Politecnico di Milano

Master in Computer Science and Engineering

Software Engineering 2 Project:
myTaxiService

Code Inspection Document

Authors:
Filippo Leporati
Danilo Fusi

Professor:
Elisabetta Di Nitto

1 TABLE OF CONTENTS

2 CODE INSPECTION CHECKLIST

This is the code inspection checklist used to analyze the java class.

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.
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

3 THE CLASS

The class we are going to analyse is called LogHandle.java. It is part of open source project: GlassFish (4.1.1 version).

It is located in package:

appserver/transaction/jts/src/main/java/com/sun/jts/CosTransactions/

3.1 FUNCTIONAL ROLE OF THE CLASS

According to the Javadoc in the file, this class contains attributes of an open log file.

```
/**A class containing attributes of an open log file.
 *
 * @version 0.01
 *
 * @author Simon Holdsworth, IBM Corporation
 */
```

3.2 CODE INSPECTION

Checklist #	Observed	Comment	Code Lines of reference
1	✓	All class names, interface names, method names, class variables, method variables, and constants have meaningful names and do what the name suggests.	-
2	✓	No one-character are used in the class	-
3	✓	Class name is a noun with the first letter of each word in capitalized (LogHandle)	-
4	✓	No interfaces are used in the class	-
5	×	One method is not a verb (logFileName():string)	2021
6	✓	Class variables are corrected spelled correctly, mixed case and the remaining word have their first letter capitalized.	-
7	✓	Constants are declared using all uppercase with the words separated by underscore	-
12	✓	Blank lines and optional comments separates sections	-
20	✓	Java source file contains a single public class	-
21	✓	The public class is the first class in the file	-
22	✓	(No external interfaces used)	-
23	✓	Javadoc is completed. It covers the class, the variables and the methods	-
24	✓	Package statements are the first non-comment statements. Import statements follow.	-
25	×	The class declarations do not follow the correct order. D) Class static private variables is before the package ones.	87
26	✓	Methods are grouped by functionality	-

27	×	Class contains long methods. Empty method with inadequate commented out code.	284,940,1074; 1940
28	✓	Variables and class members are of the correct type	-
29	✓	Variables are declared in correct scope	-
30	✓	Constructor are called when a new object is desired	-
31	✓	All object are initialized before use	-
32	✓	Variables are initialized where they are declared	-

4 METHODS

This section analyses the method selected in order to find possible issues and problems.

4.1 TRUNCATE(LOGLSN TRUNCLSN , INT INCLUSIVE)

4.1.1 Description

This method (line 1074) has the following signature:

```
synchronized void truncate( LogLSN truncLSN, int inclusive )  throws LogException
```

According to the Javadoc, this function truncates the log at the given point.

```
/**Truncates the log at the given point.
 *
 * @param truncLSN  The LSN of the truncation point.
 * @param inclusive Indicates whether truncation includes the LSN.
 *
 * @return
 *
 * @exception LogException  The operation failed.
 *
 * @see
 */
```

4.1.2 Code Inspection

Checklist #	Observed	Comment	Code Lines of reference
8	✓	Four spaces are used consistently	-
9	✓	No tabs are used to indent	-
10	✓	Kernighan and Ritchie style is used: the first brace is on the same line of instruction that opens the new block	-
11	×	A lot of “if” statements with only one statement are not surrounded with curly braces	1084, 1090, 1096, 1140, 1203, 1220
13	×	Some lines exceed 80 characters of length	1084, 1178
14	✓	All lines don’t exceed 120 characters of length	-
15	✓	Line break occurs after a comma or an operator	-
16		??	
17	✓	A new statement is aligned with the beginning of the expression at the same level as the previous line	-
18	✓	Comments are used to adequately explain what the blocks of code are doing	-

Checklist #	Observed	Comment	Code Lines of reference
19	×	One line of commented out code does not contain any type of explanation	1284
29			
30	✓	Constructors are called when a new object is desired	-
31	✓	All object references are initialized before use	-
32	✓	Variables are initialized where they are declared, unless dependent upon a computation	-
33	×	Many variables are not declared at the beginning of the block	1118, 1119, 1153, 1154, 1155, 1178, 1183, 1184, 1195, 1237, 1238, 1285, 1286
34	✓	Method parameters are presented in the correct order	-
35	✓	Correct methods are being called	-
36	✓	Methods returned values are used properly	-
37	✓	There are no off-by-one errors in array indexing	-
38	✓	All array indexes have been prevented from going out-of-bound	-
39	✓	Constructor are called when a new array item is desired	-
40	✓	All comparison between objects are done with “equals” method	-
41	✓	No displayed output	-
42	✓	No error messages displayed	-
43	✓	No displayed output	-
44	✓	Implementation avoids “brutish programming”	-
45	✓	Order of computation/evaluation, operator precedence and parenthesizing	-
46	✓	No problems of operator precedence	-
47	✓	No division by zero	-
48	✓	Integer arithmetic are used appropriately to avoid causing unexpected truncation	-
49	✓	Comparison and boolean operators are correct	-
50	✓	Throw-catch expressions are valid	-
51	✓	Code is free of any implicit type conversions	-
52	✓	Relevant exceptions are caught	-
53	✓	For each catch block, an exception is throws	-
54	✓	No switch statements	-
55	✓	No switch statements	-
56	✓	All loops are correctly formed	-
57	✓	No file used	-
58	✓	No file used	-
59	✓	No file used	-
60	✓	No file used	-

4.2 CHECKLSN(LOGLSN CHKLSN)

4.2.1 Description

This method (line 1343) has the following declaration:
synchronized void checkLSN(LogLSN chkLSN) throws LogException

Its function is to ensure that the log is written up to the given point. Its Javadoc follows:

```
/**Ensures that the log is written up to the given point.
 *
 * @param chkLSN  The last LSN which must be written.
 *
 * @return
 *
 * @exception LogException  The operation failed.
 *
 * @see
 */
```

4.2.2 Code Inspection

Checklist #	Observed	Comment	Code Lines of reference
8	✓	Four spaces are used consistently	-
9	✓	No tabs are used to indent	-
10	✓	Kernighan and Ritchie style is used: the first brace is on the same line of instruction that opens the new block	-
11	×	A lot of "if" statements with only one statement are not surrounded with curly braces	1351, 1357,1363, 1374,1399
13	×	Just one line exceeds 80 characters of length	1352
14	✓	All lines don't exceed 120 characters of length	-
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	-
18	✓	Comments are used to adequately explain what the blocks of code are doing	-
19	✓	Commented out code contain adequate explanations	-
33	✓	Declaration appear at the beginning of the block	-
34	✓	Method parameters are presented in the correct order	-
35	✓	Correct methods are being called	-
36	✓	Methods returned values are used properly	-
37	✓	There are no off-by-one errors in array indexing (no arrays used)	-
38	✓	All array indexes have been prevented from going out-of-bound(no arrays are used)	-

39	✓	Constructor are called when a new array item is desired (no arrays are used)	-
40	✓	All objects are compared with “equals”	-
41	✓	No displayed output	-
42	✓	No error messages displayed	-
43	✓	No displayed output	-
44	✓	Implementation avoids “brutish programming”	-
45	✓	Order of computation/evaluation, operator precedence and parenthesizing	-
46	✓	No problems of operator precedence	-
47	✓	No division by zero	-
48	✓	No arithmetic operations (i.e. division)	-
49	✓	Comparison and boolean operators are correct	-
50	✓	No throw-catch expressions	-
51	✓	Code is free of implicit type conversion	-
52	✓	No relevant exceptions caught	-
53	✓	No catch blocks	-
54	✓	No switch statements	-
55	✓	No switch statements	-
56	✓	All loops (just one “for” statement) are correctly formed	-
57	✓	No file used	-
58	✓	No file used	-
59	✓	No file used	-
60	✓	No file used	-