Project 2

CPSC 2150
Due: Sunday, Oct 11th at 10:00 pm
100 pts

Introduction:

In this assignment, we will be implementing the design you created in Project 1. While hopefully the TAs will be able to provide feedback on that assignment quickly, it is unlikely that the feedback will be available with much (or any) time left before this assignments due date. You will need to be proactive in evaluating and fixing your own design

Your implemented code should include the three classes described in the Project 1 prompt and should match all requirements from that prompt. Some minor changes are listed below as new requirements.

Again, it is important to follow the design and requirements laid out in the Project 1 prompt. Any changes could cause issues with later assignments. While it is important to consider possible future changes and write our code in a way that will make it easier to adapt to those future changes, we should not write code that implements *predicted* future changes. Any deviation from the requirements could result in breaking automated grading scripts, and result in losing points due to not following directions.

New Requirements:

You must provide an interface called `IGameBoard` that the `GameBoard` class will implement. The `IGameBoard` interface should contain all the methods from Project 1, as well as the following methods:

`public int getNumRows()` – returns the number of rows in the `GameBoard`
`public int getNumColumns()` – returns the number of columns in the `GameBoard`
`public int getNumToWin()` – returns the number of tokens in a row needed to win the game

For each method, determine whether or not is a primary or a secondary method. Any secondary methods in the `IGameBoard` interface should be provided as a `default` method in the interface itself and not overridden in the implementation (unless you gain an efficiency advantage by doing so). Consider whether or not it *can* be secondary and still meet the requirements. If you added in private helper functions or additional private data variables that may have made other methods easier, but by removing them you may be able to make the method secondary.

- Correctly identifying all the secondary methods and coding them to be `default` methods is a large part of what you will be evaluated on for this assignment. Spend some time considering each method.
- You may choose to override the `default` implementation if you can get an efficiency gain by directly accessing the private data, but you are not required to do so.
- **Example:** You may have come up for a solution to more check if there is a tie game by keeping track of the number of markers that have been placed on the board in a private field. You would not be able to access this private field in the interface, however there is a way to check to see if there is a tie game in the interface without accessing that

private variable. You should still provide a `default` implementation to check for a tie game, and you can choose to override it in `GameBoard` since it would be much faster to just check the private data.

You must add in an `abstract class` called `AbsGameBoard` which implements `IGameBoard`. `AbsGameBoard` will have no private data, and will just provide an overridden implementation of `toString` which will be implemented by calling primary methods. We are not allowed to override `toString` in our interface (even though it could be implemented as a `default` method) because an `interface` does not extend `Object`, and therefore cannot override the `toString` method provided by `Object`. `GameBoard` should extend `AbsGameBoard`.

Contracts, Comments, and formatting

For every function, you should already have Javadoc comments and contracts available. If you made mistakes on these for the Project 1 submission, you should correct them now. Again, you may not have feedback from the TAs on your submission, but while working on your code you may discover on your own that you made a mistake.

You will need to create a complete interface specification for the `IGameBoard` interface, which includes an abstraction for `IGameBoard`. You have written most of these already, but you need to copy them to the interface file and make the appropriate changes, so it uses the new abstraction you have defined and modify your preconditions and postconditions for each method accordingly. The descriptions, `@params` and `@returns` for each method should remain the same as Project 1. You do not need to repeat any contracts from the interface in `GameBoard`.

Now that you have an interface specification, you must provide correspondences and invariants in your `GameBoard` class.

In general, your code should be well commented. Your Javadoc comments will handle a lot of this, but you will need more comments in longer and more detailed sections of code. Your code should be properly indented, you should use good variable names, you should follow naming conventions, you should avoid magic numbers, etc. 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.

Your code should be easy to read. Ask yourself, would someone with no knowledge of this assignment be able to tell what is happening in the program from quickly reading my code.

Project Report.

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

<u>Requirements Analysis</u>

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. (**Note:** these should only change if you identify a mistake you made)

<u>Design</u>

Create a UML class diagram for each class and interface 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. Most of these should

already be completed from the Project 1 submission, however once you implement and test your code you may discover you made a mistake. If so, you should correct the mistake in your code, but also correct the diagrams as well. As software engineers, we use UML diagrams not just as a design tool, but as a way to document our code so other software engineers have reference materials to help them when they need to maintain, fix, test, or build off of our code.

Deployment

You must include a `makefile` with the following targets:

- `default`: compiles your code. Runs with the `make` command.
- `run`: runs your code. Runs with the `make run` command.
- `clean`: removes your compiled (`.class`) files. Runs with the `make clean` command.

Your project report should include instructions on how to use the `makefile` to compile and run your program.

General Notes and additional requirements:

- If it does not compile, it is a zero. Make sure it will compile on SoC Unix machines.
- Name your package `cpsc2150.extendedTicTacToe`
- Remember our "best practices" that we've discussed in class. Use good variable names, 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
- 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])"

- `GameScreen` is the only class that should get input from the user or print to the screen.
- 0,0 should be the top left of the board. If you do not make 0,0 the top left of the board it could cause issues with the scripts the TA will use to grade the assignment.
- If you identify any mistakes in your design while implementing it, make sure to correct your diagrams in the project report to ensure it matches your code.
- You should not have any "Dead Code" in your code. "Dead Code" is code that is commented out because it is no longer used.
- Since there is a delay in the feedback for the previous assignment, you will not be docked for mistakes you made on Project 1 that you did not know about. However, if it is an obvious mistake that you should have known about, or something that you just didn't do for Project 1, then you could lose points for it again.

Submission:

You will submit your program on Handin. It will be labeled as Project2. 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 `makefile`. Make sure your code is your Java files and not your `.class` files. The TA should be able to unzip your folder and type "`make`", "`make run`" to start running your code and "`make clean`" to remove any compiled (`.class`) files.

Your assignment is due at 10:00 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. No late assignments will be accepted.

You are responsible for ensuring your submission is correct. After submitting to Handin you can view your submission, download it, and unzip it to make sure everything is correct. Handin will allow you to resubmit as well.

This is an INDIVIDUAL assignment. You should not be working on this assignment with any other student.

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.

Checklist

Use this checklist to ensure you are completing your assignment. **Note:** this list does not cover all possible reasons you could miss points but should cover a majority of them. I am not intentionally leaving anything out, but I am constantly surprised by some of the submissions we see that are wildly off the mark. For example, I am not listing "Does my program play Tic Tac Toe?" but that does not mean that if you turn in a program that plays Checkers you can say "But it wasn't on the checklist!" The only complete list that would guarantee that no points would be lost would be an incredibly detailed and

complete set of instructions that told you exactly what code to type, which wouldn't be much of an assignment.

- Is my `Gameboard` the correct size?
- Do players need 5 in a row to win?
- Does my game have 2 players?
- Do my classes have invariants?
- Does my `GameBoard` class have correspondences?
- Does my game allow for players to play again?
- Are my methods reasonably efficient? You don't need to take this to the extreme, but obvious inefficiencies should be corrected.
- Does my game take turns with the players?
- Does my game correctly identify wins?
- Does my game correctly identify tie games?
- Does my game run without crashing?
- Does my game validate user input?
- Did I protect my data by keeping it private, except for `public static final` (constant) variables?
- Did I encapsulate my data and functionality and include them in the correct classes?
- Did I follow Design-By-Contract?
- Did I provide contracts for each method?
- Did I comment my code?
- Did I provide Javadoc comments for each method?
- Did I avoid using magic numbers?
- Did I use good variable names?
- Did I follow best practices?
- Did I remove "Dead Code." Dead Code is old code that is commented out.
- Did I make any additional helper functions I created private?
- Did I use the `static` keyword correctly?
- Did I express all functional requirements as user stories?
- Did I create my activity diagrams for all methods?
- Did I create a class diagram for each class?
- Did I provide a working `makefile`?
- Does my code compile and run on Unix?
- Is my program written in Java?
- Does my output look like the provided examples?
- Did I create my `interface` and write the full interface specification?
- Did I add my `abstract class`?

Sample input and output:

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

Player X Please enter your ROW
12
Player X Please enter your COLUMN
5
  0|1|2|3|4|5|6|7|
0| | | | | | | | |
1| | | | | | | | |
2| | | | | | | | |
3| | | | | | | | |
4| | | | | | | | |
5| | | | | | | | |
6| | | | | | | | |
7| | | | | | | | |

That space is unavailable, please pick again
Player X Please enter your ROW
5
Player X Please enter your COLUMN
12
  0|1|2|3|4|5|6|7|
0| | | | | | | | |
1| | | | | | | | |
2| | | | | | | | |
3| | | | | | | | |
4| | | | | | | | |
5| | | | | | | | |
6| | | | | | | | |
7| | | | | | | | |

That space is unavailable, please pick again
Player X Please enter your ROW
5
Player X Please enter your COLUMN
3
  0|1|2|3|4|5|6|7|
0| | | | | | | | |
```

```
1| | | | | | | | |
2| | | | | | | | |
3| | | | | | | | |
4| | | | | | | | |
5| | | |X| | | | |
6| | | | | | | | |
7| | | | | | | | |

Player O Please enter your ROW
0
Player O Please enter your COLUMN
0
  0|1|2|3|4|5|6|7|
0|O| | | | | | | |
1| | | | | | | | |
2| | | | | | | | |
3| | | | | | | | |
4| | | | | | | | |
5| | | |X| | | | |
6| | | | | | | | |
7| | | | | | | | |

Player X Please enter your ROW
4
Player X Please enter your COLUMN
3
  0|1|2|3|4|5|6|7|
0|O| | | | | | | |
1| | | | | | | | |
2| | | | | | | | |
3| | | | | | | | |
4| | | |X| | | | |
5| | | |X| | | | |
6| | | | | | | | |
7| | | | | | | | |

Player O Please enter your ROW
3
Player O Please enter your COLUMN
3
  0|1|2|3|4|5|6|7|
0|O| | | | | | | |
1| | | | | | | | |
2| | | | | | | | |
3| | | |O| | | | |
4| | | |X| | | | |
5| | | |X| | | | |
6| | | | | | | | |
```

```
7| | | | | | | | |

Player X Please enter your ROW
5
Player X Please enter your COLUMN
2
  0|1|2|3|4|5|6|7|
0|O| | | | | | | |
1| | | | | | | | |
2| | | | | | | | |
3| | | |O| | | | |
4| | | |X| | | | |
5| | |X|X| | | | |
6| | | | | | | | |
7| | | | | | | | |

Player O Please enter your ROW
1
Player O Please enter your COLUMN
1
  0|1|2|3|4|5|6|7|
0|O| | | | | | | |
1| |O| | | | | | |
2| | | | | | | | |
3| | | |O| | | | |
4| | | |X| | | | |
5| | |X|X| | | | |
6| | | | | | | | |
7| | | | | | | | |

Player X Please enter your ROW
3
Player X Please enter your COLUMN
3
  0|1|2|3|4|5|6|7|
0|O| | | | | | | |
1| |O| | | | | | |
2| | | | | | | | |
3| | | |O| | | | |
4| | | |X| | | | |
5| | |X|X| | | | |
6| | | | | | | | |
7| | | | | | | | |

That space is unavailable, please pick again
Player X Please enter your ROW
3
Player X Please enter your COLUMN
```

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

Player O Please enter your ROW
2
Player O Please enter your COLUMN
2
  0|1|2|3|4|5|6|7|
0|O| | | | | | | |
1| |O| | | | | | |
2| | |O| | | | | |
3| | | |O|X| | | |
4| | | |X| | | | |
5| | |X|X| | | | |
6| | | | | | | | |
7| | | | | | | | |

Player X Please enter your ROW
2
Player X Please enter your COLUMN
5
  0|1|2|3|4|5|6|7|
0|O| | | | | | | |
1| |O| | | | | | |
2| | |O| | |X| | |
3| | | |O|X| | | |
4| | | |X| | | | |
5| | |X|X| | | | |
6| | | | | | | | |
7| | | | | | | | |

Player O Please enter your ROW
4
Player O Please enter your COLUMN
4
Player O wins!
  0|1|2|3|4|5|6|7|
0|O| | | | | | | |
1| |O| | | | | | |
2| | |O| | |X| | |
```

```
3| | | |O|X| | | |
4| | | |X|O| | | |
5| | |X|X| | | | |
6| | | | | | | | |
7| | | | | | | | |

Would you like to play again? Y/N
n
```