Project 1

CPSC 2150

Part 1 Due: Wednesday, January 27th at 10:00 pm

Part 2 Due: Sunday, February 7th at 10:00 pm

100 pts

Introduction:

In this assignment, we will be designing an extended version of 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. You must follow all best practices we have discussed in class.

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. However, we will be creating an extended version of this game for a 6 x 9 board, and requiring 5 in a row either horizontally, vertically, or diagonally to win.

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 nor 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.

For the first stage of this project we will just be designing our game. You will be creating UML diagrams for classes and methods, as well as writing contracts for each method and class. Designing your system first will allow you to work through the complicated decisions about how your classes and methods interact without having to implement them yet. This will make it easier to implement individual methods later.

In order to do this you will need several classes and functions. Please note it is important to follow these directions exactly, to design the classes and functions as they are described, and to use the same class and function names, as well as the same function signatures (return type, and parameter list). This will be important for future assignments as well. Assignments that achieve the overall goal, but do not follow these instructions will lose a substantial amount of points.

`GameScreen.java`

This will be the class that contains the `main()` function and controls the flow of the game itself. It will alternate the game between players, say whose turn it is, get the column they would like, and place their marker. If the player chooses a location that is not available (either because the column is full or because it is outside the bounds of the 6 x 9 board) then the program will print a message

saying the space is unavailable and ask them to enter a new column. Once a placement has been made, it will check to see if either player has won. If so it will print off a congratulatory message, and ask if they would like to play again and prompt them for a response of "Y" or "N." If they choose to play again, it should present them with a blank board and start the game over. If a player has not won, it will check to see if the game has ended in a tie (no spaces left on the board). If the game has ended in a tie, it will print a message saying so, and ask if the player would like to play again. If the game has not ended in a win or a tie, then print the current board to the screen and prompt the next player for their next move.

When asking a player for their move, you can assume the user will enter an integer value for the `Column`. You cannot assume that they will enter a value in range (they could enter 15 or 21). Player `X` should go first in every game.

This class has the least amount of specific requirements. You may design your class and add as many private helper functions that you believe may help you. All input and output to the screen must happen in only this class.

For this stage of the assignment, you will need to include in your project report:

- A UML class diagram for `GameScreen`
- A UML activity diagram for each method in `GameScreen`

Additionally you will submit a code file for `GameScreen` called `GameScreen.java`. This code file will just be an outline for the eventual code file. It should include the signature, Javadoc comments, and contracts for all of the methods, but you do not need to write the code for each method yet. Because of this, your code will not compile yet.

`BoardPosition.java`

This class will be used to keep track of an individual cell for a board. `BoardPosition` will have variables to represent the `Row` position, and the `Column` position.

There will only be one constructor for the class, which will take in an `int` for the `Row` position, and an `int` for the `Column` position. After the constructor has been called, there will be no other way to change any of the fields through any setter functions.

Other functions that will be necessary:

```
public int getRow(){
      //returns the row
}

public int getColumn(){
      //returns the column
}
```

You must also override the `equals()` method inherited from the `Object` class. Two `BoardPosition`s are equal if they have the same row and column. You should override the `toString()` method to create a string in the following format "<row>,<column>". Example "3,5". You won't call these in your code, but it is a best practice to override them and provide a meaningful implementation.

No other methods are necessary, and no other methods should be provided.

For this stage of the assignment, you will need to include in your project report:

- A UML class diagram for `BoardPosition`

Additionally, you will submit a code file for `BoardPosition` called `BoardPosition.java`. This code file will just be an outline for the eventual code file. It should include the signature, Javadoc comments, and contracts for all of the methods, but you do not need to write the code for each method yet. Because of this, your code will not compile yet. You do not need to create activity diagrams for the methods in `BoardPosition`.

It may seem unnecessary to include this class. However, when designing our system we need to ask if pieces of data are strongly related to each other. The row and column numbers are strongly related to each other, they represent a position on the board, so we choose to encapsulate them later. We do this as a way to help anticipate the potential for change in the future. In the future we may need this object more, or wish to add more functionality or constraints to them. If so, those changes will be easier to manage because we encapsulated row and column.

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 [5][8] 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. I have provided contracts for some, but not all of the methods. You must include the provided contracts as well as write contracts for the remaining methods. You must provide Javadoc comments for each method.

`public boolean checkIfFree(int c)`: returns true if column is able to accept another token, false otherwise.

`public boolean checkForWin(int c)`: returns true if the last token placed (which was placed in column `c`) resulted in the player winning the game. Otherwise, it returns false. **Note:** this is not checking the entire board for a win, it is just checking if the last token placed results in a win.

`public boolean checkTie()`: returns true if the game board results in a tie game, otherwise it returns false. **Note:** use your preconditions to make this much simpler. You should not be checking to see if the board contains any win conditions.

`public 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`.

`public boolean checkHorizWin(BoardPosition pos, char p)`: returns true if the last token placed (which was placed in position `pos` and player `p`) resulted in the player winning by getting 5 in a row horizontally. Otherwise, it returns false.

`public boolean checkVertWin(BoardPosition pos, char p)`: returns true if the last token placed (which was placed in position `pos` and player `p`) resulted in the player getting 5 in a row vertically. Otherwise, it returns false.

`public boolean checkDiagWin(BoardPosition pos, char p)`: returns true if the last token placed (which was placed in position `pos` and player `p`) resulted in the player getting 5 in a row diagonally. Otherwise, it returns false.

`public char whatsAtPos(BoardPosition pos)`: returns the char that is in position `pos` of the game board. If there is no token at the spot it should return a blank space character: ' '.

`public boolean isPlayerAtPos(BoardPosition pos, char player)`: returns true if the player is at that position. **Note:** this method will be implemented very similarly to `whatsAtPos`, but it's asking a different question. We only know they will be similar because we know `GameBoard` will contain a 2D array. If the data structure were to change in the future these two methods could end up being radically different. When designing our class, we don't want to focus on the data structure, but more on the concept it represents and how we would want to interact with it. It's helpful to think about what questions we would ask the object, and "What's at this position?" and "Is this player at this position?" are different questions, so we have different methods for them.

`public 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. This does not print to the screen, it returns a `String` that represents the game board.

Additionally, this class must have a default constructor (no parameters). You should not add any other methods to this class.

For this stage of the assignment, you will need to include in your project report:

- A UML class diagram for `GameBoard`
- A UML activity diagram for each method in `GameBoard`

Additionally, you will submit a code file for `GameBoard` called `GameBoard.java`. This code file will just be an outline for the eventual code file. It should include the signature, Javadoc comments, and contracts for all of the methods and invariants for the class, but you do not need to write the code for each method yet. Because of this, your code will not compile yet.

Contracts, Comments, and formatting

For every function, you will need to create Javadoc comments, and create contracts for each function. Contracts should be formally stated when possible. We have not covered much formal notation, but if a parameter named `row` needs to be greater than 0, then the precondition should say "row > 0" not "[row must be greater than 0]." The invariant for the classes should also be specified in Javadoc comments.

Everything that has been mentioned as a "best practice" in our lectures or in a video should be followed, or you risk losing points on your assignment.

Project 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 as specified in the above directions. All diagrams should be well formatted and easy to read. I recommend Diagrams.net (formally Draw.io) as a program to create the diagrams. You may not submit hand drawn diagrams, even if they are scanned in or drawn on a table.

The project report should be one organized PDF file. You should not be submitting individual image files for the diagram, or separate reports for the design and requirements. Your diagrams should be readable so consider their layout on the page when making them. Don't just try to cram all the diagrams into one image.

Questions to consider while designing your program:

- Remember to consider possible changes that could be made to this program
- Remember to consider what information is publicly available and what is hidden. All data fields should be private, and you should not add any other public functions other that the ones specified.
- Remember to consider what each class knows, and what it cares about. Follow separation of concerns.
- Our contracts make our code easier to write by defining what situations should not even be considered. Write those first before making activity diagrams.

General Notes:

- Name your package `cpsc2150.extendedConnectX`
- Remember our "best practices" that we've discussed in class. Use good variables, avoid magic numbers, etc.
- Start early. You have time to work on this assignment, but you will need time to work on it.
- Starting early means more opportunities to go to TA or instructor office hours for help
- The first part of the assignment is due Wednesday, January 27th at 10:00 pm. See the submission section for more information.
- The full assignment is due Sunday, February 7th at 10:00 pm. The instructor and TAs have lives as well, so don't expect email responses late at night. If you need help, you will need to start early enough to ask for help.
- Start by analyzing the requirements of this program to ensure you understand what you are designing.
- 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.

- Your UML Diagrams must be made electronically. I recommend the freely available program Diagrams.net (formally Draw.io), but if you have another diagramming tool you prefer feel free to use that. It may be faster to draw rough drafts by hand, and then when you have the logic worked out create the final version electronically.
- 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."
- You may want to call one `public` method from another `public` method. We can do this, but we have to be very careful about doing this. Remember that during the process of a `public` method we cannot assume that the invariant true, but at the beginning of one we assume that it is true. So, if we want to call one `public` method from inside another, we need to be sure that the invariants are true as well as the preconditions of the method we are calling.
- Activity Diagrams should be detailed. Every decision that is made should be shown, not just described in one step. I.E. it is not sufficient to have one step/activity in the diagram say "Check if the input is valid and prompt again if it is not." You need to show the logic for that check. Calling another method can be a single step in the activity diagram since that method will have its own diagram. The activities in the diagrams do not need to be written as code. It is fine to write "Check if X is in position i, j" instead of "if ('X' == board[i][j])"
- You do not need to write the actual code yet. Do not try to solve this by writing the code, then drawing diagrams to match your code. It saves time in the long run to use the activity diagrams to work through the complicated logic.
- `GameScreen` is the only class that should get input from the user or print to the screen.
- 0,0 should be the bottom left corner of the board. If you do not make 0,0 the bottom left corner of the board it could cause issues with the scripts the TA will use to grade the assignment.
- Future assignments will build off of this assignment. If you do not put your best effort into this assignment, then you will just make future assignments more difficult.

Submission:

To ensure that you do not wait until the last minute to start the assignment, we are requiring two submissions for this assignment. Part 1 will be due Wednesday, January 27th at 10:00 pm, while the full assignment is due Sunday, February 7th at 10:00 pm. **For part 1, submit completed class diagrams and requirements analysis as a single PDF.** For this part, we will only be checking for completeness and not correctness, so you will have time to fix your class diagrams by listing new helper functions or class variables and/or add or remove requirements before you submit your full assignment by Sunday, February 7th at 10:00 pm.

You will submit your files on Handin. It will be labeled as **Project1-Part1** and **Project1**. Please make sure you submit to the correct Handin folder by the specified due date. You should place your code files in the appropriate package structure for Unix (inside a `cpsc2150/extendedConnectX` folder). Place that package structure in a folder with your project report, then zip up that folder to submit. Make sure you submit the code files (.java) not the compiled files (.class). Your program report

should be a PDF. Include all the files with your submission. Late submissions will not be accepted. If you "Forget" to include something and cannot prove it was done before the deadline (time stamps on your personal machine are not sufficient evidence) you will not be able to submit it later for credit.

Disclaimer:

It is possible that these instructions may be updated. As students ask questions, I may realize that something is not as clear as I thought it was, and I may add detail to clarify the issue. Changes to the instructions will not change the assignment drastically, and will not require reworking code. They would just be made to clarify the expectations and requirements. If any changes are made, they will be announced on Canvas.

Sample input and output:

```
|0|1|2|3|4|5|6|7|8|
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |

Player X, what column do you want to place your marker in?
3
|0|1|2|3|4|5|6|7|8|
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | |X| | | | | |

Player O, what column do you want to place your marker in?
3
|0|1|2|3|4|5|6|7|8|
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | |O| | | | | |
| | | |X| | | | | |

Player X, what column do you want to place your marker in?
3
|0|1|2|3|4|5|6|7|8|
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
```

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

Player O, what column do you want to place your marker in?
3
|0|1|2|3|4|5|6|7|8|
| | | | | | | | | |
| | | | | | | | | |
| | | |O| | | | | |
| | | |X| | | | | |
| | | |O| | | | | |
| | | |X| | | | | |

Player X, what column do you want to place your marker in?
3
|0|1|2|3|4|5|6|7|8|
| | | | | | | | | |
| | | |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|7|8|
| | | |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|7|8|
| | | |O| | | | | |
| | | |X| | | | | |
| | | |O| | | | | |
| | | |X| | | | | |
| | | |O| | | | | |
| | | |X|X| | | | |

Player O, what column do you want to place your marker in?
```

```
4
|0|1|2|3|4|5|6|7|8|
| | | |O| | | | | |
| | | |X| | | | | |
| | | |O| | | | | |
| | | |X| | | | | |
| | | |O|O| | | | |
| | | |X|X| | | | |

Player X, what column do you want to place your marker in?
5
|0|1|2|3|4|5|6|7|8|
| | | |O| | | | | |
| | | |X| | | | | |
| | | |O| | | | | |
| | | |X| | | | | |
| | | |O|O| | | | |
| | | |X|X|X| | | |

Player O, what column do you want to place your marker in?
-1
Column cannot be less than 0
Player O, what column do you want to place your marker in?
9
Column cannot be greater than 8
Player O, what column do you want to place your marker in?
5
|0|1|2|3|4|5|6|7|8|
| | | |O| | | | | |
| | | |X| | | | | |
| | | |O| | | | | |
| | | |X| | | | | |
| | | |O|O|O| | | |
| | | |X|X|X| | | |

Player X, what column do you want to place your marker in?
1
|0|1|2|3|4|5|6|7|8|
| | | |O| | | | | |
| | | |X| | | | | |
| | | |O| | | | | |
| | | |X| | | | | |
| | | |O|O|O| | | |
| |X| |X|X|X| | | |

Player O, what column do you want to place your marker in?
6
|0|1|2|3|4|5|6|7|8|
```

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

Player X, what column do you want to place your marker in?
2
|0|1|2|3|4|5|6|7|8|
| | | |O| | | | | |
| | | |X| | | | | |
| | | |O| | | | | |
| | | |X| | | | | |
| | | |O|O|O| | | |
| |X|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|7|8|
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |

Player X, what column do you want to place your marker in?
1
|0|1|2|3|4|5|6|7|8|
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| |X| | | | | | | |

Player O, what column do you want to place your marker in?
2
|0|1|2|3|4|5|6|7|8|
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
```

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

Player X, what column do you want to place your marker in?
1
|0|1|2|3|4|5|6|7|8|
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| |X| | | | | | | |
| |X|O| | | | | | |

Player O, what column do you want to place your marker in?
2
|0|1|2|3|4|5|6|7|8|
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| |X|O| | | | | | |
| |X|O| | | | | | |

Player X, what column do you want to place your marker in?
1
|0|1|2|3|4|5|6|7|8|
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| |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|7|8|
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| |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|7|8|
| | | | | | | | | |
| | | | | | | | | |
| |X| | | | | | | |
```

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

Player O, what column do you want to place your marker in?
2
|0|1|2|3|4|5|6|7|8|
| | | | | | | | | |
| | | | | | | | | |
| |X|O| | | | | | |
| |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|7|8|
| | | | | | | | | |
| |X| | | | | | | |
| |X|O| | | | | | |
| |X|O| | | | | | |
| |X|O| | | | | | |
| |X|O| | | | | | |

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

Process finished with exit code 0
```