
3 - - - O X
4 - - - - -

Player 1, enter hit row/column:
4 4
PLAYER 1 HIT PLAYER 2's SHIP!
  0 1 2 3 4
0 - - - - X
1 - - - - -
2 X - - - -
3 - - O - -
4 - - - - X

Player 2, enter hit row/column:
```

```
0 2
PLAYER 2 MISSED!
  0 1 2 3 4
0 - - O - -
1 O - - - -
2 - - - - -
3 - - - O X
4 - - - - -
```
****Skipping to the last turn****

```
Player 1, enter hit row/column:
3 1
PLAYER 1 HIT PLAYER 2's SHIP!
  0 1 2 3 4
0 - X - - X
1 - - - - -
2 X - - - -
3 - X O - -
4 - - - - X
PLAYER 1 WINS! YOU SUNK ALL OF YOUR OPPONENT'S SHIPS!

Final boards:

  0 1 2 3 4
0 - @ O O -
1 O - - @ -
2 - @ - - -
3 @ - - O X
4 - O - - -

  0 1 2 3 4
0 - X - - X
1 - - - - -
2 X - - - -
3 - X O - -
4 - - - - X
```

Rubric

[100] **Battleship.java**

- [40] Correctly generates boards
    - [10] Prompts user for ship coordinates and correctly takes in inputs
    - [20] Creates and prints a correct Location Board for Players 1 and 2
    - [10] Creates a Target History Board for tracking shots for each player
- [10] Correctly prompts users to type in target coordinates
- [10] Uses a for loop and a do-while loop.
- [10] Correct game flow
- [10] Correctly tracks hits and misses and prints them
- [20] Correctly ends the game when a player wins
    - [10] All ships of one player were sunk
    - [10] The correct player is declared the winner

The Checkstyle cap for this homework assignment is 15. Up to 15 points can be lost from Checkstyle errors.

We reserve the right to adjust the rubric, but this is typically only done for correcting mistakes.

Allowed Imports

To prevent trivialization of the assignment, you may only import `java.util.Scanner`.

Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or break our auto grader. For that reason, do not use any of the following in your final submission:

- `var` (the reserved keyword)
- `System.exit`

Collaboration

Collaboration Statement

To ensure that you acknowledge a collaboration and give credit where credit is due, **we require that you place a collaboration statement as a comment at the top of at least one .java file that you submit**. That collaboration statement should say either:

I worked on the homework assignment alone, using only course materials.

or

In order to help learn course concepts, I worked on the homework with [give the names of the people you worked with], discussed homework topics and issues with [provide names of people], and/or consulted related material that can be found at [cite any other materials not provided as course materials for CS 1331 that assisted your learning].

Allowed 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 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:

- **approved**: "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?"
- **disapproved**: "Hey, it's 10:40 on Thursday... Can I see your code? I won't copy it directly I promise"

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.

Turn-In Procedure

Submission

To submit, upload the files listed below to the corresponding assignment on Gradescope:

- Battleship.java

Make sure you see the message stating "HW03 submitted successfully". From this point, Gradescope will run a basic autograder on your submission as discussed in the next section.

You can submit as many times as you want before the deadline, so feel free to resubmit as you make substantial progress on the homework. We will only grade your last submission: be sure to **submit every file each time you resubmit**.

Gradescope Autograder

For each submission, you will be able to see the results of a few basic test cases on your code. Each test typically corresponds to a rubric item, and the score returned represents the performance of your code on those rubric items only. If you fail a test, you can look at the output to determine what went wrong and resubmit once you have fixed the issue.

The Gradescope tests serve two main purposes:

- Prevent upload mistakes (e.g. non-compiling code)
- Provide basic formatting and usage validation

In other words, the test cases on Gradescope are by no means comprehensive. Be sure to thoroughly test your code by considering edge cases and writing your own test files. You also should avoid using Gradescope to compile, run, or Checkstyle your code; you can do that locally on your machine.

Other portions of your assignment can also be graded by a TA once the submission deadline has passed, so the output on Gradescope may not necessarily reflect your grade for the assignment.

Important Notes (Don't Skip)

- Non-compiling files will receive a 0 for all associated rubric items
- Do not submit `.class` files
- Test your code in addition to the basic checks on Gradescope
- Run Checkstyle on your code to avoid losing points
- Submit every file each time you resubmit
- Read the "Allowed Imports" and "Restricted Features" to avoid losing points
- Check on Piazza for a note containing all official clarifications