Software Engineering 2 Project, AA 2015-2016
Assignment 3: Code Inspection

Code inspection (e.g., code analysis, visual inspection, reverse engineering, etc.) is systematic examination (often known as peer review) of computer source code. It is intended to find mistakes overlooked in the initial development phase, improving both the overall quality of software and the developers' skills. Reviews are done in various forms such as pair programming, informal walkthroughs, and formal inspections. You are to apply Code Inspection techniques (supported by the review checklist at the end of this document) for the purpose of evaluating the general quality of selected code extracts from a release of the Glassfish 4.1 application server (see the information at the end of the document).
In the scope of this assignment, you will be given a selected number of classes extracted from said software release. Said selection is done systematically, and will assign different sets of classes to different groups. The systematic selection makes sure that the difficulty of the assignment is homogeneous per every group and takes into account the fact that we have groups of different sizes. We have allocated about 80 lines of code per person[1].

Your tasks

1. Download the source code of Glassfish directly with the following command:
   svn checkout https://svn.java.net/svn/glassfish~svn/tags/4.1.1@64219 src
2. Find the classes/methods to inspect at the following URL: http://assignment.pompel.me/. Each group is identified by the name of the student who registered the group and by the reference professor. Refer to the javadoc for the selected version of Glassfish here http://glassfish.pompel.me
3. Perform the inspection. Each group shall report on the quality status of selected code extracts using the checklist for JAVA code inspection reported in appendix to this document. Students are expected to deliver a document (which we will call Deliverable from now on) having the structure described in the next section.

Suggested structure for the inspection document

Front page: Include at least the project title, the version of the document, your names and the release date

Table of content: Include the table of content of your document

[1] Clustering has been performed thanks to a series of scripts developed by Andrea Brancaleoni, a last year course student, as part of his 150 hours service. The scripts ensure that the complexity of clusters is homogeneous.

Classes that were assigned to the group: <state the namespace pattern and name of the classes that were assigned to you>

Functional role of assigned set of classes: <elaborate on the functional role you have identified for the class cluster that was assigned to you, also, elaborate on how you managed to understand this role and provide the necessary evidence, e.g., javadoc, diagrams, etc.>

List of issues found by applying the checklist: <report the classes/code fragments that do not fulfill some points in the check list. Explain which point is not fulfilled and why>.

Any other problem you have highlighted: <list here all the parts of code that you think create or may create a bug and explain why>.

Additional Material

If you want to run our reference version of Glassfish, you can download the virtual machine that we have prepared from any of the following mirroring sites:

- https://www.dropbox.com/s/3nwjyd6v0b8360j/box.ova?dl=0
- https://mega.nz/#!tM8WAYJB!7rwaS6zGehspQLOQcLiCi3pQbfuI5lq9Hvi9d2Plim8
- https://polimi365-my.sharepoint.com/personal/10323862_polimi_it/_layouts/15/guestaccess.aspx?guestaccesstoken=bIAhEbmV%2bvT7M%2fzjsogcU5vD%2bVC8b%2fXd%2fDqGAQ%2b5N1c%3d&docid=0c4c8faeeb19b4251812813fffc292036

Code inspection checklist

Naming Conventions

1. All class names, interface names, method names, class variables, method variables, and constants used should have meaningful names and do what the name suggests.
2. If one-character variables are used, they are used only for temporary "throwaway" variables, such as those used in for loops.
3. Class names are nouns, in mixed case, with the first letter of each word in capitalized. Examples: class Raster; class ImageSprite;
4. Interface names should be capitalized like classes.
5. Method names should be verbs, with the first letter of each addition word capitalized. Examples: getBackground(); computeTemperature().
6. Class variables, also called attributes, are mixed case, but might begin with an underscore ('_') followed by a lowercase first letter. All the remaining words in the variable name have their first letter capitalized. Examples: _windowHeight, timeSeriesData.
7. Constants are declared using all uppercase with words separated by an underscore. Examples: MIN_WIDTH; MAX_HEIGHT;

Indention

8. Three or four spaces are used for indentation and done so consistently
9. No tabs are used to indent

Braces

10. Consistent bracing style is used, either the preferred "Allman" style (first brace goes underneath the opening block) or the "Kernighan and Ritchie" style (first brace is on the same line of the instruction that opens the new block).
11. All if, while, do-while, try-catch, and for statements that have only one statement to execute are surrounded by curly braces. Example:

Avoid this:

```
if ( condition )
  doThis();
```

Instead do this:

```
if ( condition )
{
  doThis();
}
```

File Organization

12. Blank lines and optional comments are used to separate sections (beginning comments, package/import statements, class/interface declarations which include class variable/attributes declarations, constructors, and methods).
13. Where practical, line length does not exceed 80 characters.

14. When line length must exceed 80 characters, it does NOT exceed 120 characters.

Wrapping Lines

15. Line break occurs after a comma or an operator.
16. Higher-level breaks are used.
17. A new statement is aligned with the beginning of the expression at the same level as the previous line.

Comments

18. Comments are used to adequately explain what the class, interface, methods, and blocks of code are doing.
19. Commented out code contains a reason for being commented out and a date it can be removed from the source file if determined it is no longer needed.

Java Source Files

20. Each Java source file contains a single public class or interface.
21. The public class is the first class or interface in the file.
22. Check that the external program interfaces are implemented consistently with what is described in the javadoc.
23. Check that the javadoc is complete (i.e., it covers all classes and files part of the set of classes assigned to you).

Package and Import Statements

24. If any package statements are needed, they should be the first non-comment statements. Import statements follow.

Class and Interface Declarations

25. The class or interface declarations shall be in the following order:
    A. class/interface documentation comment
    B. class or interface statement
    C. class/interface implementation comment, if necessary
    D. class (static) variables
        a. first public class variables
        b. next protected class variables
        c. next package level (no access modifier)
        d. last private class variables
    E. instance variables
        a. first public instance variables
        e. next protected instance variables
        f. next package level (no access modifier)
        g. last private instance variables

F. constructors
G. methods

26. Methods are grouped by functionality rather than by scope or accessibility.
27. Check that the code is free of duplicates, long methods, big classes, breaking encapsulation, as well as if coupling and cohesion are adequate.

Initialization and Declarations

28. Check that variables and class members are of the correct type. Check that they have the right visibility (public/private/protected)
29. Check that variables are declared in the proper scope
30. Check that constructors are called when a new object is desired
31. Check that all object references are initialized before use
32. Variables are initialized where they are declared, unless dependent upon a computation
33. Declarations appear at the beginning of blocks (A block is any code surrounded by curly braces "{" and "}"). The exception is a variable can be declared in a 'for' loop.

Method Calls

34. Check that parameters are presented in the correct order
35. Check that the correct method is being called, or should it be a different method with a similar name
36. Check that method returned values are used properly

Arrays

37. Check that there are no off-by-one errors in array indexing (that is, all required array elements are correctly accessed through the index)
38. Check that all array (or other collection) indexes have been prevented from going out-of-bounds
39. Check that constructors are called when a new array item is desired

Object Comparison

40. Check that all objects (including Strings) are compared with "equals" and not with "=="

Output Format

41. Check that displayed output is free of spelling and grammatical errors
42. Check that error messages are comprehensive and provide guidance as to how to correct the problem
43. Check that the output is formatted correctly in terms of line stepping and spacing

Computation, Comparisons and Assignments

44. Check that the implementation avoids "brutish programming: (see http://users.csc.calpoly.edu/~jdalbey/SWE/CodeSmells/bonehead.html)

45. Check order of computation/evaluation, operator precedence and parenthesizing
46. Check the liberal use of parenthesis is used to avoid operator precedence problems.
47. Check that all denominators of a division are prevented from being zero
48. Check that integer arithmetic, especially division, are used appropriately to avoid causing unexpected truncation/rounding
49. Check that the comparison and Boolean operators are correct
50. Check throw-catch expressions, and check that the error condition is actually legitimate
51. Check that the code is free of any implicit type conversions

Exceptions

52. Check that the relevant exceptions are caught
53. Check that the appropriate action are taken for each catch block

Flow of Control

54. In a switch statement, check that all cases are addressed by break or return
55. Check that all switch statements have a default branch
56. Check that all loops are correctly formed, with the appropriate initialization, increment and termination expressions

Files

57. Check that all files are properly declared and opened
58. Check that all files are closed properly, even in the case of an error
59. Check that EOF conditions are detected and handled correctly
60. Check that all file exceptions are caught and dealt with accordingly