HW03

Bookmark this page

Homework 3 due May 26, 2024 17:07 -03 Past due

Important:

This version of HW3 uses a **new autograder**, which aims to provide more detailed error messages than the legacy autograder (located in the next module section). Your score may be different than the one from the previous autograder: it could be higher or lower. We **recommend using the new autograder** if you have not yet submitted HW3 or are having problems with the legacy autograder.

Grades from the previous autograder are valid and will continue to be valid throughout the course. Your HW3 grade will be computed as the highest between the new and legacy autograders, so **if you encounter issues with this new autograder**, you can rely on your grade from the previous version still being valid, and **you can still submit to the legacy autograder**.

Important if you already worked on HW3 based on the prompt for the Legacy Autograder:

On the legacy version of the homework, several output lines use the curly (or "punctuation") apostrophe: ’. In this new version of the homework, we will use the straight (or "typewriter") apostrophe: '. This should be easier to type directly from a keyboard.

Other than this change, the homework prompt remains the same as the one in the legacy version.

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.
   - If the user enters invalid integers, print
     `Invalid coordinates. Choose different coordinates.`
   - If the user enters a coordinate that they had already entered, print
     `You already have a ship there. Choose different coordinates.`
   - 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.
   - 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 5×5 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 '–' character must represent an empty space.
   - An '@' character must represent a ship that is not hit. When the game begins, all ships will start fresh with no hits.
   - An 'X' character will represent a space with a ship that has been hit.
   - 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.
   - 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 5×5 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.
   - 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.
   - If the user enters invalid integers, print
     `Invalid coordinates. Choose different coordinates.`
   - If the user enters a coordinate that they had already entered, print out the following
     `You already fired on this spot. Choose different coordinates.`
   - 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!`
   - 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.
   - 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!`
   - 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. 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 below.

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

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

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`

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.

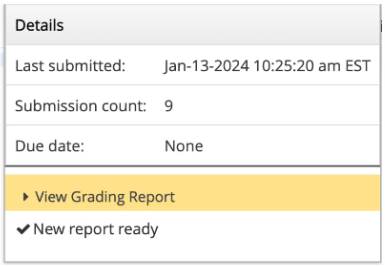
The Vocareum (code editor) interface has six main components:

- The **Files Navigation** is in the top left. This lets you choose from multiple available files, in the "work" folder.
- The **Build / Run** button. For all assignments in this course, the build and run button will perform the same action: compile your code and run a file scan. Building and running your code will not count towards your total allowed submission attempts, therefore you are free to build / run as many times as needed.
- The **Submit** button. This will compile your code, grade your assignment, and produce a grading report. **We expect that you build or run your code on your local computer/JDK before submitting to ensure that there are no issues that will prevent your code from being graded and that every submission attempt will generate meaningful results**. Your local JDK's errors, in some cases, will be even more descriptive than what Vocareum can offer you.
- The **Reset** button. This will revert all your changes and reset your code to the default code template.
- The **Code Window**. This is where you will write your code. Again, We highly recommend copying the starter code and working in your preferred IDE.
- The **Output Window**. This window will appear whenever you build, run, or submit your code and will display the results for you to view.

For additional help, please visit the Vocareum information page located in the course information module!

Submitting

You will submit through Vocareum, which will be loaded in a new window. Edit the .java files, and then submit once you are ready to have your homework autograded. Once it finishes autograding (this may take several minutes - be patient), you can access your report in the details tab. You can resubmit as many times as you want.



It is possible that due to how strict the autograder is with output formatting, several tests fail due to incorrect output. It is **OK** if your submissions initially fail on most, or even all, tests. **Review the expected and actual outputs to understand the difference, and adjust your code accordingly**.

The autograder reports output differences in two formats: **raw** (as the Java strings the autograder compares) and **visualized** (the output printed, which can sometimes look identical if the only differences are in whitespace). **We recommend reviewing the raw output if the visualized output is too long or when the visualized output looks the same for the expected and actual outputs**.

If you're having trouble finding the difference, you can use a **diff viewer**, such as the one in https://www.diffchecker.com/ (you can paste the expected in "original" and actual in "changed", or the other way around, as you prefer).

For learners unable to access the Vocareum environment, this is the provided Battleship.java file:

Battleship.java

< Previous | Next >