Project Deliverable 2: Personal Software Process & Quality

PSP0 and Javadoc—50 points

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

Submission Instructions:

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

This compressed folder should contain the following:

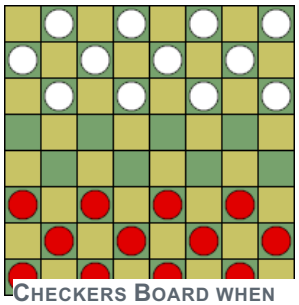
1. A folder called `core` containing `CheckersLogic.java` (Game Logic Module).
2. A folder called `ui` containing `CheckersTextConsole.java` (Console-based UI to test the game).
3. A folder called `docs` with Javadoc documentation files (`index.html` and all other supporting files such as `.css` and `.js` files generated by the tool). Submit the entire folder.
4. `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 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
- 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)

Checkers Game:

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



CHECKERS BOARD WHEN GAME STARTS (PUBLIC DOMAIN)

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:

Implement a Java-based Checkers 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 `CheckersLogic.java` and place it in the `core` package.
- Create a separate class for the text-based UI called `CheckersTextConsole.java` and place it in the `ui` package.

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. Player X – your turn.
```

Choose a cell position of piece to be moved and the new position. e.g., 3a-4b

3g-4h

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

PlayerO – your turn.

Choose a cell position of piece to be moved and the new position. e.g., 3a-4b

6f-5e

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

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

Player X Won the Game

Or

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

General procedure:

- When the game starts, indicate that it is Player X's turn. Ask the player to choose a piece to move by indicating the cell# followed by the new position, e,g, 3a-4b.
- Check if the move is valid. If valid move, then show the state of the grid by placing the piece in the correct position (after capturing opponent piece, where applicable). Next check for "WIN/LOSE" state (i.e., if one of the players has no pieces left, or if one of the players is unable to make any legal move). Continue the game if valid moves are possible.
- Next, indicate that it is Player O's turn. Ask the player to choose a piece to move by indicating the cell# followed by the new position, e.g., 3a-4b.
- A player wins the game when the opponent cannot make a move. This could be the case because all of the opponent's pieces have been captured or because all of opponent's pieces are blocked in.
- Only allow moving forward.
- You don't have to implement 'Kinging' or 'Crowning' or 'Double piece' features.

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	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

PSP0 Project Summary

Time in Phase

Phase	Actual time (in minutes)	% of total time
Planning		
Design		
Code		
Test		
Postmortem		
TOTAL		

Defects Injected

Phase	Actual defects (defect count)	% of total defects
Planning		
Design		
Code		
Test		
Postmortem		
TOTAL		

Final Summary

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

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)

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

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