Project Deliverable 3: Personal Software Process & Quality

PSP1, Javadoc, Exception Handling—50 points

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

Submission Instructions:

Submit a zipped folder named: `{YourASURiteUserID}-ProjectDeliverable3.zip`
(e.g., skbansa2-ProjectDeliverable3.zip)

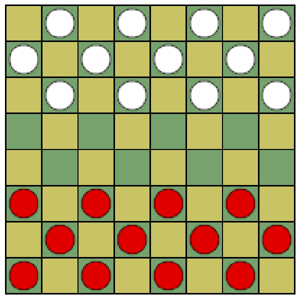
This compressed folder should contain the following:

1. A folder called `core` containing:
    a. `CheckersLogic.java` (Game Logic Module)
    b. `CheckersComputerPlayer.java` (logic to play against computer; generate computer moves)
2. A folder called `ui` containing `CheckersTextConsole.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)

Checkers Game:


CHECKERS BOARD WHEN GAME STARTS (PUBLIC DOMAIN)

Checkers is a strategy board game for two players which involve diagonal moves of uniform game pieces and mandatory captures by jumping over opponent pieces. It is played by two opponents, on opposite sides of the 8x8 checkered gameboard. One player has the dark pieces (or 'x' tokens); the other has the light pieces (or 'o' tokens). Each player has 12 pieces. Players alternate turns. A player may not move an opponent's piece. A move consists of moving a piece diagonally to an adjacent unoccupied square. If the adjacent square contains an opponent's piece, and the square immediately beyond it is vacant, the piece may be captured (and removed from the game) by jumping over it.

Only the dark squares of the checkered board are used. A piece may move only diagonally into an unoccupied square. The player without pieces remaining, or who cannot move due to being blocked, loses the game. Pieces move one step diagonally forwards, and capture an opponent's piece by moving two consecutive steps in the same line, jumping over the piece on the first step. Multiple enemy pieces can be captured in a single turn provided this is done by successive jumps made by a single piece; the jumps do not need to be in the same line and may "zigzag" (change diagonal direction). Pieces can move/jump only in forward direction.

Aim of the game: is to capture all the opponent's pieces or render them unable to move.

How the game ends:

- The first player to lose all of his or her pieces loses the game.

- If a player is put in a position where they cannot move, they lose.

Reference: https://simple.wikipedia.org/wiki/Checkers

Program Requirements:

To the previously developed Java-based game, add a module to "play against the computer". Create a separate class called `CheckersComputerPlayer.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.

```
8 | _ | o | _ | o | _ | o | _ | o |
7 | o | _ | o | _ | o | _ | o | _ |
6 | _ | o | _ | o | _ | o | _ | o |
5 | _ | _ | _ | _ | _ | _ | _ | _ |
4 | _ | _ | _ | _ | _ | _ | _ | _ |
3 | x | _ | x | _ | x | _ | x | _ |
2 | _ | x | _ | x | _ | x | _ | x |
1 | x | _ | x | _ | x | _ | x | _ |
    a   b   c   d   e   f   g   h


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

```
8 | _ | o | _ | o | _ | o | _ | o |
7 | o | _ | o | _ | o | _ | o | _ |
6 | _ | o | _ | o | _ | o | _ | o |
5 | _ | _ | _ | _ | _ | _ | _ | _ |
4 | _ | _ | _ | _ | _ | _ | _ | _ |
3 | x | _ | x | _ | x | _ | x | _ |
2 | _ | x | _ | x | _ | x | _ | x |
1 | x | _ | x | _ | x | _ | x | _ |
    a   b   c   d   e   f   g   h


Start game against computer.
You are Player X. It is your turn.
Choose a cell position of piece to be moved and the new position. e.g., 3a-4b
3g-4h
```

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)

Module Estimates

Module Description	Estimated Size
Total Estimated Size:	

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

PSP1 Project Summary

Time in Phase

Phase	Estimated time (in minutes)	Actual time (in minutes)	To Date time	% of total time to Date
Planning			50	(50/785)*100
Design			100	(100/785)*100
Code			500	
Test			90	
Postmortem			45	
TOTAL			**785**	**100**

Defects Injected

Phase	Estimated Defects	Actual Defects	To Date defects	% of total to Date
Planning	———		Week2 + Week3	
Design	———			
Code	———			
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 estimate for various phases of software development?

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

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

[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.