**Project Deliverable 6 - Group**

**Personal Software Process & Quality**

**(Peer review of code, Software Testing Tool survey)**

**Points: 20**

**----------------------------------------------------------------------------------------------------**

**Instructions: GROUP DELIVERABLE**

This is a group submission and is sufficient for one person from the team to submit. A separate group submission link will be provided in the course shell. Submit the following: Team#-ProjectDeliverable6.zip containing the following files in pdf format.

1. Peer review form for all team members (10 points)
2. Software testing tools – survey report (for 2 tools) (10 points)

**----------------------------------------------------------------------------------------------------**

1. Peer Review

Conduct an inspection and peer review of your team members Connect4 code. Review the latest code from Deliverable 4 with a focus on core package classes (*GameLogic* class and *ComputerPlayer* class). It is sufficient if every team member gets their code reviewed by one other member of their team. Use the checklist provided (feel free to customize it). Make sure you provide appropriate comments on the checklist to demonstrate your review and feedback that was provided. Points will be deducted if there are no comments at all or if it is at a minimal level.

1. Software Testing Tool survey report

A consolidated list of open source software testing tools is available at <http://www.opensourcetesting.org/> (click on Test tools and select a category). A list of tools is also provided on next page. You can feel free to select a tool outside of this list as well. Each team is required to select 2 testing tools from this list provided, each from a different category of tools (such as Unit Testing, Functional Testing, Defect tracking, Continuous Integration, etc.). A tool not listed here may also be selected as long as it is applicable to your Connect4 project. Each team will provide a written report with description of the 2 tools. For each tool, you must download the tool, learn to use it, try it out on your Connect4 project, and demonstrate findings on your project (pick any deliverable and sufficient to demonstrate on one team member’s project). Your written report must answer the questions given below.

1. Who (company or individual) developed the tool? What is the cost to the tool user? How do you acquire it?
2. What testing purpose does the tool serve? (i.e, what problem does it attempt to solve? How does it improve productivity?)
3. What programming language(s) does the tool support, if any?
4. In what phase of software testing is the tool useful?
5. What do you need to do in order to use the tool?
   1. How do you install it?
   2. How do you configure it?
   3. How do you use it?
6. What are the strengths and limitations of the tool?
7. Demonstration of the tool on your project (via appropriate screenshots) and findings (i.e., how useful is it, how did it make testing process better, etc).

Table 1: List of Tools

|  |  |
| --- | --- |
| Unit Testing | EasyMock |
| Mockito |
| Arquillian |
| TestNG |
| Continuous Integration / Automation | TravisCI |
| JenkinsCI |
| CircleCI |
| ConcourseCI |
| Code Coverage | Cobertura |
| Static Analysis | CheckStyle |
| Spotbugs |
| JHawk |
| GUI Testing / Automation | Selenium |
| AutoIt |
| Cucumber |
| Project Management | Jira |
| Taiga |
| Functional Testing | Jubula |
| JMeter |
| System Testing | JSystem |

**Code Review Checklist – Java**

Connect4.java: **Blue**

Connect4ComputerPlayer.java: **Green**

1. Specification / Design

[**Y** **Y**] Is the functionality described in the specification fully implemented by the code?   
[**Y** **Y**] Is there any excess functionality in the code but not described in the specification?

**The logic asks for additional functionality for arena registration. The “computer” is named “Samantha” which does not specifically express to the user that this is the “computer.”**

**Logic in this class asks for additional functionality that coincides with the Connect4.java class. It asks the user for their name, an arena ID as well as naming the computer “Samantha.”**

2. Initialization and Declarations

[**Y** **Y**] Are all local and global variables initialized before use?   
[**Y** **Y**] Are variables and class members of the correct type and appropriate mode   
[**Y Y**] Are variables declared in the proper scope?   
[**Y** **Y**] Is a constructor called when a new object is desired?

**The default constructor is called. No other constructor is called that passes arguments, so accessors and mutators are not needed.**

**Connect4ComputerPlayer extends an additional class of Player. Player has a constructor that sets the registerID, gamerTag and gamerPiece. Connect4ComputerPlayer has its own constructor that calls upon the super constructor and sets the computer’s name to “Samantha” with a setPiece of String “O”**

[**Y** **Y**] Are all needed import statements included?

[**Y** **Y**] Names are simple and if possible short

[**N** N] There are no usages of ‘magic numbers’ (i.e, hard-coded values)  
**Several instances of magic numbers exist, such as in for loops and while loops. I am not sure it is entirely necessary to eradicate all magic numbers since these numbers will not need to be dynamic.**

**Magic numbers are set for the method, chooseRandom.**

3. General

[**Y** **Y**] Code is easy to understand

**Code is well documented by either comments or Javadoc. Code does not have exceeding characters on one line and is coded in tight logical statements without the need for extraneous code.**

**Code for chooseRandom is not your classical use of a random method. No commenting exists for the usage on line 30. There is some logic behind what is happening when the computer places a piece on the board, but documentation detailing this is left out. Having comments that give an understanding of the usage would help in understanding the logic.**

[**Y** **Y**] Variable and Methods names are spelt correctly

[**N** **N**] There is no dead code (i.e., code inaccessible at Runtime)

**No dead code located within the class**

[**N** **N**] Code is not repeated or duplicated

**No unnecessary repeated or duplicated code. There is code that looks similar, but this is due to the checking of vertical/horizontal and diagonal conditions.**

[**N** **N**] No empty blocks of code

4. Method Calls   
[**Y** **Y**] Are parameters presented in the correct order?   
[**Y** **Y**] Are parameters of the proper type for the method being called?  
[**Y** **Y**] Is the correct method being called, or should it be a different method with a similar name?

**All methods are named appropriately and are called as needed. No other method with similar name structure is called resulting in a bug/inoperable program.**

[**Y** **Y**] Are method return values used properly? Cast to the needed type?

**All return statements return to either True/False or Winner. There is no need for Winner to be casted to a different data type.**

**The Java Random method returns a double. The chooseRandom method returns an integer and is casted properly to take the return from a double to an integer. This is assumed because columns and rows are in integers and a double would return an error.**

5. Arrays/Data structures   
[**N** **N**] Are there any off-by-one errors in array indexing?   
[**N** **N**] Can array indexes ever go out-of-bounds?   
[**N** **N**] Is a constructor called when a new array item is desired?

**Only the default constructor was utilized. Arrays do not call upon a constructor when needed.**

**No arrays were used in this class**

[**Y** **Y**] Ideal data structures are used

[**N/A** **N/A**] Collections are initialized with a specific estimated capacity

**Collections were not used, nor needed.**

**Collections were not used, nor needed.**

6. Object   
[**N/A** **N/A**] Are all objects (including Strings)  compared with "equals" and not "=="?

[**N/A** **N/A**] No object exists longer than necessary

[**N/A** **N/A**] Files/Sockets and other resources if used are properly closed even when an exception occurs in using them

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

**This class does not output to the console as this method is only used for the logic behind how a computer places a piece on the board.**

8. Computation, Comparisons and Assignments   
[**N/A** **Y**] Check order of computation/evaluation, operator precedence and parenthesizing

**No mathematical calculations were needed for this class**

**All operator precedence and parenthesizing is in the correct placement/order.**

[**N/A** **N**] Can the denominator of a division ever be zero?

**No division in the chooseRandom method exists**

[**N/A** **N**] Is integer arithmetic, especially division, ever used inappropriately, causing unexpected truncation/rounding?   
[**Y** **N/A**] Check each condition to be sure the proper relational and logical operators are used.

**All conditions follow the proper logic**

[**N/A** **N/A**] If the test is an error-check, can the error condition actually be legitimate in some cases?   
[**N/A** **N**] Does the code rely on any implicit type conversions?

9. Exceptions

[**N** **N**] Are all relevant exceptions caught?

**There were no instances of code that handled exceptions using the throws method, however, there are customized methods that handle if a user enters an integer in the range of 1-7.  
No exceptions were caught in this class.**

[**N/A** **N/A**] Is the appropriate action taken for each catch block?

[**N/A** **N/A**] Are all appropriate exceptions thrown?

[**N/A** **N/A**] Are Catch clauses are fine-grained and catch specific exceptions?

10. Flow of Control

[**N/A** **N/A**] In a switch statement is every case terminated by break or return?

**No switch statements were used or needed in this class**

**No switch statements were used or needed in this class**

[**N/A** **N/A**] Do all switch statements have a default branch?  
[**N** **N**] Check that nested if statements don't have “dangling else” problems.   
[**Y** **Y**] Are all loops correctly formed, with the appropriate initialization, increment and termination expressions?   
[**Y** **Y**] Are open-close parentheses and brace pairs properly situated and matched?

11. Files

[**N/A** **N/A**] Are all files properly declared and opened?   
[**N/A** **N/A**] Are all files closed properly, even in the case of an error?   
[**N/A** **N/A**] Are EOF conditions detected and handled correctly?   
[**N/A** **N/A**] Are all file exceptions caught?

12. Documentation

[**Y** **N**] All methods are commented in clear language.

**Comments, and especially the Javadoc were commented extremely well. The Javadoc explains each method, what the parameters are and what it returns. The Javadoc header is well detailed and updated for each variation (update) of the class**

**The chooseRandom method is not commented and uses some logic as to where the computer places a piece on the board. Commenting here would be helpful to understand what the developer was trying to accomplish in terms of their algorithm.**

[**Y N**] Comments exist and describe rationale or reasons for decisions in code

**No commenting exists for this class, however a Javadoc is presented at the beginning of the class.**

[**Y** **N**] All public methods/interfaces/contracts are commented describing usage

[**N** **N**] All edge cases are described in comments

**Edge cases are not described in comments, that is, if, edge cases even exist.**

[**N** **N**] All unusual behavior or edge case handling is commented

[**N/A N/A**] Data structures and units of measurement are explained