Project Deliverable 2
Personal Software Process (PSP0 and JavaDoc)
Points: 50

Instructions:

This assignment has to be completed by each student individually. NO COLLABORATION IS ALLOWED.
Submit the following: YourASURiteID-ProjectDeliverable2.zip This compressed folder should contain the following files:

1. Folder called core containing Connect4.java (Game Logic Module)
2. Folder called ui containing Connect4TextConsole.java (Console-based UI to test the game)
3. 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 TimeLog, Design form, DefectLog, and ProjectSummary provided at the end of this assignment description.
5. Make sure to provide responses to reflection questions listed in ProjectDeliverable2 file (this document).
6. A few screen shots showing test results of your working game and answers to reflection questions written inline in this document
7. Readme file (optional: submit if you have any special instructions for testing)

Connect4 Game:
Connect4 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 pieces (like coins); each of them uses a different color. The board is empty at the start of the game. The aim for both players is to make a straight line of four own pieces; the line can be vertical, horizontal or diagonal.

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

Program Requirements:
Implement a Java-based Connect4 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. Ensure that 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 Connect4.java and place it in core package
- Create a separate class for the text-based UI called Connect4TextConsole.java and place it in ui package

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
```

- When the game starts, indicate that it is PlayerX's turn. Ask the player to choose column number from 1-7.
- Check if the move is valid. If 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 "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 column number from 1-7. Continue the game till one of them wins or game results in a draw.

Personal Process:

Follow a good personal process for implementing this 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 Post-mortem phase.

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 bugs found. Keep track of time spent in this phase and log. Also keep track of any defects found and log them.

5. Post Mortem – **Complete the project summary form and answer the following questions**.

    i. 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)

    ii. In which Phase did you introduce most number of defects?

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

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)

PSP Time Recording Log

Date	Start	Stop	Interruption Time	Delta Time	Phase	Comments

- **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 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.

PSP Defect Recording Log

Sl. No.	Date	Defect Type	Defect Inject Phase	Defect Removal Phase	Fix Time	Fix Ref	Description

Instructions

- **Defect Type**: Use your best judgment in selecting which defect type applies from list provided below.
- **Defect Inject Phase**: Enter the phase 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 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, pointers, 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 (minutes)	Actual Time (in minutes)	% of Total
Planning		
Design		
Code		
Test		
Postmortem		
TOTAL		

Defects Injected	Actual Number of Defects	% of Total
Planning		
Design		
Code		
Test		
Postmortem		
TOTAL		

SUMMARY

	Actual
Program Size (Lines of Code - LOC)	
Productivity (calculated) LOC/Hour	
Defect Rate (calculated) Defects/KLOC	

- LOC is lines of Code
- KLOC is Kilo lines of code (i.e. 1000 lines)