Project Deliverable 3: Personal Software Process & Quality

PSP1, Javadoc, Exception Handling—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}-ProjectDeliverable3.zip`
(e.g., skbansa2-ProjectDeliverable3.zip)

This compressed folder should contain the following:

1. A folder called `core` containing:
    a. `Connect4Logic.java` (Game Logic Module)
    b. `Connect4ComputerPlayer.java` (logic to play against computer; generate computer moves)
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. `ProjectDeliverable3.docx` (or pdf) with Completed Time Log, Estimation worksheet, Design form, Defect Log, and Project Summary provided at the end of this assignment description.
    a. Make sure to provide responses to the reflection questions listed in ProjectDeliverable3 file (this document).
5. A few screen shots showing test results of your working game.
6. A readme file (optional; submit if you have any special instructions for testing).

Grading Rubric:

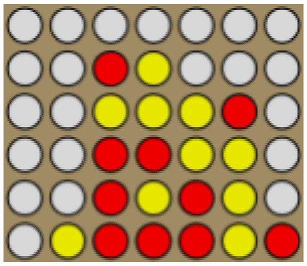
Working game—20 points
Javadoc Documentation—5 points
Exception Handling—5 points

Test Results and Postmortem reflection question responses—5 points
PSP process—15 points (3 points each for Time log, Defect log, Estimating Worksheet, 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.

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.



Connect 4 game in progress (public domain)

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:

To the previously developed Java-based Connect4 game, add a module to "play against the computer". Create a separate class called `Connect4ComputerPlayer.java` in the `core` package that generates the moves for the computer player. The logic to automatically generate computer moves does NOT have to be a sophisticated AI algorithm. A naïve algorithm to generate the moves is sufficient for this assignment.

- Continue to make use of good Object-Oriented design
- Provide documentation using Javadoc and appropriate comments in your code.
- Generate HTML documentation using Javadoc tool
- Make sure you provide appropriate Exception Handling throughout the program (in the previously created classes as well)

Sample Output

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

```
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |

Begin Game. Enter ‘P’ if you want to play against another player; enter ‘C’ to
play against computer.

C
```

```
Start game against computer.

It is your turn. Choose a column number from 1-7.

4

|   |   |   |   |   |   |   |

|   |   |   |   |   |   |   |

|   |   |   |   |   |   |   |

|   |   |   |   |   |   |   |

|   |   |   |   |   |   |   |

|   |   |   | X |   |   |   |
```

and so on…

Personal Process:

Follow a good personal process for implementing this game. You will be using PSP1 in this assignment. So, in addition to *tracking* your effort and defects you will have to *estimate* the effort

- for the “play against computer” module as well as
- adding exception handling to the previously created classes.

PSP Forms

- Please use the **estimating worksheet** contained herein to estimate how much time, and how big your program might be.
- Please include in the **design form** any materials you create during your design process. It’s at the end of this document.
- 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.

Phases

Follow these steps in developing the game:

Plan—understand the program specification and get any clarifications needed.

1. **Estimate** the **time** you are expecting to spend on the new module(s) to be added.
2. **Estimate** the **size** of the program (only for **new code** that you will be adding).
3. Enter this information in the **estimation columns** of the Project Summary form. Use your best guess based on your previous programming experience. There is no penalty for having an estimate that is not close to the actual. It takes practice to get better at estimation.
4. Use the provided **estimating worksheet**.

Design—create a design (for the new modules being added) in the form of flow charts, breakdowns of classes and methods, class diagrams or pseudocode. Provide this design in the PSP **design form** provided at the end of this document. Keep track of time spent in this phase and log. Also keep track of any defects found and log them.

Code—implement the program. Keep track of **time** spent in this phase and log. Also keep track of any **defects** found and log them.

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.

Postmortem—complete the actual and to date columns of the project summary form and answer the reflection questions.

Estimating Worksheet

PSP1 Informal Size Estimating Procedure

1. Study the requirements.
2. Sketch out a crude design.
3. Decompose the design into “estimatable” chunks.
4. Make a size estimate for each chunk, using a combination of:
    a. visualization.
    b. recollection of similar chunks that you’ve previously written
    c. intuition.
5. Add the sizes of the individual chunks to get a total.

Conceptual Design (sketch your high-level design here)

Connect 4 Text Console

- Calls Connect4Logic
  - If play against computer chosen
    - call Logic w/ 'c'
  - Else
    - call logic w/o parameters

Connect4Logic

- Create second constructor w/ parameter for computer player
  - creates GameBoard object w/ parameter as well

GameBoard

- create second constructor w/ parameter for comp. player
  - creates computer player class

connect4 computer player

- moves piece to first available spot when it's turn
  - searchs bottom to top, left to right

Module Estimates

Module Description	Estimated Size
Parameter and constructor additions	20 mins; 20 LOC
Move logic for Connect4ComputerPlayer	45 mins; 30 LOC
Update the GameBoard with the computer move	30 mins; 25 LOC
Total Estimated Size:	**95 mins; 75 LOC**

PSP Time Recording Log

Date	Start	Stop	Interruption Time	Delta Time	Phase	Comments
3/28/24	4:00PM	4:33PM	5	28	Plan	Created a rough plan and estimated size.
3/28/24	6:30PM	7:00PM	0	30	Design	Updated flow chart and class diagram from deliverable 2

3/28/24	7:10PM	9:45PM	30	125	Code + Test	Created code and tested the code for the new additions
3/29/24	10:30AM	11:00AM				Finished postmortem work

- **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	3/28/24	10	Design	Code	1		Duplicated a display message in the flowchart
101	3/28/24	80	Code	Test	10		When the computer made its move, the loop I set up to search the board for an empty spot filled all empty spots and automatically made the computer win.
102	3/28/24	60	Code	Test	10		I forgot to include an exception handle for the user input of the game type so the program could fail.

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

PSP1 Project Summary

Time in Phase

Phase	Estimated time (in minutes)	Actual time (in minutes)	To Date time	% of total time to Date
Planning	20	28	78	9.3
Design	45	30	95	11.3
Code + Test	200	125	605	72.2
Postmortem	25	30	60	7.2
TOTAL	**290**		**838**	**100**

Defects Injected

Phase	Estimated Defects	Actual Defects	To Date defects	% of total to Date
Planning	—	0	0	0
Design	—	1	14	73.7%
Code + Test	—	2	5	26.3%
	—			
Postmortem	—	0	0	0
TOTAL	—	**3**	**19**	**100**

Final Summary

Metric	Estimated	Actual	To Date
Program Size (Lines of Code—LOC) [1]	75	102	325
Productivity (calculated by LOC/Hour)	48	35.2	38.8
Defect Rate (calculated by Defects/KLOC) [2]	—	29.4	58.5

Reflection Questions

1. How good was your time estimate for various phases of software development?

   Overall, my estimations were pretty good. I completed this deliverable faster than I expected. I think based on how long the last deliverable took, it made it think

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

that I would take longer.

2. How good was your program size estimate, i.e., was it close to actual?

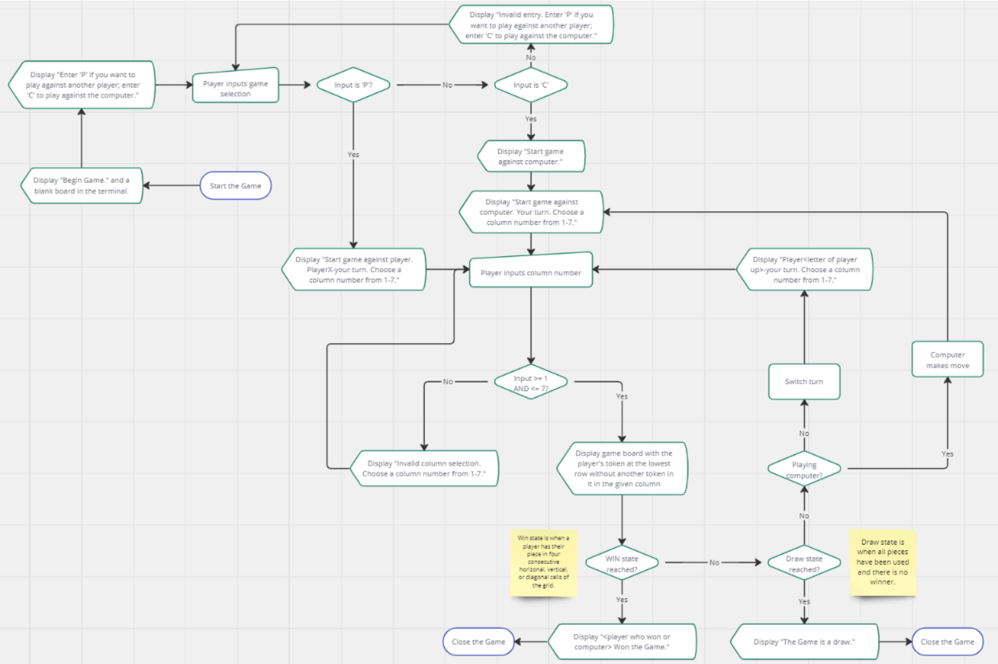
   It was okay. I was within ~25 LOC from a program size estimation perspective vs. the actual. I noticed this time around that I wasn't quite sure how many lines of code it takes me to write certain aspects of the programs so I'm working on that.

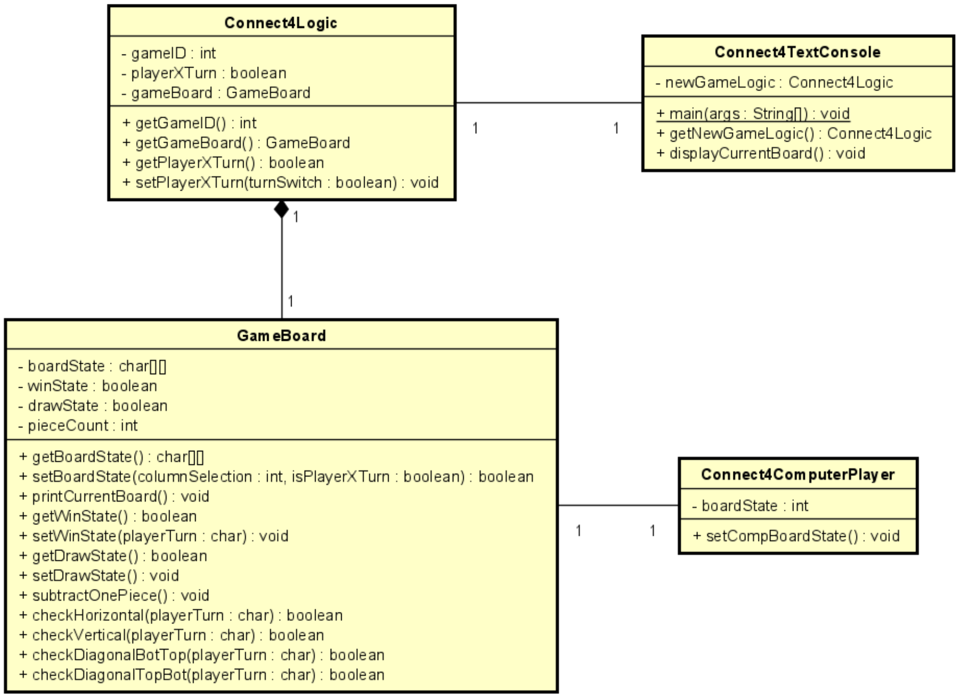
3. In which phase did you introduce the most number of defects?

   In the code phase

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.


Start the Game
Display "Begin Game." and a blank board in the terminal.
Display "Enter 'P' if you want to play against another player; enter 'C' to play against the computer."
Player inputs game selection
Input is 'P'?
No
Input is 'C'
No
Display "Invalid entry. Enter 'P' if you want to play against another player; enter 'C' to play against the computer."
Yes
Display "Start game against computer."
Display "Start game against computer. Your turn. Choose a column number from 1-7."
Yes
Display "Start game against player. PlayerX-your turn. Choose a column number from 1-7."
Player inputs column number
Display "Player<letter of player up>-your turn. Choose a column number from 1-7."
Input >= 1 AND <= 7?
No
Display "Invalid column selection. Choose a column number from 1-7."
Yes
Display game board with the player's token at the lowest row without another token in it in the given column
Win state is when a player has their piece in four consecutive horizonal, vertical, or diagonal cells of the grid.
WIN state reached?
No
Draw state reached?
Draw state is when all pieces have been used and there is no winner.
No
Playing computer?
No
Switch turn
Yes
Computer makes move
Yes
Display "<player who won or computer> Won the Game."
Close the Game
Yes
Display "The Game is a draw."
Close the Game


Connect4Logic
- gameID : int
- playerXTurn : boolean
- gameBoard : GameBoard
+ getGameID() : int
+ getGameBoard() : GameBoard
+ getPlayerXTurn() : boolean
+ setPlayerXTurn(turnSwitch : boolean) : void
1
1
Connect4TextConsole
- newGameLogic : Connect4Logic
+ main(args : String[]) : void
+ getNewGameLogic() : Connect4Logic
+ displayCurrentBoard() : void
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
1
1
Connect4ComputerPlayer
- boardState : int
+ setCompBoardState() : void