

Homework 4

Due Saturday November 17th at 11:59 pm
100 Points

For this assignment we will continue to work on our ConnectX game, but we will not be adding any new functionality to our program. We will be creating automated test cases for our IGameBoard interface and implementations using JUnit. You do not need to create test cases for the Connect4Game class.

Requirements

You must test the following methods in a JUnit class called TestIGameBoard. Each test case should have it's own JUnit test function.

Test the following methods:

Constructor

- Create 3 distinct test cases for the constructor

CheckIfFree

- Create 3 distinct test cases forCheckIfFree

CheckHorizWin

- Create 5 distinct test cases

CheckVertWin

- Create 5 distinct test cases

CheckDiagWin

- Create 8 distinct test cases
- Note, the different diagonals are distinct

CheckTie

- Create 4 distinct test cases

WhatsAtPos

- Create 7 distinct test cases

PlaceToken

- Create 5 distinct test cases

Your JUnit test code must also use the Factory Method Design Pattern by having a private method that calls the constructor for your IGameBoard object and returns it. By always using the Factory Method, you will be able to easily switch between your 2 implementations while running your JUnit test cases.

You will also need an additional private helper method that will take in a 2D array of characters and return a string representation of that array. This string representation should exactly match what the toString method returns from our GameBoard classes. This will allow you to create an "expected" version of the GameBoard in a 2D array, then get the string version to compare to the actual output of your GameBoard. So your process for comparing your expected GameBoard and your actual GameBoard will look like this:

1. Create an empty "expected" 2D array
2. Call the factory method to create your empty IGameBoard object "gb"

3. Foreach token you need to place for your test case
 - a. Use placeToken to place it on your IGameBoard object "gb"
 - b. Place that same character in the same location in your 2D array "expected"
4. Call your helper method to create the expected string version of your 2D array
5. Call assertEquals to ensure your expected string and gb.toString() are equivalent

The Program Report

Along with your code you must submit a well formatted report with the following parts:

Requirements Analysis

Fully analyze the requirements of this program. Express all functional requirements as user stories. Remember to list all non-functional requirements of the program as well. Creating test cases is not a non-functional requirement, so these should be the same as previous assignments.

Design

Create a UML class diagram for each class and interface in the program. Also create UML Activity diagrams for every method in your Connect4Game class and every method in your GameBoard or GameBoardMem class (except the constructor). If a method is defined as a default method in the interface, you need to update it to remove any mention of the 2D array or other private data of the GameBoard class, and instead refer to primary methods. If a method is provided as a default method in the interface you only need to provide an Activity Diagram for the default method, you do not need to include the Activity for each implementation as well.

You do not need to create diagrams for your Test class that you will make for this assignment, so these diagrams are the same from the previous assignment.

Testing

In your report include a description for each test case which includes your input values, your expected output, and a reason for why you chose this test case and what makes it distinct. This can be a bulleted list or a table.

Deployment

Provide instructions about how your program can be compiled and run. Since you are required to submit a makefile with your code, this should be pretty simple. Your makefile should include targets to compile your program (default), run your program (run) compile your test cases (test) and run your test cases (make test)

Additional Requirements and Tips

- You must include a makefile with your program. We should be able to run and test your program by unzipping your directory and using the make, make run, make test, make runtest commands.
- If you discover any failures with your test cases, you will need to find the underlying fault and correct it.
- Your UML Diagrams must be made electronically. I recommend the freely available program draw.io, but if you have another diagramming tool your prefer feel free to use that.
- Hard coded values are acceptable (and necessary) for our input and expected output values
- You will need to call several methods in order to set up your test case. That is fine, as long as your Assert statements are only testing one method. Remember if you run your test cases and

you have failed test cases for whatsAtPos and checkDiagWin, you want to fix whatsAtPos first, as that method is called in checkDiagWin.

- You do not need to follow the entire process in your main function to test your IGameBoard class, because our input is hard coded and always follows our preconditions. This means
 - o You aren't asking the user for input
 - o You aren't validating input
 - o You don't need to call checkIfFree before placing tokens since they are hard coded and we know the column is free
 - o You don't need to call checkForWin after every token placement
 - o You aren't printing anything to the screen
 - o You don't need to follow the turn order when placing tokens.
- Remember test cases should be distinct in a meaningful way. The difference between checking for a win with the exact same board, but swapping X's and O's is not distinct, as IGameBoard doesn't actually care what the character is.
- You do not need to use any specific method (such as path testing) to identify test cases.
- Remember to think about what was challenging for you when you were writing code. What mistakes did you make in earlier attempts? Those would be good scenarios to test for.
- Make sure you are using JUnit 4, not the JUnit 3 shortcut on the school unix machines.
- The JUnit code for the test cases will be very similar for each method. Make sure you get one example working, then copy and paste to change the input values and expected output values.
- Remember to follow the best practices we discussed for JUnit test cases. They are different than the best practices for our normal production code.
- You may need to use several assert statements to check to see if everything worked correctly by checking the string representation, the return value of the method, the getter methods, and so on.
- While writing JUnit code is not necessarily difficult, it is time consuming. Thinking of all the test cases and fixing any faults you find will also take some time. Avoid the temptation to put this assignment off because its "just" writing test cases.
- You should not create test cases that violate the preconditions for the method being tested.
-

Submission

You will submit your program on Handin. You should have one zipped folder to submit. Inside of that folder you'll have your package directory (which will contain your code), your report (a pdf file) and your make file. Make sure your code is your java files and not your .class files.

Note: This assignment is due on a Saturday night. Friday night would be the same time as a lab assignment is due. There is an exam on the following Monday, so I don't want you working on the assignment late Sunday night. Please start this assignment early to give you the ability to balance this work load.

Academic Dishonesty

This is an INDIVIDUAL assignment. You should not collaborate with others on this assignment. See the syllabus for further details on academic dishonesty.

Late Policy

Your assignment is due at 11:59 pm. Even a minute after that deadline will be considered late. Make sure you are prepared to submit earlier in the day so you are prepared to handle any last second issues with submission.

If your assignment is late, but turned in within 24 hours of the due date, you will automatically receive 25 points off. If it is more than 24 hours of the due date, but less than 48 hours of the due date, then you will receive 50 points off automatically. After 48 hours from the due date late assignments will no longer be accepted.