CPSC 233: Introduction to Computer Science for Computer Science Majors 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 that you hand-in that are not your original work. You can put the citation into comments in your program. For example, if you find and use code found on a web site, 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. Citing sources avoids accusations of plagiarism and penalties for academic misconduct. **However, you may still get a low grade if you submit code that is not primarily developed by yourself. Cited material should never be used to complete core assignment specifications. You can and should verify and code you are concerned with your instructor/TA before submit.**
3. Discuss and share ideas with other programmers as much as you like, but make sure that when you write your code that 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, then this code is not yours.
4. **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 code itself, or modelling code after another student's algorithm. **You can not use (even with citation) another student's code.**
5. Making your code available, even passively, for others to copy, or potentially copy, is also plagiarism.
6. We will be looking 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/).
7. Remember, if you are having trouble with an assignment, it is always better to go to your TA and/or instructor to get help than it is to plagiarize. **The most common penalty is an F on a plagiarized assignment.**

Late Penalty

Late assignments will not be accepted.

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 correctness of portions of the assignment code created.

Technology

Java 16, 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 of 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 16.** You must use **Gitlab** hosted at **gitlab.ucalgary.ca** (not GitHub or another Git host). You must use **JUnit 5** and not other unit testing libraries.

Description

You are going to 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 TicTacToe.java file). This game must use the exact function names requested in this assignment document. Your job will be to complete these functions and then combine them to create a Tic Tac Toe game that uses these functions. While you are creating this game you will also be tasked with creating unit tests for these functions using **JUnit 5**. At the same time, you will be expected to store your code to a **private** repository at **gitlab.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 3 and take turns entering their symbol into the grid. The first one to 3 in a row (across, down, or diagonal) is the winner. As mentioned in class, this game is a solved game. That is, there is a known strategy such that a 'perfect' player can never lose the game. Two 'perfect' opposing players will always play to a draw in the game as well.

We will be implementing a flexible version of the game. The user will be able to play the base game on a 3 by 3 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 3 by 5, 4 by 3, or even 5 by 5 in size. A starter **TicTacToe.java** file will be provided with code to handle some of the requirements of the assignment. This code should not be changed unless communication with instructor clearly approves a change. There are about 12 different functions requiring completion for the Tic Tac Toe game to operate correctly. Your job will be to complete the

implementation of **TicTacToe.java** by implementing each of its methods to specifications indicated in the assignment. You will then need to create a **TicTacToeTest.java** JUnit 5 file to test these completed functions. You will also be expected to comment and document these two files and upload them to gitlab.ucalgary.ca and submit them in D2L.

Do not change the existing portions of the code. If you attempt the bonus, create a new TicTacToeBonus.java file and TicTacToeBonusTest.java file, in addition to the files for the non-bonus version. If you break your non-bonus program while doing the bonus, then you will lose those marks. YOU SHOULD NOT IMPORT ANY OTHER LIBRARIES FOR THIS ASSIGNMENT and you will get not credit for code copied from other places (cited code will get a grade of 0 as you are required to complete the required functions yourself).

Git Requirement: As we move through topics we will introduce Git as a version control system. For this assignment we will have the requirement that you upload your code to the universities gitlab.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, then you can start this process early and do all your code storage in Gitlab from the start.* If Git is new to you, then you will be taught it and we will expect your mostly complete project to get uploaded nearer to the end of your work on it.

The minimum expectation for Git usage is that when you submit your code that 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 gitlab.ucalgary.ca you will need a functional **Department of Computer Science account** username/password.

Program Coding Requirement:

The 11 functions you need to complete are as follows (all must be public static functions). I recommend first going through an 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 TicTacToe.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 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 size of first dimension of array, columns the second dimension.

Function Name: rowsIn

Parameters: board: 2D integer array

Return: integer

Assume board is valid 2D int array.

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

Function Name: columnsIn

Parameters: board: 2D integer array

Return: integer

Assume board is valid 2D int array.

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

Function Name: canPlay

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

Return: boolean

Assume board is 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).

Function Name: play

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

Return: nothing

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

Play, by assigning piece, in the location in the board at the indicated row/column.

Function Name: full

Parameters: board: 2D integer array

Return: boolean

Assume board is valid 2D int array.

Return true if board is filled with pieces that are all not EMPTY. Otherwise, false.

Function Name: winInRow

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

Return: boolean

Assume board is valid 2D int array, row is valid index in the board, piece is X==1/O==2

Look at indicated row at given index in board. If that row has at least 3 consecutive entries with given type of piece (X/O) (3 in a row XXX, OOO), then return true, otherwise false.

Function Name: winInColumn

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

Return: boolean

Assume board is valid 2D int array, column is valid index in the board, piece is X==1/O==2

Look at indicated column at given index in board. If that column has at least 3 consecutive entries with given type of piece (X/O) (3 in a row XXX, OOO), then return true, otherwise false.

Function Name: winInDiagonalBS

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

Return: boolean

Assume board is valid 2D int array, piece is X==1/O==2.

Look at all backward slash \ diagonals in the board. If any backward slash \ diagonals has at least 3 consecutive entries with given type of piece (X/O) (3 in a row XXX, OOO), then return true, otherwise false.

Function Name: winInDiagonalFS

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

Return: boolean

Assume board is valid 2D int array, piece is X==1/O==2.

Look at all forward slash / diagonals in the board. If any forward slash / diagonals has at least 3 consecutive entries with given type of piece (X/O) (3 in a row XXX, OOO), then return true, otherwise false.

Function Name: hint

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

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

Assume board is valid 2D int array, piece is X==1/O==2

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

```
For every row board
        For every column in 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 hint
                        Otherwise nobody has won game,
                                Remove the player's piece from the last played location
Default return -1 for both row and column
```

Unit Testing Requirement:

For the second part of the assignment we will be adding in Unit Testing. This will be accomplished via JUnit5. You will be required to create and submit a TicTacToeTest.java file for 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 TicTacToe.java file. So 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\assumptions 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.

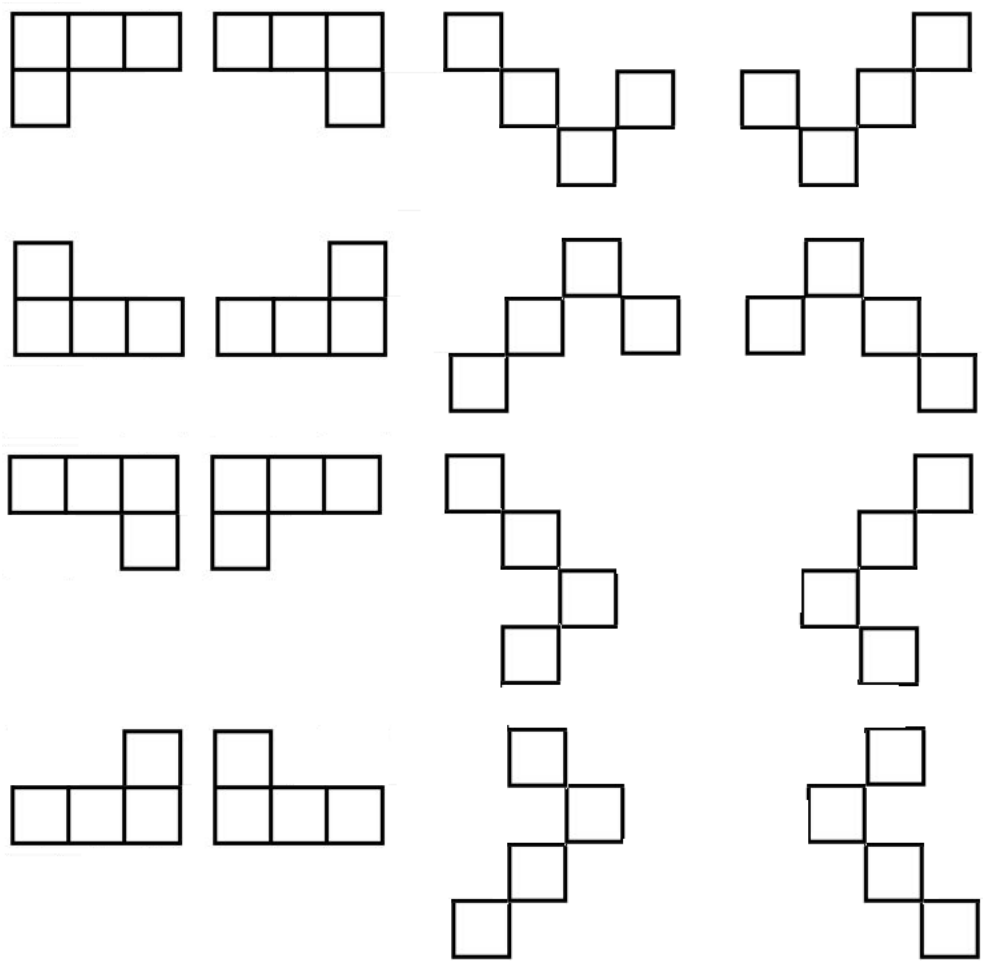
A couple simple functions will need 1 or 2 tests, but you will notice many will need more tests than 5. I don't want you to spend time making too many tests so I've limited the required test quantity for grading to 5 at most for each. (you are free to make more but the TAs will only mark the 5 you submit 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 5 tests for a function that are almost identical in purpose, then you 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. The goal should be one input, 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 TicTacToeBonus.java file and TicTacToeBonusTest.java file.

If a board size in includes a 4 or 5 as the length of one of the sides, then a win will no longer be just 3 pieces in a row. It will instead look like a tetris L Requiring this on a 3 by 3 board would not make any sense so 3x3 boards will retain the old 3 in a row. The new win types should look like the following.



Modify the game functions such that a win (for size 4/5 boards now) requires 3 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 3 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 4 code files for grading. 2 for regular assignment and these 2 new files for bonus.

Additional Specification

- You must comment your code with **Javadoc** comments.
- **Use in-line comments** to indicate blocks of code 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 to this assignment. In general, their use can be avoided by using a combination of 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 **Javadoc** of the class for your TAs to identify your work.
- **Do not rename the provided files.** You must you exactly the request filenames.
- **You should not import ANY libraries to complete the regular assignment. Using these could result in grade of 0 for that portion of assignment. Citing code from internet to complete a function will also result in 0.**
- **You will likely not need any constants for these functions. However, a constant for WIN_LENGTH=3 is likely useful.** 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 provided code without discussion with instructor. If there is 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 program is only inputting valid inputs.
- **You must use loops for you win condition checking. Giant nest if/else chains 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.

TicTacToe.java (out of 25)

createBoard	2
rowsIn	1
columnsIn	1
canPlay	1
play	1
full	2
winInRow	3
winInColumn	3
winInDiagonalFS	4
winInDiagonalBS	4
hint	3

TicTacToeTest.java (out of 15)

createBoard	1
rowsIn	0.5
columnsIn	0.5
canPlay	0.5
play	0.5
full	1
winInRow	2
winInColumn	2
winInDiagonalFS	2
winInDiagonalBS	2
hint	2
factorial	1

Gitlab Usage (out of 5)

Gitlab account exists, private project exists, at least 1 commit, small commits, regular commits
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, etc.

BONUS MARKS
TicTacToeBonus.java (out 2.5)
TicTacToeBonusTest.java (out of 2.5)

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

Submit the following using the Assignment 1 Dropbox in D2L

1. TicTacToe.java
2. TicTacToeTest.java
3. Name of your private gitlab.ucalgary.ca repository
4. *TicTacToeBonus.java (only if Bonus was completed)*
5. *TicTacToeBonusTest.java (only if Bonus was completed)*