CPSC 219: Introduction to Computer Science for Multidisciplinary Studies II

Assignment 1: Procedural Java, Git, and JUnit

Weight: 10%

Collaboration

Discussing the assignment requirements with others is a reasonable thing to do and an excellent way to learn. However, the work you hand in must ultimately be your work. This is essential for you to benefit from the learning experience and for the instructors and TAs to grade you fairly. Handing in work that is not your original work but is represented as such is plagiarism and academic misconduct. Penalties for academic misconduct are outlined in the university calendar.

Here are some tips to avoid plagiarism in your programming assignments.

1. Cite all sources of code you hand in that are not your original work. You can put the citations into comments in your program. For example, if you find and use code found on a website, include a comment that says, for example:

   ```
   # The following code is from
   https://www.quackit.com/python/tutorial/python_hello_world.cfm.
   ```

   Use the complete URL so that the marker can check the source.

2. A tool like chat-GPT can be used to improve small code blocks. For example, five lines of code. If you get help from code assistance like Chat-GPT, you should comment above the block of code you requested assistance on debugging or improving and cite the tool used to get that suggestion. Using a tool like chat-GPT to write the majority of your assignment requirements will be treated as plagiarism if found without citation, and with citation, it will be treated as 0 for the component the student did not complete. Code improvement of short length will get credit if commented/cited properly.
3. Citing sources avoids accusations of plagiarism and penalties for academic misconduct. **However, you may still get a low grade if you submit code not primarily developed by yourself. Cited material should never be used to complete core assignment specifications. Before submitting, you can and should verify any code you are concerned about with your instructor/TA.**
4. Discuss and share ideas with other programmers as much as you like, but make sure that when you write your code, it is your own. A good rule of thumb is to wait 20 minutes after talking with somebody before writing your code. If you exchange code with another student, write code while discussing it with a fellow student, or copy code from another person's screen, this code is not yours.
5. **Collaborative coding is strictly prohibited**. **Your assignment submission must be strictly your code**. Discussing anything beyond assignment requirements and ideas is a strictly forbidden form of collaboration. This includes sharing code, discussing the code itself, or modelling code after another student's algorithm. **You can not use (even with citation) another student's code.**
6. Making your code available, even passively, for others to copy or potentially copy is also plagiarism.
7. We will look for plagiarism in all code submissions, possibly using automated software designed for the task. For example, see Measures of Software Similarity (MOSS - https://theory.stanford.edu/~aiken/moss/).

8. Remember, if you are having trouble with an assignment, it is always better to go to your TA and/or instructor for help rather than plagiarizing. A common penalty is an F on a plagiarized assignment.

Late Policy

Assignments will be accepted up until 48 hours late with a penalty. From $0 < hours \leq 24$ will result in 10% penalty. From $24 < hours \leq 48$ will result in a 20% penalty.

Goal

Writing a first program in **Java** with a standard CPSC 217/231 procedural structure. Use **Git** version control properly to store this project. Perform unit testing via **JUnit** to establish the correctness of portions of the assignment code created.

Technology

Java 20, Git, JUnit 5

Submission Instructions

You must submit your assignment electronically using **GitLab** and **D2L**. Use the Assignment 1 dropbox in **D2L** for a final codebase electronic submission. You will also share a link to your **GitLab** codebase with your TA in that D2L submission. In **D2L**, you can submit multiple times over the top of a previous submission. Do not wait until the last minute to attempt to submit. You are responsible if you attempt this, and time runs out. Your assignment must be completed in **Java** (not Kotlin or others) and be executable with **Java version 20.** You must use **Gitlab** hosted at **https://csgit.ucalgary.ca/** (not GitHub or another Git host). You must use **JUnit 5** and not other unit-testing libraries.

Description

You will complete a Java program which plays a Tic Tac Toe game. This game will be procedurally designed (no classes and objects and completely within one Board.java file). This game must use the exact function names requested in this assignment document. You will complete these functions and then combine them to create a Tic Tac Toe game that uses these functions. While creating this game, you will be tasked with creating unit tests for these functions using **JUnit 5**. At the same time, you will be expected to store your code in a **private** repository at **csgit.ucalgary.ca** using **Git** as you work on the assignment.

The game of X's and O's, Tic Tac Toe, or Noughts and Crosses has many different names. Two players are given a square array of size **three** and take turns entering their symbol into the grid. The winner is the first to **three** in a row (across, down, or diagonal). As mentioned in class, this game is solved. That is, there is a known strategy for three-by-three such that a 'perfect' player can never lose the game. Two 'perfect' opposing players will always play to a draw.

We will be implementing a flexible version of the game. The user will be able to play the base game on a **three-by-three** grid but will also have the option to play with row and column combinations selected from the sizes of **3**, **4**, or **5**. For example, boards can be **three-by-five**, **four-by-three**, or even **five-by-five** in size. The user will also be able to select a win length equal to or greater than the standard win length of **three**. The longest row or column will limit the largest size. e.g., A five-by-five board could have win lengths of 3, 4, or 5 chosen.

You will be provided with three files. A **Game.java** file contains the game logic you do not need to change, a **Starter.java** file in which you must add the requested functions and a **BoardTest.java** file where you will create your **JUnit** tests. The **Game.java** code should not be changed unless communication with the instructor approves a change. The **Game.java** file will look for the functions inside a file called **Board.java**. You must rename the **Starter.java** file to **Board.java** and change the internal class from **public class Starter** to **public class Board** inside the file.

About 12 different functions require completion for the Tic Tac Toe game to operate correctly. Your job will be to complete the implementation of **Board.java** by implementing each of its methods to the specifications indicated in the assignment. You must modify a BoardTest.java JUnit 5 file to test these completed functions. You will also be expected to properly comment **Board.java** and **BoardTest.java**, use regular version control hosted at **csgit.ucalgary.ca** to track as you change things and submit these **Board.java** and **BoardTest.java** in D2L.

Do not change the existing portions of the code in **Game.java**. If you attempt the bonus, create a new **GameBonus.java** file, **BoardBonus.java** file, and **BoardBonusTest.java** file in addition to the files for the non-bonus version. You will lose those marks if you break your non-bonus program while doing the bonus. YOU SHOULD NOT IMPORT ANY OTHER LIBRARIES FOR THIS ASSIGNMENT, and you will not get credit for code copied from other places.

Git Requirement: As we move through topics, we will introduce Git as a version control system. For this assignment, we will require you to upload your code to the university **csgit.ucalgary.ca** hosting site as a **private** repository shared with your TA (in addition to submitting it via D2L Dropbox). *If you are comfortable with Git before we cover it, you can start this process early and do all your code storage in Gitlab.* If Git is new to you, you will be taught it and won't expect regular commits for **Board.java** but will expect them for **BoardTest.java.**

The minimum expectation for Git usage is that when you submit your code, the TA can go and view it in Gitlab and see that you've made multiple commits as you edited it before the submission deadline to D2L. To use csgit.ucalgary.ca, you will need a functional **UCIT account** username/password.

Example Tic-Tac-Toe Boards. (MxN -> M is rows, N is columns)

EMPTY = 0, X = 1, O = 2

3x3 – start of game

0	0	0
0	0	0
0	0	0

4x5 – win \ diagonal for O

2	0	1	0	0
0	2	0	1	0
0	0	2	0	0
0	0	0	0	0

3x3 full board

2	1	1
2	1	2
1	2	2

4x3 – win in row 1 for X

0	0	0
1	1	1
0	2	0
0	0	2

4x4 – game undecided

0	0	0	0
1	1	0	1
0	2	0	0
0	0	2	0

Program Coding Requirement:

The 11 functions you must complete are as follows (all must be public static functions). I recommend first creating all 11 function definitions (you don't need to finish the inside of any immediately). You can either Javadoc comment ** *\ the functions as you make them or do this later. Remember to inline \\ comment your functions as well. At the same time, don't forget to add your name and student info to the **Board.java** file for your TA. These all have marks in the grading rubric.

(Note: When we say **integer**, we will mean the primitive type **int** for this assignment.)

Function Name: **createBoard**

Parameters: **rows**: integer, **columns**: integer

Return: 2D integer array

*Assume **rows/columns** are valid positive integers in the inclusive range [3,5].*

*Create and return a 2D integer array for the board of the game. Filled with **EMPTY** = 0 pieces.*

***Rows** should be the size of the first dimension of the array and **columns** the second dimension.*

Board.createBoard(3,3);

0	0	0
0	0	0
0	0	0

Function Name: **rowCount**

Parameters: **board**: 2D integer array

Return: integer

*Assume the **board** is a valid 2D int array.*

*Take in a **board** and return the integer number of rows the **board** has (i.e. the size of the first dimension of the array).*

Board.rowCount(board) == 4;

0	0	0
1	1	1
0	2	0
0	0	2

Function Name: **columnCount**

Parameters: **board**: 2D integer array

Return: integer

*Assume the **board** is a valid 2D int array.*

*Take in a **board** and return the integer number of columns the **board** has (i.e. the size of the second dimension of the array).*

Board.columnCount(board) == 3;

0	0	0
1	1	1
0	2	0
0	0	2

Function Name: **canPlay**

Parameters: **board**: 2D integer array, **row**: integer, **column**: integer

Return: boolean

*Assume the **board** is a valid 2D int array and **row/column** are valid indices in the **board**.*

*Return boolean True if the location in the **board** at the indicated **row/column** index is open (**EMPTY**).*

Board.canPlay(board,0,0) == true;

Board.canPlay(board,1,1) == false;

0	0	0
1	1	1
0	2	0
0	0	2

Function Name: **play**

Parameters: **board**: 2D integer array, **row**: integer, **column**: integer, **piece**: integer

Return: nothing (void)

*Assume the **board** is a valid 2D int array, **row/column** are valid indices in the **board**, and the **piece** is X==1/O==2. Assume the location (**row, column**) is **EMPTY** in the **board**.*

*Play by assigning a **piece** in the location on the **board** at the indicated **row/column**.*

Board.play(board,0,0,X) ;

0	0	0
1	1	1
0	2	0
0	0	2

→

1	0	0
1	1	1
0	2	0
0	0	2

Function Name: **full**

Parameters: **board**: 2D integer array

Return: boolean

*Assume the **board** is a valid 2D int array.*

*Return true if the **board** is filled with non-EMPTY pieces. Otherwise, false.*

Board.full(board) == true;

2	1	1
2	1	2
1	2	2

Function Name: **winInRow**

Parameters: **board**: 2D integer array, **row**: integer, **piece**: integer, **length**: integer

Return: boolean

*Assume the **board** is a valid 2D int array, the **row** is a valid index in the board, the **piece** is X==1/O==2, and the **length** is a valid win length for a given size.*

*Look at the indicated **row** at the given index on the **board**. If that **row** has at least **length** consecutive entries with a given type of **piece** (X/O), then return true; otherwise, false.*

Board.winInRow(board, 0, X, 3) == false;

Board.winInRow(board, 0, O, 3) == false;

Board.winInRow(board, 1, X, 3) == true;

Board.winInRow(board, 1, O, 3) == false;

0	0	0
1	1	1
0	2	0
0	0	2

Function Name: **winInColumn**

Parameters: **board**: 2D integer array, **column**: integer, **piece**: integer, **length**: integer

Return: boolean

*Assume the **board** is a valid 2D int array, the **column** is a valid index in the board, the **piece** is X==1/O==2, and the **length** is a valid win length for a given size.*

*Look at the indicated **column** at the given index on the **board**. If that **column** has at least **length** consecutive entries with a given type of **piece** (X/O), then return true; otherwise, it is false.*

Board.winInColumn(board, 0, X, 3) == true;

Board.winInColumn (board, 0, O, 3) == false;

Board.winInColumn (board, 0, X, 4) == false;

Board.winInColumn (board, 1, X, 3) == false;

1	0	0
1	2	1
1	2	0
0	0	2

Function Name: **winInDiagonalBackslash**

Parameters: **board**: 2D integer array, **piece**: integer, **length**: integer

Return: boolean

*Assume the **board** is a valid 2D int array, and the **piece** is X==1/O==2, and the **length** is a valid win **length** for a given size.*

*Look at all backward slash \ diagonals in the **board**. If any backward slash \ diagonals have at least **length** consecutive entries with a given type of **piece** (X/O), then return true; otherwise, false.*

Function Name: **winInDiagonalForwardSlash**

Parameters: **board**: 2D integer array, **piece**: integer, **length**: integer

Return: boolean

*Assume the **board** is a valid 2D int array, the **piece** is X==1/O==2, and the **length** is a valid win **length** for a given size.*

*Look at all forward slash/diagonals on the **board**. If any forward slash/diagonals have at least **length** consecutive entries with a given type of **piece** (X/O), then return true; otherwise, false.*

Function Name: **hint**

Parameters: **board**: 2D integer array, **piece**: integer, **length**: integer

Return: 1D integer array (length 2) where array stores {row, column} of hint

*Assume the **board** is a valid 2D int array, the **piece** is X==1/O==2; **length** is a valid win **length** for a given size*

The hint scans across each row left to right, starting at the top row and working down in rows. It returns a 1D integer array {row, column} containing the first hint found for the piece given to win. It does not examine for other hints, like blocking the opponent's win or other hints. The only hint returned will be the first hint found that wins the game on the next play for the player. It returns {-1, -1} if it does not find a hint.

You are required to follow this pseudo-code for the hint function:

```
For every row on the board board
    For every column on the board
        If we can play at this row and column
            Play the player's piece
            If the player has won the game
                Remove the player's piece from the last played location
                Return the row and column of the hint
            Otherwise, nobody won the game,
                Remove the player's piece from the last played location
Default return -1 for both row and column
```

Board.hint(board, X, 3) == new int[]{1,0}

1	0	0
0	2	1
1	2	0
0	0	2

Unit Testing Requirement:

For the second part of the assignment, we will be adding in Unit Testing. This will be accomplished via JUnit5. You must create and submit a modified **BoardTest.java** file using JUnit5. This file will consist of unit tests written to test the above 11 functions plus the **public static BigInteger factorial(int n)** that is already complete in the provided starter **Game.java** file. So, there are 12 different functions to test in total.

For each function, design 1 to 5 tests. The quantity of these tests you create will be based on your decision of what is necessary to test for a function. To determine what to test, you should read the function requirements in this document. Do not test for things that you are told to assume are correct inputs. *For example, don't test createBoard for input sizes that are 2 or 6. The valid range of input sizes is 3,4,5 for row/column.* In general, all functions are expected to have correct, valid inputs, so you will be testing if the output of your function is correct when given a valid input.

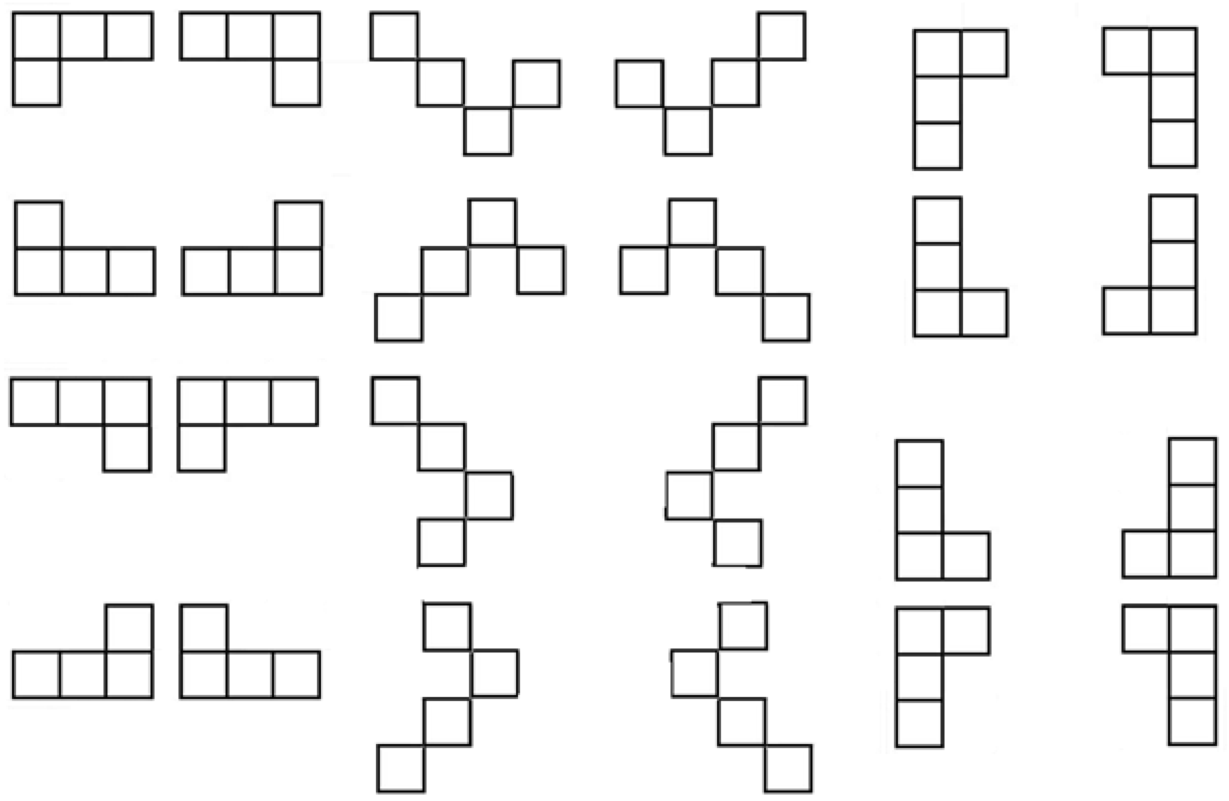
A couple of simple functions will need 1 or 2 tests, but you will notice many will need more tests than 5. I don't want you to spend time making too many tests, so I've limited the required test quantity for grading to 5 at most for each. (You are free to make more, but the TAs will only mark the five you submit for each). Your goal with unit tests is to make up a variety of tests that demonstrate different challenges for a function. TAs will look for a range of variety of tests for a full grade. If they see five tests for a function that are almost identical in purpose, then they will not get a full grade.

Remember, a good unit test examines one thing. So when the test fails, you know exactly what to look for to fix it. The goal should be one input and one output per test. Resist the urge to make tests that explore more than one input, as these will not get a full grade.

Bonus

To complete the bonus, you must create a second **GameBonus.java** file, **BoardBonus.java** file, and **BoardBonusTest.java** file.

A win will now look like a Tetris L instead of a line. Requiring this on a three-by-three board would not make any sense, so 3x3 boards will retain the old 3 in a row. But if any length of the board is 4 or 5, then a win length selection of 4 or 5 will result in a three-long L shape. The new win types should look like the following.



Modify the game functions such that a win (for size 4/5 boards now) requires three pieces in a sequence and one more piece perpendicular to the end of the sequence to complete an L shape. As a hint, a way to do this would be to track the start and end of when you find three pieces in a sequence. Then, check if this extra perpendicular piece exists relative to either end location of the sequence.

Create/modify your existing TicTacToeTest.java to create a new TicTacToeBonusTest.java file to check for these new requirements. You should only need to modify the test winInNNNNN() functions.

Submit your five code files for grading—two for the regular assignment and these three new files for the bonus.

Additional Specification

- You must comment code with **Javadoc** comments.
- **Use in-line comments** to indicate code blocks and describe decisions or complex expressions.
- **Do not use inline conditionals.**
- **Break and continue are generally considered bad form when learning to program**. As a result, you are **NOT** allowed to use them when creating your solution for this assignment. Their use can generally be avoided by combining if statements and writing better conditions on your while loops. You are allowed to use return within a loop inside a function.
- Put your **name, date, and tutorial** into the comments for the class's **Javadoc** for your TAs to identify your work.
- **Do not rename the provided files.** You must use exactly the requested filenames.

- **You should not import ANY libraries to complete the regular assignment. Using these could result in a grade of 0 for that portion of the assignment. Citing code from the internet to complete a function will also result in 0.**
- Use constants appropriately. Your TA may note one or two magic numbers as a comment, but more will result in lost marks.
- Do not change the provided code without discussion with the instructor. If a bug or something is broken, the instructor should be informed to fix this issue.
- You should not perform error checking in this assignment when writing your functions. The provided code should be designed to assume the rest of the program only inputs valid inputs when calling a function.
- **You must use loops for your win condition checking. Giant nest if/else chains without any looping will not be accepted for full marks.**

Grading

The total grade is out of 50. Bonus marks can reach up to 55 marks if completed.

Board.java (out of 25)

createBoard	2
rowCount	1
columnCount	1
canPlay	1
play	1
full	2
winInRow	3
winInColumn	3
winInDiagonalForwardSlash	4
winInDiagonalBackSlash	4
hint	3

BoardTest.java (out of 15)

createBoard	1
rowCount	0.5
columnCount	0.5
canPlay	0.5
play	0.5
full	1
winInRow	2
winInColumn	2
winInDiagonalForwardSlash	2
winInDiagonalBackSlash	2
hint	2
factorial	1

Gitlab Usage (out of 5)

A Gitlab account exists, a private project exists, at least one commit, small commits (regular commits for majority of this grade)

Style/Commenting (out of 5)

Name/Date/Tutorial, Functions commented, Javadoc, Inline commenting, doesn't use inline conditionals, doesn't use break, limited magic numbers, don't change function names, don't change filenames except Starter.java to Board.java, etc.

BONUS MARKS
You must have Gitlab 5/5 to get any bonus marks.
GameBonus.java (shouldn't have to touch this file after making a renamed copy)
BoardBonus.java (out 2.5)
BoardBonusTest.java (out of 2.5)

Invite your TA to your private!!!! csgit.ucalgary.ca project for the assignment

Submit the following using the Assignment 1 Dropbox in D2L

1. Board.java
2. BoardTest.java
3. Link to your private csgit.ucalgary.ca repository for TA
4. *GameBonus.java (only if Bonus was completed)*
5. *BoardBonus.java (only if Bonus was completed)*
6. *BoardBonusTest.java (only if Bonus was completed)*