POLITECNICO
MILANO 1863

Politecnico di Milano

Software Engineering 2 project

PowerEnJoy

Code Inspection Document

Matteo Penco
mat. 875740

Riccardo Pressiani
mat. 874948

Version 1.0 approved

February 5, 2017

Revision History

Revision	Date	Author(s)	Description
0.1	31.01.17	RP	Document created
0.1	05.01.17	RP and MP	Document approved

Hours of work

Matteo Penco	10h
Riccardo Pressiani	10h

Contents

Chapter 1

Code Description

The purpose of this document is to perform a thorough code analysis, also known as "Code Inspection", of the assigned class. The goal of this process is to find mistakes in the source code of possible "best practices" that have not been adopted by the developers. In the Appendix A, the checklist that has been used to perform the code inspection is provided.

The class that has been inspected is part of the Apache OFBiz project (version 16.11.01), an open source product for the automation of enterprise processes that includes framework components and business applications for ERP (Enterprise Resource Planning), CRM (Customer Relationship Management) and other business-oriented functionalities.

1.1 Assigned Class

In this document the `MiscConverter` class of the Apache OFBiz project [1] will be inspected. The class is located in the `org.apache.ofbiz.base.conversion` package.

1.2 Functional Role

The class `MiscConverters` implements `ConverterLoader` interface: other applications implement this interface in order to load their Java object converters. More in detail, this interface provides `loadConverters` method which has the following role, as stated by the Javadoc: "*Create and register converters with the Java object type conversion framework. If the converter extends one of the converter abstract classes, then the converter will register itself when an instance is created.*"

Chapter 2

Code Issues

2.1 Checklist Issues

2.1.1 Naming Conventions

C3 Class `NotAConverter_Helper` declared at line 343 has an underscore in the name which is not allowed. Moreover, several classes do not have a noun in the name. The lines where those classes can be found are listed below:

- L.41
- L.55
- L.69
- L.83
- L.97
- L.111
- L.140
- L.179
- L.204
- L.214
- L.228
- L.243
- L.253
- L.263
- L.273
- L.283
- L.293

- L.303
- L.313
- L.323
- L.333
- L.349

2.1.2 Indention

C8 At L.210, 13 spaces found when 12 expected.

2.1.3 Braces

No issues found.

2.1.4 File Organization

C12 Blank lines to separate code sections have been inserted quite accurately. No comments to introduce the sections are provided.

C13 The following lines exceed 80 characters and are not considered to be easily modified to fit the given length:

- L.48
- L.123
- L.147
- L.171
- L.186
- L.221
- L.238

2.1.5 Wrapping Lines

No issues found.

2.1.6 Comments

C18 Only a comment with the extended name of the class is provided. There are no comments to explain the purpose of the code written.

2.1.7 Java Source Files

C22 Since the Javadoc is not provided, it is not possible to check that the external program interfaces are implemented consistently.

C23 The whole class is not covered by Javadoc.

2.1.8 Package and Import Statements

No issues found.

2.1.9 Class and Interface Declarations

C25 The class declarations order is correct except for the fact that the class documentation comment and implementation comment are missing.

C27 The whole class MiscConverters is composed by several inner classes, reaching a total lenght of 320 lines.

2.1.10 Initialization and Declarations

No issues found.

2.1.11 Method Calls

No issues found.

2.1.12 Arrays

No issues found.

2.1.13 Object Comparison

C40 Two objects are compared using `==` instead of `equals` at L.171.

2.1.14 Output Format

C42 Two error messages are defined in case of exceptions in the class (L.238-L.345). These two messages could have been more expressive. Moreover, many `catch` blocks do not provide error messages at all.

2.1.15 Computation, Comparisons and Assignments

No issues found.

2.1.16 Exceptions

C53 No instructions are provided in the `catch` block of the `IOException` at L.133.

2.1.17 Flow of Control

No issues found.

2.1.18 Files

No issues found.

2.2 Other Issues

After a thorough analysis on the assigned class, no other issues have been found.

Appendix A

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.
4. Interface names should be capitalized like classes.
5. Method names should be verbs, with the first letter of each addition word capitalized.
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.
7. Constants are declared using all uppercase with words separated by an underscore.

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.

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. External program interfaces are implemented consistently with what is described in the Javadoc.

23. Check that the Javadoc is complete.

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;
        i. first public class variables;
        ii. next protected class variables;
        iii. next package level (no access modifier);
        iv. last private class variables.
    (e) instance variables;
        i. first public instance variables;
        ii. next protected instance variables;
        iii. next package level (no access modifier);
        iv. 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. Variables and class members are of the correct type. Check that they have the right visibility (public/private/protected).

29. Variables are declared in the proper scope.

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. Declarations appear at the beginning of blocks (block: any code surrounded by curly braces '{' and '}'). The exception is a variable can be declared in a `for` loop.

Method Calls

34. 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. Method returned values are used properly.

Arrays

37. There are no off-by-one errors in array indexing (that is, all required array elements are correctly accessed through the index).

38. All array (or other collection) indexes have been prevented from going out-of-bounds.

39. Constructors are called when a new array item is desired.

Object Comparison

40. All objects (including `String`s) are compared with `equals` and not with `==`.

Output Format

41. Displayed output is free of spelling and grammatical errors.

42. Error messages are comprehensive and provide guidance as to how to correct the problem.

43. The output is formatted correctly in terms of line stepping and spacing.

Computation, Comparisons and Assignments

44. Implementation avoids "brutish programming". (See `http://users.csc.calpoly.edu/~jdalbey/SWE/CodeSmells/bonehead.html` for more information.)
45. Check order of computation/evaluation, operator precedence and parenthesizing.
46. Liberal use of parenthesis is used to avoid operator precedence problems.
47. All denominators of a division are prevented from being zero.
48. Integer arithmetic, especially division, is used appropriately to avoid causing unexpected truncation/rounding.
49. 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 most relevant exceptions are caught.
53. 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. All switch statements have a default branch.
56. All loops are correctly formed, with the appropriate initialization, increment and termination expressions.

Files

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

Bibliography

[1] The Apache Software Foundation. Apache OFBiz. Available at `http://mirror.nohup.it/apache/ofbiz/apache-ofbiz-16.11.01.zip` (February 5, 2017).