Project Deliverable 2: Personal Software Process & Quality

PSP0 and Javadoc—50 points

Instructions:

- This is an Individual assignment - Collaboration is NOT allowed.
- Use of generative AI tools (such as ChatGPT, etc.) is NOT allowed.
- Late submissions will NOT be accepted (please note course policies in Syllabus).

Submission Instructions:

Submit a zipped folder named: `{YourASURiteUserID}-ProjectDeliverable2.zip`
(e.g., skbansa2-ProjectDeliverable1.zip)

This compressed folder should contain the following:

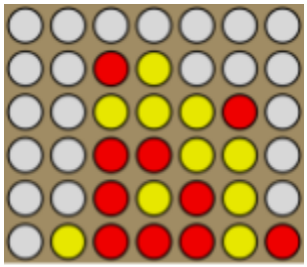
1. A folder called `core` containing `Connect4Logic.java` (Game Logic Module).
2. A folder called `ui` containing `Connect4TextConsole.java` (Console-based UI to test the game).
3. A folder called `docs` with Javadoc documentation files (`index.html` and all other supporting files such as `.css` and `.js` files generated by the tool). Submit the entire folder.
4. `ProjectDeliverable2.docx` (or pdf) with completed Time Log, Design Form, Defect Log, and Project Summary provided at the end of this assignment description.
    a. Make sure to provide responses to reflection questions listed in ProjectDeliverable2 file (this document).
5. A few screenshots showing test results of your working game.
6. A readme file (optional; submit if you have any special instructions for testing).

Grading Rubric:

- Working game—20 points
- Javadoc Documentation—5 points
- Test Results and Postmortem reflection question responses—5 points
- PSP process—20 points (5 points each for Time log, Defect log, Design form, Project Summary)

Connect 4 Game:

Connect 4 is a 2-player turn-based game played on a vertical board that has seven hollow columns and six rows. Each column has a hole in the upper part of the board, where pieces are introduced. There is a window for every square, so that pieces can be seen from both sides.



CONNECT 4 GAME IN PROGRESS (PUBLIC DOMAIN)

In short, it's a vertical board with 42 windows distributed in 6 rows and 7 columns. Both players have a set of 21 thin coin-like pieces; each player uses a different color. The board is empty at the start of the game.

The objective for either player is to make a straight line of four of their own pieces; the line can be vertical, horizontal or diagonal.

Reference: https://en.wikipedia.org/wiki/Connect_Four

Program Requirements:

Implement a Java-based Connect 4 game to be hosted in the ARENA Game system in the future. In this first version, build it as a simple console-based game played by 2 players—Player X and Player O. Be sure to do the following:

- Make use of good Object-Oriented design.
- Provide documentation using Javadoc and appropriate comments in your code.
- Generate HTML documentation using Javadoc tool.
- Create 2 packages: `core` and `ui`.
- Create a separate class for the game logic called `Connect4Logic.java` and place it in the `core` package.
- Create a separate class for the text-based UI called `Connect4TextConsole.java` and place it in the `ui` package.

Sample Output

Create a simple console-based UI as shown in the figures below.

```
Begin Game.
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
```

```
PlayerX—your turn. Choose a column number from 1-7.
4
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   | X |   |   |   |
```

```
PlayerO—your turn. Choose a column number from 1-7.
1
|   |   |   |   |   |   |   |
```

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

```
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   | O | X |
|   |   |   | O | O | X | X |
|   |   |   | O | X | X | X |
| O |   |   | X | X | O | O |
Player X Won the Game
```

General procedure:

- When the game starts, indicate that it is PlayerX's turn. Ask the player to choose a column number from 1-7.
- Check if the move is valid. If it's a valid move, then show the state of the grid by placing an X at the bottom-most empty row of the specified column. Next check for a "WIN" state (i.e., if PlayerX has an X in four consecutive horizontal, vertical, or diagonal cells of the grid). If it is a "WIN" state, inform that PlayerX is the winner and close the game. If not, continue the game.
- Indicate that it is PlayerO's turn. Ask the player to choose a column number from 1-7. Continue the game till one of the players wins or the game results in a draw.

Personal Process

Follow a good personal process for implementing the game.

- Please use the time log (provided at the end of this document) to keep track of time spent in each phase of development.
- Please use the defect log (provided at the end of this document) to keep track of defects found and fixed in each phase of development.
- When you are done implementing and testing your program, complete the Project Summary form to summarize your effort and defects. Also answer the reflection questions listed below in Postmortem phase.

Phases

Follow these steps in developing this game:

1. **Plan**—Understand the program specification and get any clarifications needed.
2. **Design**—Create a design in the form of a flow chart, break up of classes and methods, class diagram, pseudocode. Provide this design in the PSP design form provided later in the document. Keep track of time spent in this phase and log. Also keep track of any defects found and log them.
3. **Code**—Implement the program. Keep track of time spent in this phase and log. Also keep track of any defects found and log them.

4. **Test**—Test your program thoroughly and fix any bugs found. Keep track of time spent in this phase and log. Also keep track of any defects found and log them.
5. **Postmortem**—Complete the project summary form and answer the reflection questions.

PSP Time Recording Log

Date	Start	Stop	Interruption Time (mins)	Delta Time (mins)	Phase	Comments
3/20/24	7:30PM	8:30PM	10	50	Plan	Logged requirements
3/20/24	8:40PM	9:10PM	0	30	Design	Developed logic flow chart
3/21/24	6:30PM	7:05PM	0	35	Design	Developed class diagram
3/21/24	7:10PM	8:45PM	0	95	Code	Built classes and code structure
3/23/24	10:05AM	4:30PM	45+30+30	280	Code + Test	Completed program without full javadoc
3/23/24	5:15PM	7:00PM	0	105	Code + Test	Completed javadoc
3/23/24	7:00PM	7:30PM	0	30	Postmortem	Finished reflections and documentation of PSP.

- **Interruption time**: Record any interruption time that was not spent on the task. Write the reason for the interruption in the "Comment" column. If you have several interruptions, record them with plus signs (to remind you to total them).
- **Delta Time**: Enter the clock time you spent on the task, less the interrupt time.
- **Phase**: Enter the name or other designation of the programming phase being worked on. Example: Design or Code.
- **Comments**: Enter any other pertinent comments that might later remind you of any details or specifics regarding this activity.

PSP Defect Recording Log

Serial No.	Date	Defect Type No.	Defect Inject Phase	Defect Removal Phase	Fix Time (duration)	Fix Ref	Description
100	3/21/24	10	Design	Code	2		Had createPlayer methods in my Connect4Logic class diagram when the constructor for said class will already create the players.
101	3/21/24	10	Design	Code	2		Had gameID as an attribute in the Connect4TextConsole class diagram, but it shouldn't exist in that class.
102	3/21/24	10	Design	Code	2		Had createGameBoard method in my Connect4TextConsole class diagram when the constructor for said class will already create the game board.
103	3/21/24	10	Design	Code	2		Made the boardState attribute in my GameBoard class as type 'array' when it should be char[][]. I typed what I was thinking instead of the proper type.
104	3/23/24	100	Design	Code	5		GameBoard object should be related to the logic instead of the console where I originally had it. Using console specifically for visual and I/O. Made the Console's attribute a Connect4Logic object instead of the GameBoard.
105	3/23/24	100	Design	Code	5		There was no need for the methods displayPlayerXMove(), displayPlayerYMove(), displayWinState(),or displayDrawStat() because I can use only displayCurrentBoard() and the text displayed will be in the main() method of the Console class.
106	3/23/24	10	Design	Code	2		Updated names from PlayerY to PlayerO to match the requirements.
107	3/23/24	100	Design	Code	1		Added printCurrentBoard() method to the GameBoard class.
108	3/23/24	100	Design	Code	2		Added subtractOnePiece() method to Player and Logic classes
109	3/23/24	100	Design	Code	5		Added playerXTurn attribute to check who's turn it is and get and set methods for the attribute since it was missing.
110	3/23/24	100	Design	Code	1		startGame() and endGame() methods in Connect4Logic were not needed. Removed them.

111	3/23/24	80	Code	Test	10		Incorrect loop for adding the move to the current board state. Added the move to every row in the selected column instead of just the first empty one.
112	3/23/24	10	Design	Test	2		Moved pieceCount and subtractOnePiece from the Player class to the GameBoard class to call it during making a move.
113	3/23/24	80	Code	Test	15		When checking the vertical win state, I was incorrectly traversing the game board and I was getting an ArrayIndexOutOfBounds error.
114	3/23/24	70	Design	Code	5		Removed Player class from the game. They were not necessary in the way I set up the game to operate. They could be reintroduced in the future.
115	3/23/24	60	Code	Test	10		I forgot to handle for inputs other than integers so the program would crash if a string was inputted.

Instructions

- **Serial No.**: The unique id you associate with the defect; allows you to reference it later.
- **Defect Type No.**: The type number of the type—see the PSP Defect Type Standard table below and use your best judgement.
- **Defect Inject Phase**: Enter the phase (plan, design, code, etc.) when this defect was injected using your best judgment.
- **Defect Removal Phase**: Enter the phase during which you fixed the defect.
- **Fix Time**: Enter the amount of time that you took to find and fix the defect.
- **Fix Ref**: If you or someone else injected this defect while fixing another defect, record the number of the improperly fixed defect. If you cannot identify the defect number, enter an X. If it is not related to any other defect, enter N/A.
- **Description**: Write a succinct description of the defect that is clear enough to later remind you about the error and help you to remember why you made it.

PSP Defect Type Standard

Type Number	Type Name	Description
10	Documentation	Comments, messages
20	Syntax	Spelling, punctuation, typos, instruction formats
30	Build, Package	Change management, library, version control
40	Assignment	Declaration, duplicate names, scope, limits
50	Interface	Procedure calls and references, I/O, user formats
60	Checking	Error messages, inadequate checks
70	Data	Structure, content
80	Function	Logic, loops, recursion, computation, function defects
90	System	Configuration, timing, memory
100	Environment	Design, compile, test, or other support system problems

PSP0 Project Summary

Time in Phase

Phase	Actual time (in minutes)	% of total time
Planning	50	8
Design	65	10.4
Code + Test	480	76.8
~~Test~~		
Postmortem	30	4.8
TOTAL	**625**	**100**

Defects Injected

Phase	Actual defects (defect count)	% of total defects
Planning	0	0
Design	13	81.25
Code + Test	3	18.75
~~Test~~		
Postmortem	0	0
TOTAL	**16**	**100**

Final Summary

Metric	Value
Program Size (Lines of Code—LOC) [1]	223
Productivity (calculated by LOC/Hour)	21.41
Defect Rate (calculated by Defects/KLOC) [2]	71.75

Reflection Questions

1. In which Phase did you spend most of your effort? (look at the time spent in different phases of this assignment to answer this question)

   Coding and Testing.

2. In which Phase did you introduce the greatest number of defects?

   Design

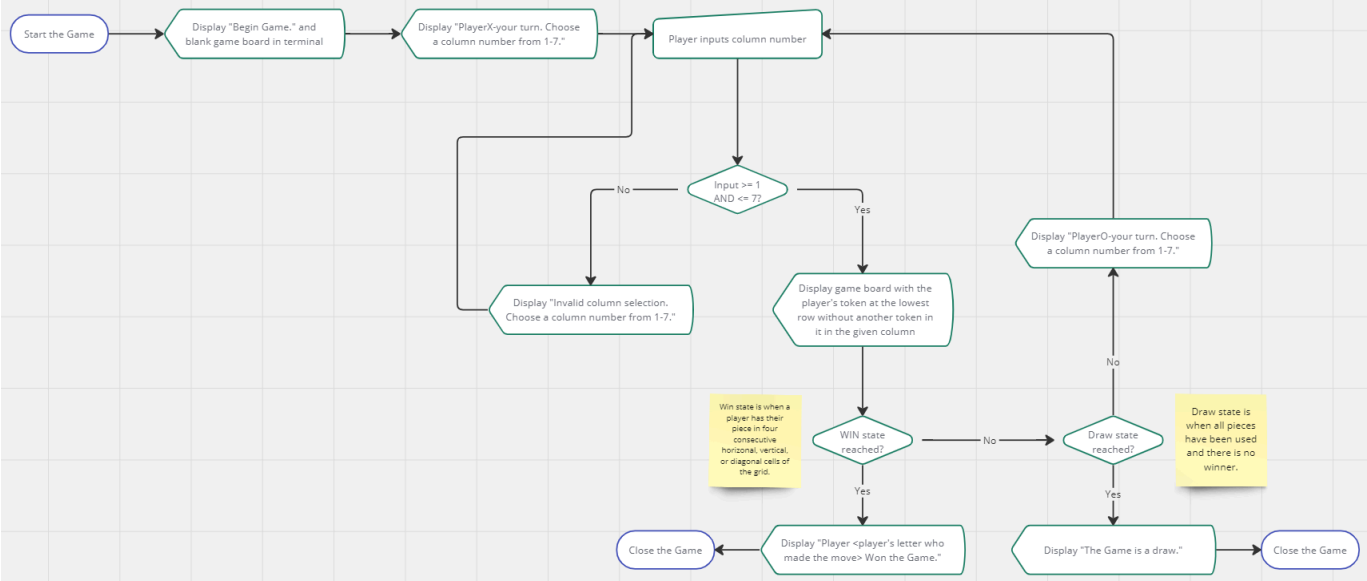
3. Did you find it useful to follow a systematic process and track your effort and defects?

   I found it useful, but I struggled with documenting the defects properly and accurately. I think the biggest struggle I had is that I didn't properly visualize the program in the design phase which made most of my corrections and work sit in the coding and testing phase.

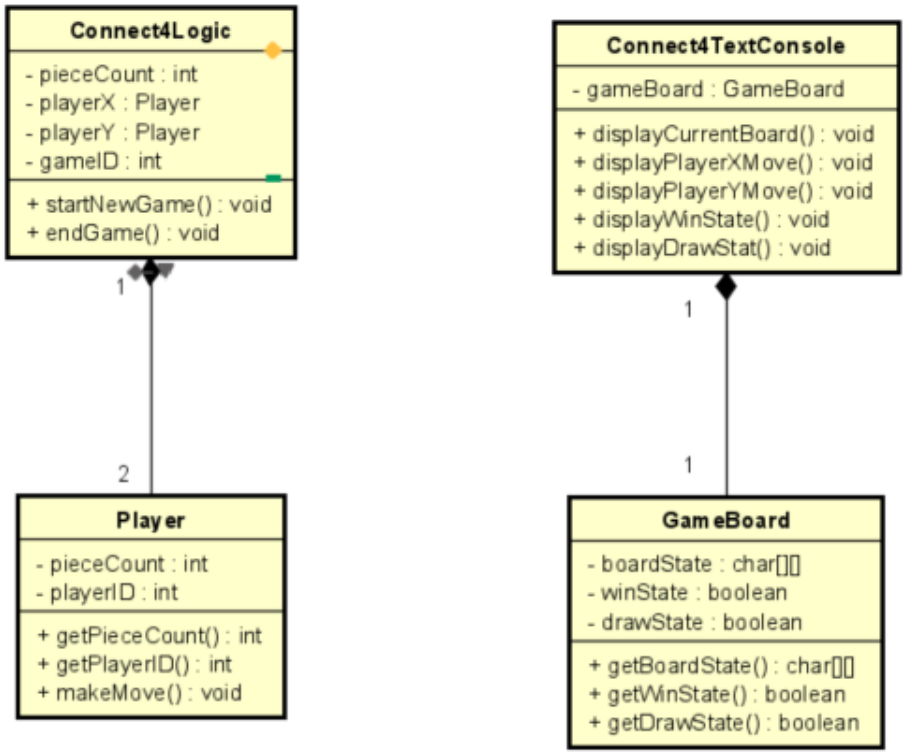
[1] LOC stands for lines of code.
[2] KLOC stands for kilo lines of code (1000 lines)

PSP Design Form

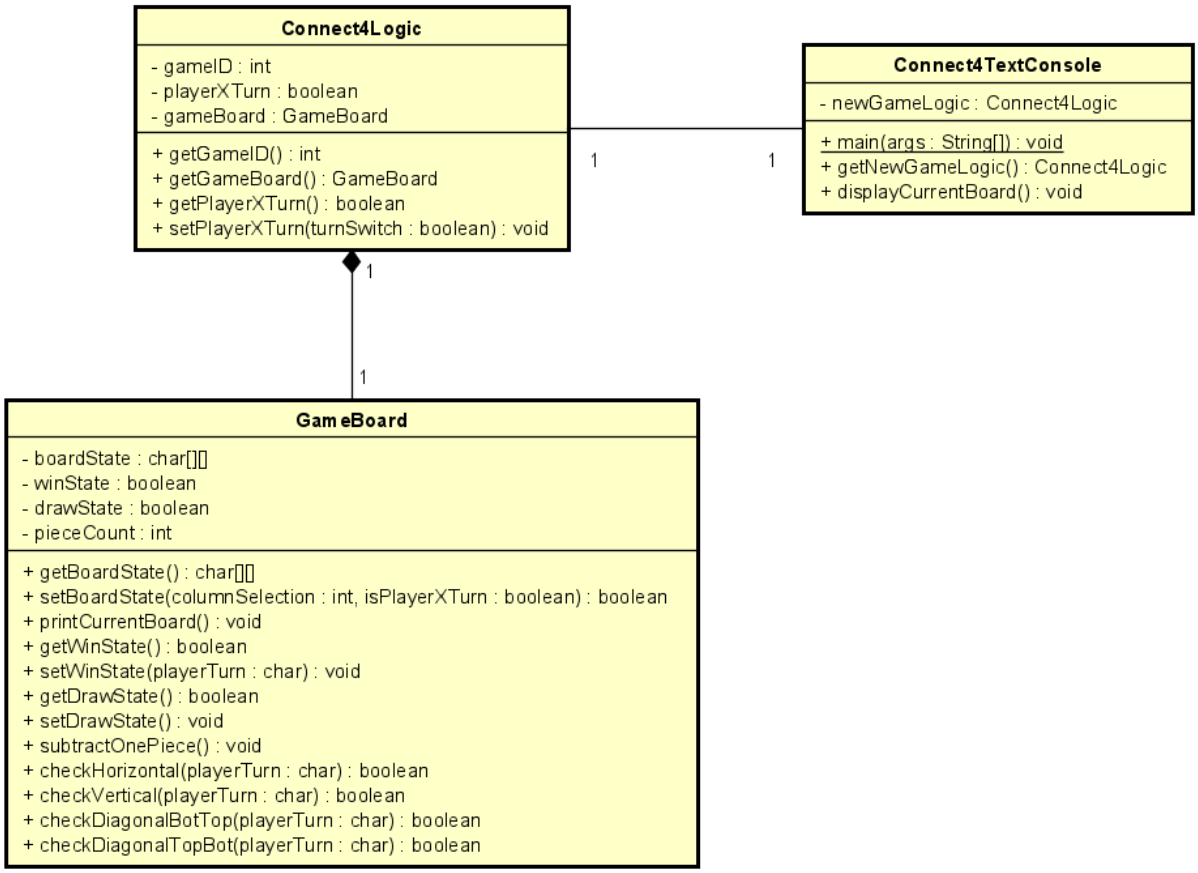
Use this form to record whatever you do during the design phase of development. Include notes, class diagrams, flowcharts, formal design notation, or anything else you consider to be part of designing a solution that happens BEFORE you write program source code. Attach additional pages if necessary.



First State:



Final State:


Connect4Logic
- gameID : int
- playerXTurn : boolean
- gameBoard : GameBoard
+ getGameID() : int
+ getGameBoard() : GameBoard
+ getPlayerXTurn() : boolean
+ setPlayerXTurn(turnSwitch : boolean) : void
Connect4TextConsole
- newGameLogic : Connect4Logic
+ main(args : String[]) : void
+ getNewGameLogic() : Connect4Logic
+ displayCurrentBoard() : void
1
1
1
1
GameBoard
- boardState : char[][]
- winState : boolean
- drawState : boolean
- pieceCount : int
+ getBoardState() : char[][]
+ setBoardState(columnSelection : int, isPlayerXTurn : boolean) : boolean
+ printCurrentBoard() : void
+ getWinState() : boolean
+ setWinState(playerTurn : char) : void
+ getDrawState() : boolean
+ setDrawState() : void
+ subtractOnePiece() : void
+ checkHorizontal(playerTurn : char) : boolean
+ checkVertical(playerTurn : char) : boolean
+ checkDiagonalBotTop(playerTurn : char) : boolean
+ checkDiagonalTopBot(playerTurn : char) : boolean