Project Deliverable 4
Personal Software Process & Quality
(PSP2, Exception Handling, GUI development)
Points: 50

Instructions:

This assignment has to be completed by each student individually. NO COLLABORATION IS ALLOWED.

Submit the following: YourASURiteID-ProjectDeliverable4.zip This compressed folder should contain the following files:

1. Connect4.java (Game Logic Module)
2. Connect4TextConsole.java (Console-based UI to test the game)
3. Connect4ComputerPlayer.java (logic to play against computer; generate computer moves)
4. Connect4GUI.java (graphical user interface for the game)
5. JavaDoc documentation files (index.html and all other supporting files such as .css and .js files generated by the tool). Submit the entire folder.
6. Completed TimeLog, Estimation worksheet, Design form, DefectLog, Personal Code Review, and ProjectSummary provided at the end of this assignment description
7. A few screen shots showing test results of your working game and answers to reflection questions written inline in this document
8. 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:

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 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 GUI or console-based UI. Then show the corresponding UI as per user preference. Next ask user if they want to play against computer or another player and then continue with the game.

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, provide exception handling routines, and conduct a personal code review.

- 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.
   - estimate the time you are expecting to spend on the GUI development task.
   - estimate the defects you are expecting to inject in each phase for GUI development task.
   - estimate the size of the program (only for new code that you will be adding)
   - enter this information in the Estimation columns of the Project Summary form. Use your best guess based on your previous programming experience. You will not be penalized for not having an estimate that is close to the actual. It takes practice to get better at estimation.
   - use the provided estimating worksheet.

2. **Design** – create a design (for the new modules being added) 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. **Code Review** – use the code review guidelines/checklist 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.

5. **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.

6. **Post Mortem** – Complete the actual columns of the project summary form and answer the following questions.

      i.   How good was your time and defect estimate for various phases of software development? Not very good. I spent most of my time in the code phase because I've never used Javafx before and most of my defects were because of that.

      ii.   How good was your program size estimate, i.e., was it close to actual?
I underestimated by 87 LOC. I think this is pretty good for never using JavaFX before.

      iii.   How many issues did you find in your code during code review?
I found one in the files section of the checklist. I forgot to close a file. I found the checklist helpful.

Grading Rubric:
Working game and GUI – 15 points
Javadoc and Exception Handling – 5 points
Code review report – 10 points
Test Results and Postmortem reflection question responses – 5 points
PSP process – 15 points (Time log (4), Defect log (3), Estimation Worksheet (2), Design form(2), Project Summary(4)

PSP Time Recording Log

Date	Start	Stop	Interruption Time	Delta Time	Phase	Comments
4/6/19	3:00 PM	11:00 PM	180 min	300 min	code	Playing around with JavaFX
4/7/19	9:00 AM	11:30AM	0 min	15 min	code	Playing around with JavaFX
4/7/19	1:30 PM	1:45 PM	0 min	15 min	Plan	
4/7/19	1:45 PM	3:05 PM	0 min	80 min	Design	Flow Chart
4/7/19	3:07 PM	3:15 PM	0 min	8 min	Design	Class Diagram
4/7/19	4:10 PM	10:30 PM	60 min	380 min	Code	
4/8/19	2:30 PM	5:30 PM	0 min	180 min	Code	
4/8/19	5:30 PM	6:15 PM	0 min	45 min	Code Review	
4/8/19	6:15PM	6:50 PM	0 min	35 min	Test	

- **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.
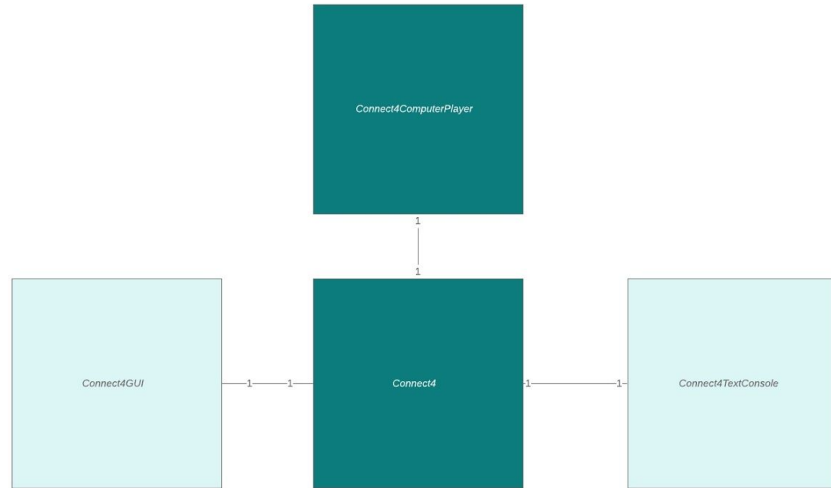
PSP1 Informal Size Estimating Procedure

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:
    * visualization.
    * recollection of similar chunks that youíve previously written
    * intuition.
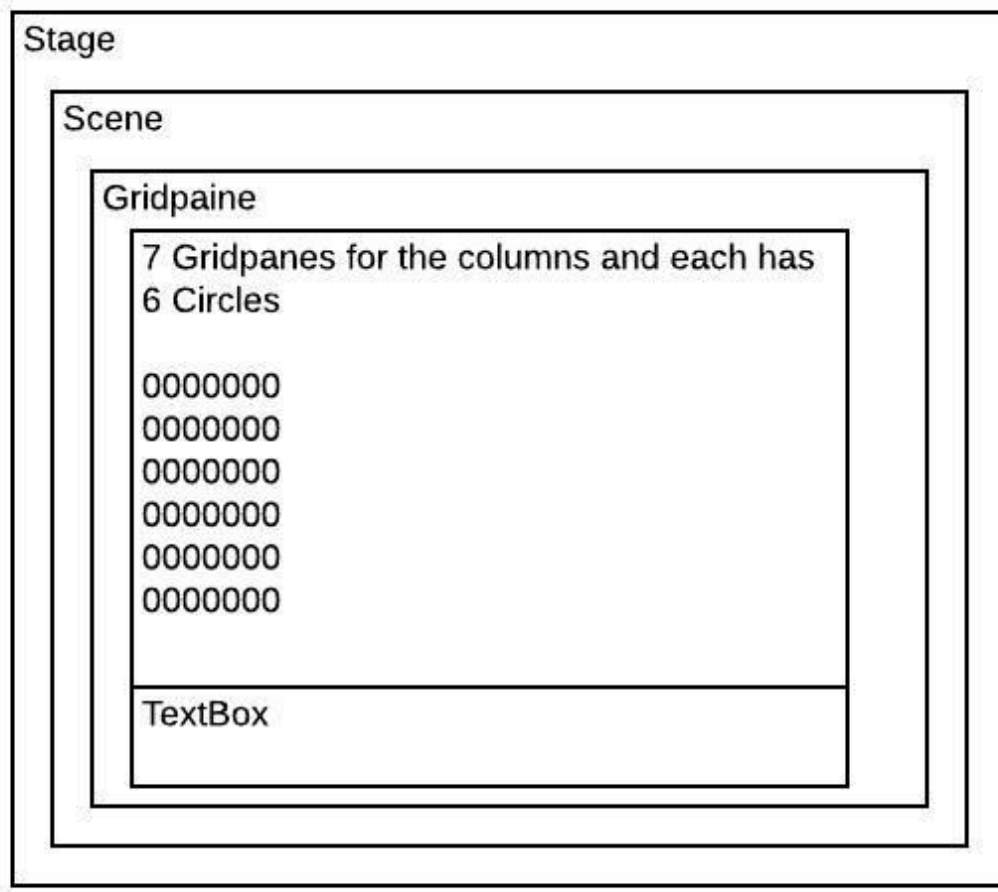5. Add the sizes of the individual chunks to get a total.

Estimating Worksheet

1. Conceptual Design (sketch your high-level design here)

The context of Connect4

Marcus Miller | April 7, 2019

```
                    ┌─────────────────────┐
                    │                     │
                    │ Connect4ComputerPlayer │
                    │                     │
                    └──────────┬──────────┘
                               1
                               │
                               1
┌─────────────┐      ┌─────────────────────┐      ┌─────────────────┐
│             │      │                     │      │                 │
│ Connect4GUI │─1──1─│      Connect4       │─1──1─│ Connect4TextConsole │
│             │      │                     │      │                 │
└─────────────┘      └─────────────────────┘      └─────────────────┘
```

```
Stage
  Scene
    Gridpaine
      7 Gridpanes for the columns and each has
      6 Circles

      0000000
      0000000
      0000000
      0000000
      0000000
      0000000


      TextBox
```

2. Module Estimates

Module description	Estimated Size

Flow Chart	45 min
Class Diagram	30 min
GUI Design	30 min
Code GUI Display	120 min
Integrate GUI and Connect4 Logic	60 min
Test	90 min
PostMortem	30 min

Total Estimated Size: _____405_____

PSP2 Project Summary

Time in Phase (minutes)	Estimated	Actual	To Date	To Date %
Planning	15	15	85	4.83
Design	105	88	296	16.82
Code	180	755	1069	60.74
Code Review	60	45	45	2.56
Test	90	35	195	11.08
Postmortem	30	15	70	3.98
TOTAL	480	953	1760	100

Defects Injected	Estimated	Actual	To Date	To Date %
Planning	0	0	0	0
Design	3	0	5	33.33
Code	5	4	9	60.00
Code Review	3	1	1	6.67
Test	1	0	0	0
Postmortem	0	0	0	0
TOTAL	12	4	15	100

SUMMARY

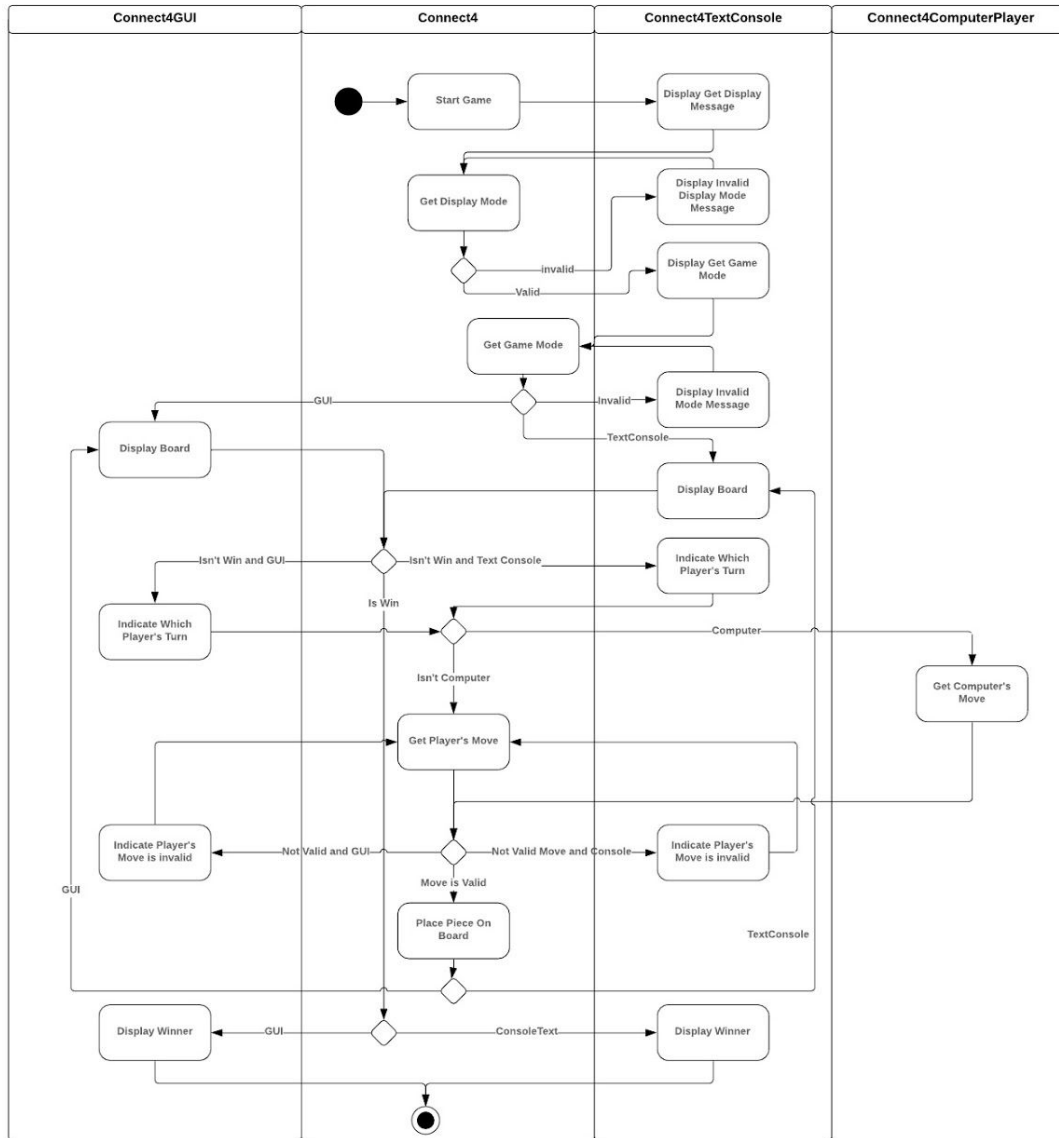|	Estimated	Actual	To Date
Program Size (LOC)	200	287	861
LOC/Hour	20	18.07	29.35
Defects/KLOC	20	13.94	16.26

- LOC is lines of Code
- KLOC is Kilo lines of code (i.e. 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.
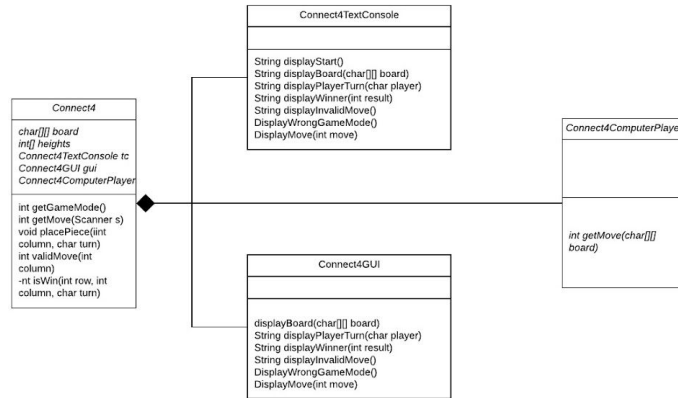
Connect4 Activity Diagram

Marcus Miller | April 7, 2019

Connect4GUI	Connect4	Connect4TextConsole	Connect4ComputerPlayer

Start Game

Display Get Display Message

Get Display Mode

Display Invalid Display Mode Message

invalid / Valid

Display Get Game Mode

Get Game Mode

Display Invalid Mode Message

GUI

Invalid

TextConsole

Display Board

Display Board

Isn't Win and GUI

Isn't Win and Text Console

Indicate Which Player's Turn

Is Win

Indicate Which Player's Turn

Computer

Isn't Computer

Get Computer's Move

Get Player's Move

Indicate Player's Move is invalid

Not Valid and GUI

Not Valid Move and Console

Indicate Player's Move is invalid

GUI

Move is Valid

Place Piece On Board

TextConsole

Display Winner

GUI

ConsoleText

Display Winner

Connect4 Class Diagram

Marcus Miller | April 8, 2019

Connect4TextConsole

String displayStart()
String displayBoard(char[][] board)
String displayPlayerTurn(char player)
String displayWinner(int result)
String displayInvalidMove()
DisplayWrongGameMode()
DisplayMove(int move)

Connect4

char[][] board
int[] heights
Connect4TextConsole tc
Connect4GUI gui
Connect4ComputerPlayer

int getGameMode()
int getMove(Scanner s)
void placePiece(iint
column, char turn)
int validMove(int
column)
-nt isWin(int row, int
column, char turn)

Connect4ComputerPlayer

*int getMove(char[][]
board)*

Connect4GUI

displayBoard(char[][] board)
String displayPlayerTurn(char player)
String displayWinner(int result)
String displayInvalidMove()
DisplayWrongGameMode()
DisplayMove(int move)

PSP Defect Recording Log

Sl. No.	Date	Defect Type	Defect Inject Phase	Defect Removal Phase	Fix Time	Fix Ref	Description
1	4/7/19	20	Code	Test	60 min	n/a	Conditionals had 'x' instead of 'X'
2	4/8/19	70	Code	Test	60 min	n/a	Board created in GUI had rows in reverse order
3	4/8/19	80	Code	Test	60 min	n/a	6 moves would be done at once when playing another player because I didn't have a way of telling when I read a move from the GUI.
4	4/8//19	10	Code	Test	5 min	n/a	I put the import statements in between javadoc comments and class.
5	4/8/19	50	Code	Code Review	5 min	n/a	Didn't close file

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

Code Review Checklist – Java

1. Specification / Design
[Y] Is the functionality described in the specification fully implemented by the code?
[N] Is there any excess functionality in the code but not described in the specification?

 2. Initialization and Declarations
[Y] Are all local and global variables initialized before use?
[Y] Are variables and class members of the correct type and appropriate mode
[Y] Are variables declared in the proper scope?
[Y] Is a constructor called when a new object is desired?
[Y] Are all needed import statements included?
[Y] Names are simple and if possible short
[ ] There are no usages of 'magic numbers' (i.e, hard-coded values) 'X' and 'O' are hard coded

3. General
[N] Code is easy to understand : Conditionals could be combined and I have multiple lambdas that are almost the same that could probably be combined
[Y] Variable and Methods names are spelt correctly
[Y] There is no dead code (i.e., code inaccessible at Runtime)
[N] Code is not repeated or duplicated
[N] No empty blocks of code

4. Method Calls

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

5. Arrays/Data structures
[N] Are there any off-by-one errors in array indexing?
[N] Can array indexes ever go out-of-bounds?
[Y] Is a constructor called when a new array item is desired?
[N] Ideal data structures are used. I think there is a better way to share data between threads than files which is what I did.
[Y] Collections are initialized with a specific estimated capacity

6. Object
[Y] Are all objects (including Strings) compared with "equals" and not "=="?
[Y] No object exists longer than necessary
[M] Files/Sockets and other resources if used are properly closed even when an exception occurs in using them. Not sure if I did this correctly

7. Output Format
[Y] Are there any spelling or grammatical errors in displayed output?
[Y] Is the output formatted correctly in terms of line stepping and spacing?

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

9. Exceptions
[Y] Are all relevant exceptions caught?
[Y] Is the appropriate action taken for each catch block?
[Y] Are all appropriate exceptions thrown?
[N] Are Catch clauses are fine-grained and catch specific exceptions?

10. Flow of Control
[n/a] In a switch statement is every case terminated by break or return?
[n/a] Do all switch statements have a default branch?
[x] Check that nested if statements don't have "dangling else" problems.
[Y] Are all loops correctly formed, with the appropriate initialization, increment and termination expressions?
[Y] Are open-close parentheses and brace pairs properly situated and matched?

11. Files

[Y] Are all files properly declared and opened?
[M] Are all files closed properly, even in the case of an error? Not sure if I did this right.
[Y] Are EOF conditions detected and handled correctly?
[Y] Are all file exceptions caught?

12. Documentation
[Y] All methods are commented in clear language.
[N] Comments exist and describe rationale or reasons for decisions in code
[Y] All public methods/interfaces/contracts are commented describing usage
[Y] All edge cases are described in comments
[Y] All unusual behavior or edge case handling is commented
[Y] Data structures and units of measurement are explained