Homework 1

Due Sunday September 23rd at 11:59 pm
100 Points

For this Homework we will be building a Connect 4 Game that we will run with a command line interface. We will continue to build off of this program throughout the semester, so it will be important to make good design decisions when creating your program.

The rules of Connect 4 are simple. The game board is an upright grid with 6 rows and 7 columns. 2 players play (X and O) and take turns dropping their tokens into the grid, so their token falls down that column to the lowest row that is not currently occupied. The players take turns dropping their tokens in until one player gets 4 tokens in a row either horizontally, diagonally, or vertically. The player who gets 4 tokens in a row wins. Once a column has 6 tokens in it, no more tokens can be added to that column. If all columns on the board are filled without any player getting 4 tokens in a row, then the game ends in a tie.

For our program, the game will always start with player X, and alternate between players X and O (who will play at the same computer). During each turn, the game will show the players the current board, ask the appropriate player to select a column to add their token to and then check to see if that results in a tie game or a win. If it results in a win, it will inform the players who won, show them the winning board, and ask if they want to play again. If it results in a tie, it will inform the players, show the game board, and ask if they want to play again. If it is neither a tie or a win, then the program will move onto the next players turn. If a player selects a column that is already full or a column that does not exist (column 15) then the game will inform the player of the error and ask them to choose again. See the sample input and output for more detail.

The Classes

GameBoard.java:

This class will represent the game board itself. All attributes of the class must be private unless they are static AND final. The GameBoard itself must be represented as a 2-Dimensional array of chars where position [0][0] would be the bottom left of the board and [6][6] would be the top right of the board. This class cannot interact with the users by asking for input or printing output to the screen.

The following methods must be publicly available. They must have the exact names, parameters, and return types listed:

boolean checkIfFree(int c): returns true if column c is not full, false if it is.

boolean checkForWin(int c): returns true if the last token placed (which was placed in column c) resulted in the player getting 4 in a row horizontally, diagonally, or vertically. Otherwise, it returns false.

Void placeToken(char p, int c): places a token p in column c on the game board. The token will be placed in the lowest available row in column c. Cannot be called if column c is full.

boolean checkHorizWin(int r, int c, char p): returns true if the last token placed (which was placed in row r, column c and player p) resulted in the player getting 4 in a row horizontally. Otherwise, it returns false.

boolean checkVertWin(int r, int c, char p): returns true if the last token placed (which was placed in row r, column c and player p) resulted in the player getting 4 in a row vertically. Otherwise, it returns false.

boolean checkDiagWin(int r, int c, char p): returns true if the last token placed (which was placed in column c) resulted in the player getting 4 in a row diagonally. Otherwise, it returns false.

char whatsAtPos(int r, int c): returns the char that is in row r and column c of the game board. If there is no token at the spot it should return a blank space character " ".

String toString(): Note: this function should be overriding the method inherited from the Object class. Returns a fully formatted (see example output) string that displays the current game board.

boolean checkTie(): returns true if the game board results in a tie game, otherwise it returns false.

Additionally, this class must have a default constructor (no parameters). No other methods should be publicly available.

Connect4Game.java:

Your program will have a class called Connect4Game.java that will contain your main function. This class will handle all of the interactions with the users, and utilize the GameBoard.java class to play the game. This is the only class that can get input from the user or print to the screen. This class will contain much of the logic of the game play.

You must have a main function for this class. You are allowed, but not required, to have additional functions if you would like to modularize your code. No class level or global variables may be used for this class.

Contracts and Comments

All methods (except for the main) must have preconditions and post-conditions specified in the Javadoc contracts. All methods (except for main) must also have the params and returns specified in the Javadoc comments as well. Any invariants for the GameBoard class should be specified at the top of the class file in Javadoc comments.

You code must be well commented. The Javadoc comments will only be enough for very simple methods. Remember, your comments will help the grader understand what you are trying to do in your code. If they don't understand what you are trying to do, you will lose points.

Sample input and output

```
|0|1|2|3|4|5|6|
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
```

```
| | | | | | | |
| | | | | | | |

Player X, what column do you want to place your marker in?
3
|0|1|2|3|4|5|6|
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | |X| | | |

Player O, what column do you want to place your marker in?
3
|0|1|2|3|4|5|6|
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | |O| | | |
| | | |X| | | |

Player X, what column do you want to place your marker in?
3
|0|1|2|3|4|5|6|
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | |X| | | |
| | | |O| | | |
| | | |X| | | |

Player O, what column do you want to place your marker in?
3
|0|1|2|3|4|5|6|
| | | | | | | |
| | | | | | | |
| | | |O| | | |
| | | |X| | | |
| | | |O| | | |
| | | |X| | | |

Player X, what column do you want to place your marker in?
3
|0|1|2|3|4|5|6|
| | | | | | | |
| | | |X| | | |
```

```
| | | |O| | | |
| | | |X| | | |
| | | |O| | | |
| | | |X| | | |

Player O, what column do you want to place your marker in?
3
|0|1|2|3|4|5|6|
| | | |O| | | |
| | | |X| | | |
| | | |O| | | |
| | | |X| | | |
| | | |O| | | |
| | | |X| | | |

Player X, what column do you want to place your marker in?
3
Column is full
Player X, what column do you want to place your marker in?
4
|0|1|2|3|4|5|6|
| | | |O| | | |
| | | |X| | | |
| | | |O| | | |
| | | |X| | | |
| | | |O| | | |
| | | |X|X| | |

Player O, what column do you want to place your marker in?
5
|0|1|2|3|4|5|6|
| | | |O| | | |
| | | |X| | | |
| | | |O| | | |
| | | |X| | | |
| | | |O| | | |
| | | |X|X|O| |

Player X, what column do you want to place your marker in?
2
|0|1|2|3|4|5|6|
| | | |O| | | |
| | | |X| | | |
| | | |O| | | |
| | | |X| | | |
| | | |O| | | |
| | |X|X|X|O| |
```

```
Player O, what column do you want to place your marker in?
-1
Column cannot be less that 0
Player O, what column do you want to place your marker in?
8
Column cannot be greater than 6
Player O, what column do you want to place your marker in?
0
|0|1|2|3|4|5|6|
| | | |O| | | |
| | | |X| | | |
| | | |O| | | |
| | | |X| | | |
| | | |O| | | |
|O| |X|X|X|O| |

Player X, what column do you want to place your marker in?
1
|0|1|2|3|4|5|6|
| | | |O| | | |
| | | |X| | | |
| | | |O| | | |
| | | |X| | | |
| | | |O| | | |
|O|X|X|X|X|O| |

Player X Won!
Would you like to play again? Y/N
m
Would you like to play again? Y/N
y
|0|1|2|3|4|5|6|
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |

Player X, what column do you want to place your marker in?
1
|0|1|2|3|4|5|6|
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| |X| | | | | |
```

```
Player O, what column do you want to place your marker in?
2
|0|1|2|3|4|5|6|
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| |X|O| | | | |

Player X, what column do you want to place your marker in?
1
|0|1|2|3|4|5|6|
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| |X| | | | | |
| |X|O| | | | |

Player O, what column do you want to place your marker in?
2
|0|1|2|3|4|5|6|
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| |X|O| | | | |
| |X|O| | | | |

Player X, what column do you want to place your marker in?
1
|0|1|2|3|4|5|6|
| | | | | | | |
| | | | | | | |
| | | | | | | |
| |X| | | | | |
| |X|O| | | | |
| |X|O| | | | |

Player O, what column do you want to place your marker in?
2
|0|1|2|3|4|5|6|
| | | | | | | |
| | | | | | | |
| | | | | | | |
| |X|O| | | | |
```

```
| |X|O| | | | |
| |X|O| | | | |

Player X, what column do you want to place your marker in?
1
|0|1|2|3|4|5|6|
| | | | | | | |
| | | | | | | |
| |X| | | | | |
| |X|O| | | | |
| |X|O| | | | |
| |X|O| | | | |

Player X Won!
Would you like to play again? Y/N
n

Process finished with exit code 0
```

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.

Design

Create a UML class diagram for each class in the program. Also create UML Activity diagrams for every method in your Connect4Game class and every method in your GameBoard class (except the constructor).

Testing

Describe how you tested your program to ensure it was functioning correctly. We have not covered testing in detail this semester, so the requirements at this point are pretty relaxed. Just describe your testing process. Include what specific inputs you used, and how you expected the program to react to those inputs. Make sure to test your program thoroughly.

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.

Additional Requirements

- You must include a makefile with your program. We should be able to run your program by unzipping your directory and using the make and make run commands.
- Remember, this class is about more than just functioning code. Your design and quality of code is very important. Remember the best practices we are discussing in class. Your code should be easy to change and maintain. You should not be using magic numbers. You should be considering Separation of Concerns, Information Hiding, and Encapsulation when designing your code. You should also be following the idea and rules of design by contract.
- A new game should always start with player X

- You class files should be in the cpsc2150.connectX package
- Your code should be well formatted and consistently formatted. This includes things like good variable names and consistent indenting style.
- You should not have any dead code in your program. Dead code is commented out code.
- Your code must be able to run on the school Unix machines. Usually there is no issue with moving code from intelliJ to Unix, but some times there can be issues, especially if you try to import any uncommon java libraries.
- Code that does not compile will receive a 0.
- Your input and output should match the example input and output provided. The TAs will prepare scripts to help with the grading process, and their scripts will rely on the input and output matching my example.
- 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.
- While you need to validate all input from the user, you can assume that they are entering numbers or characters when appropriate. You do not need to check to see that they entered 6 instead of "six."

Tips and Reminders

- Do your design work before you write any code. It will help you work through the logic and help you avoid writing code just to delete it later.
- Carefully consider how you can use your available methods to solve problems both in your main function, and in methods in the GameBoard class.
- When writing your code, try to consider how things may change in the future. How can you design your code now to be prepared for future changes and requirements?

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.

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.