Project Deliverable 4: Personal Software Process

PSP2, Exception Handling, GUI Development—50 points

This assignment must be done by each student individually. ***No collaboration is allowed.***

Submission Instructions:

Submit a zipped folder named: `{YourASURiteUserID}-ProjectDeliverable4.zip`

This compressed folder should contain the following:

1. A folder called `core` containing:
    a. `Connect4.java` (game logic module)
    b. `Connect4ComputerPlayer.java` (logic to play against computer; generate computer moves)
2. A folder called `ui` containing
    a. `Connect4TextConsole.java` (console-based UI to test the game)
    b. `Connect4GUI.java` (graphical user interface for 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. `ProjectDeliverable4.docx` (or pdf) with Completed Time Log, Estimation worksheet, Design form, Defect Log, Personal Code Review and Project Summary provided at the end of this assignment description.
    a. Make sure to provide responses to the reflection questions listed in ProjectDeliverable4 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 and GUI—15 points

Javadoc and Exception Handling—5 points

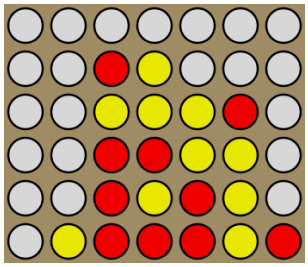
Code Review report—5 points

Test Results and Postmortem reflection question responses—5 points

PSP process—15 points
Time log (4), Defect log (3), Estimating Worksheet (2), Design form (2), Project Summary (4)

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:

To the previously developed Java-based Connect4 game, add a module to provide a graphical user interface (GUI) using JavaFX. Create a separate class called `Connect4Gui.java` in the `ui` package that generates the GUI.

- 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 (using try-catch blocks) throughout the program
- You can decide on the layout of the user interface and game board.
- At the start of the program,
    - Ask the user if they would like a GUI or console-based UI. Then show the corresponding UI.
    - Next, ask if they want to play against a computer or another player and respond accordingly.

Personal Process:

Follow a good personal process for implementing this game. You will be using PSP2 in this assignment. So, in addition to *tracking* your effort and defects you will have to *estimate* the effort and *defects* for the GUI module—recall that in PSP1 you only estimated the effort, but not the defects. Don't forget to provide exception handling routines and conduct a personal code review.

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 **code review checklist** contained herein to statically analyze your code for common mistakes.
- 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 GUI development task.
2. **Estimate** the **defects** you are expecting to inject in each phase you go through while developing the GUI
3. **Estimate** the **size** of the program (only for **new code** that you will be adding).
4. 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.
5. 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.

Code Review—use the **code** review **guidelines** provided later in the document to conduct a personal review of your code and fix any issues found. Provide comments in the checklist about your findings. There should be a minimum of 4 comments.

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

PSP2 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)

Module Estimates

Module Description	Estimated Size
Total Estimated Size:	

Code Review Checklist

Add comments. Boxes are checkable (☐+👆= ☒).

Specification / Design

☐ Is the functionality described in the specification fully implemented by the code?
☐ Is there any excess functionality in the code but not described in the specification?

Initialization and Declarations

☐ Are all local and global variables initialized before use?
☐ Are variables and class members of the correct type and appropriate mode
☐ Are variables declared in the proper scope?
☐ Is a constructor called when a new object is desired?
☐ Are all needed import statements included?
☐ Names are simple and if possible short
☐ There are no usages of “magic numbers”

General

☐ Code is easy to understand
☐ Variable and method names are spelt correctly
☐ There is no dead code (i.e., code inaccessible at runtime)
☐ Code is not repeated or duplicated
☐ No empty blocks of code

Method Calls

☐ Are parameters presented in the correct order?
☐ Are parameters of the proper type for the method being called?
☐ Is the correct method being called, or should it be a different method with a similar name?
☐ Are method return values used properly? Are they being cast to the needed type?

Arrays/Data structures

☐ Are there any off-by-one errors in array indexing?
☐ Can array indexes ever go out-of-bounds?
☐ Is a constructor called when a new array item is desired?
☐ Are your data structures ideal?
☐ Collections are initialized with a specific estimated capacity

Object

☐ Are all objects (including Strings) compared with `equals` and not `==`?
☐ No object exists longer than necessary
☐ Files/Sockets and other resources if used are properly closed even if an exception occurs when using them

Output Format

☐ Are there any spelling or grammatical errors in the displayed output?
☐ Is the output formatted correctly and consistently in terms of line stepping and spacing?

Computation, Comparisons and Assignments

☐ Check order of
    ☐ computation/evaluation
    ☐ operator precedence and
    ☐ parenthesizing
☐ Can the denominator of any divisions ever be zero?
☐ Is integer arithmetic, especially division, ever used inappropriately, causing unexpected truncation/rounding?
☐ Check each condition to be sure the proper relational and conditional operators are used.
☐ If the test is an error-check, can the error condition actually be legitimate in some cases?
☐ Does the code rely on any implicit type conversions?

Exceptions

☐ Are all relevant exceptions caught?
☐ Is the appropriate action taken for each catch block?

☐ Are all appropriate exceptions thrown?
☐ Are catch clauses fine-grained and catching specific exceptions?

Flow of Control

☐ In switch statements, is every case terminated by `break` or `return`?
☐ Do all switch statements have a default branch?
☐ Check that nested if statements don't have “dangling else” problems.
☐ Are all loops correctly formed, with the appropriate initialization, increment and termination expressions?
☐ Are open-close parentheses and brace pairs properly situated and matched?

Files

☐ Are all files properly declared and opened?
☐ Are all files closed properly, even in the case of an error?
☐ Are EOF conditions detected and handled correctly?
☐ Are all file exceptions caught?

Documentation

☐ Methods commented in clear language
☐ Most comments should describe rationale or reasons (the *why*); fewer should describe the *what*; few should describe *how*.
☐ Are there any out-of-date comments that no longer match their associated code?
☐ All public methods/interfaces/contracts are commented describing usage
☐ All edge cases are described in comments
☐ All unusual behavior or edge case handling is commented
☐ Data structures and units of measurement are explained

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 Defect Recording Log

Serial No.	Date	Defect Type No.	Defect Inject Phase	Defect Removal Phase	Fix Time (duration)	Fix Ref	Description

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

PSP2 Project Summary

Time in Phase

Phase	Estimated time (in minutes)	Actual time (in minutes)	To Date	% of total time to Date
Planning				
Design				
Code				
Code Review				
Test				
Postmortem				
TOTAL				

Defects Injected

Phase	Estimated Defects	Actual Defects	To Date	% of total to Date
Planning				
Design				
Code				
Code Review				
Test				
Postmortem				
TOTAL				

Final Summary

Metric	Estimated	Actual	To Date
Program Size (Lines of Code—LOC) [1]			
Productivity (calculated by LOC/Hour)			
Defect Rate (calculated by Defects/KLOC) [2]			

Reflection Questions

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

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

3. How many issues did you find in your code during code review?

[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.