POLITECNICO
MILANO 1863

5 February 2017

PowerEnJoy

Code Inspection

Blanco	Federica	875487
Casasopra	Fabiola	864412

Software Engineering 2 Project
2016/2017

Contents

List of Tables

1 Introduction

Code inspection is the systematic examination of computer source code. Our scope in this phase is to find mistakes overlooked during the initial development phase, in order to improve the quality of the software and also the developers' skill.
Moreover, this technique is one of the most effective to identify security flaws. In this specific context, the manual approach has a great value: a human reviewer can understand the context for coding practices and, consequently, he can make a valid risk estimation, since he has the ability to evaluate both the probability of attack and the business impact that a breach could have on the company.

1.1 Purpose

We are now going to analize the purpose of the code inspection.
The first purpose is to check that the code of the software has at least the quality needed in order to be released. When reviewing the code, the develoers are able to find errors of all types, for example those that derives from a poor structure, but also the simpler one caused by omissions.
The second purpose is to help developers improve they ability about when and how to apply techniques in order to improve code quality, consistency, and maintainability.

1.2 List of Definitions and Abbreviations

Here there is the acronims and abbreviations list:

EOF End Of File

1.3 List of Reference Documents

- Specification document: Code Inspection Assignment Task Description.pdf
- *Oracle Java Code Convention*: `http://www.oracle.com/technetwork/java/javase/documentation/codeconvtoc-136057.html`

1.4 Document overview

Here we show the stucture of the document, with a brief overview of each section.

Section 1 There is an introduction with general information about this document.

Section 2 There are shown the coding convention and the issue checklist used for the code inspection.

Section 3 There is the central part, in which we declare the classes of code to be inspected and the issue found.

Section 4 Here there are given additional information that may be useful to the reader, such as the tools used and the time spent to redact this document.

2 Coding Conventions and Issue Checklist

2.1 Coding Conventions

Coding conventions are a set of guidelines for a specific programming language that recommend programming style, practices and methods for each aspect of a program written in this language. These conventions are useful for the developers, since they allow to improve the readability of their source code and make software maintenance easier. The convention are guidelines for software structural quality, since they are about file organization, indentation, comments, declarations, statements, white space, naming conventions, programming practices, programming principles, programming rules of thumb, architectural best practices, and more on. Coding conventions cannot be checked by a compliler, but only by human maintainers and peer reviewers of a software project.

As far as our code inspection is concerned, we are going to follow the *Oracle Java Code Conventions*.

2.2 Issue Checklist

In this part of the document, we are going to report the checklist of possible issues that we are going to refer during the inspection process.

2.2.1 Naming Convention

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`.

2.2.2 Indention

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

2.2.3 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();
}
```

2.2.4 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.

2.2.5 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.

2.2.6 Comments

18. Comments are used to adequately explain what the class, interface, meth- ods, 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.

2.2.7 Java Source File

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).

2.2.8 Package and Import Statements

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

2.2.9 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. next package level (no access modifier);

(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.

2.2.10 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.

2.2.11 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.

2.2.12 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.

2.2.13 Object Comparison

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

2.2.14 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.

2.2.15 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.

2.2.16 Exceptions

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

2.2.17 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.

2.2.18 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 Code Inspection Process

3.1 Assigned Classes

This sections contains the namespace pattern and the name of the class that were assigned to us:

Name: *ContentPermissionServices*

Location:*../apache-ofbiz-16.11.01/applications/content/src/main/java/org/apache/ofbiz/content/content/ContentPermissionServices.java*

3.2 Functional Role of the Assigned Set of Classes

This class, like the name suggest, provides services for granting operation permissions on content entities in a data-driven manner. It does it with the use of two methods: *"checkContentPermission"* and *"checkAssocPermission"*. The first one check, thanks to a lot of tests, if the user have the permission to perform operations; the second one check if there is a permission to associate a content to another.

3.3 List of Issue

Here is presented a table with all the possible issues listed before and if they are present or not in the class assigned to us.

Issue 1	This issue is not present
Issue 2	This issue is not present
Issue 3	There isn't this issue: the class is properly called
Issue 4	This issue is not present
Issue 5	There isn't this issue: the methods are correctly called
Issue 6	This issue is not present: the attributes are properly called
Issue 7	There isn't constant
Issue 8	This issue is present at lines: 108 (five spaces), 114 (five spaces), 192 (five spaces), 197 (five spaces), 226 (two spaces)
Issue 9	This issue is not present: any tabs are used
Issue 10	All the class contains the bracing style "Kernighan and Ritchie", this is not an issue
Issue 11	This issue is present at lines: 171, 180, 200, 205, 267
Issue 12	Blank lines not correct: 97, 137, 152, 168, 178, 184, 194, 199, 221, 229, 237, 242, 248, 269, 295, 328
Issue 13	Lines that exceed 80 characters but that can be reduced: 49, 72, 79-81, 86, 88, 91-92, 100, 102, 138, 140, 154
Issue 14	Lines that exceed 120 characters: 95, 96, 124, 153, 166, 169, 208
Issue 15	This issue is present at lines: 222-223 (line breaks before the OR operator)
Issue 16	This issue is not present
Issue 17	At the line 110 the statement is not aligned with the beggining of the expression at the same level
Issue 18	There are blocks of code not commented and all the second method isn't commented too
Issue 19	At lines 101 and 103 there isn't a date when the commented out code can be removed; at line 258 there isn't also a motivation because the code is commented out
Issue 20	This issue is not present: there is only a single public class in our file

Table 1: Check issues from 1 to 20

Issue 21	This issue is not present: the public class is the first in the file
Issue 22	This issue is not present
Issue 23	In the javadoc there is this class and the parameter and return of the first method but there is anything about the second method, so the second method is not in the javadoc
Issue 24	This issue is not present: package and import are in the correct order
Issue 25	This issue is not present: the class is defined in the correct order
Issue 26	This issue is not present
Issue 27	The class methods are too long: they do a lot of controls and actions
Issue 28	This issue is not present
Issue 29	This issue is not present
Issue 30	This issue is not present
Issue 31	This issue is not present
Issue 32	This issue is not present: all the variables are initialized where they are declarated
Issue 33	In this class we can find a lot of declarations maked not at the beggining of a block:ex line 111
Issue 34	This issue is not present
Issue 35	This issue is not present
Issue 36	This issue is not present
Issue 37	There aren't arrays
Issue 38	There isn't this issue
Issue 39	There aren't arrays
Issue 40	The objects are correctly compared

Table 2: Check issues from 21 to 40

Issue 41	The output of this class is in the log so there is a grammatical error at line 96: "depricated" instead of "deprecated"
Issue 42	At lines 129 and 215 there isn't the specific error or the guide to solve it, at line 276 there isn't the guide to solve it
Issue 43	This issue is not present
Issue 44	This issue is not present
Issue 45	This issue is not present
Issue 46	This issue is not present: parenthesis are used in the correct way
Issue 47	There isn't division
Issue 48	In this class there isn't integer arithmetic
Issue 49	This issue is not present
Issue 50	This issue is not present
Issue 51	There are implicit type conversion at lines: 126, 134, 143, 157, 170
Issue 52	This issue is not present
Issue 53	This issue is not present
Issue 54	There isn't switch statement
Issue 55	There isn't switch statement
Issue 56	There aren't loops
Issue 57	This class doesn't manage files
Issue 58	This class doesn't manage files
Issue 59	This class doesn't manage files
Issue 60	This class doesn't manage files

Table 3: Check issues from 41 to 60

4 Appendix

In this section, we will give the information about the used tools, the hours of work done by the members of the group.

4.1 Used Tools

In this phase of the project, the following tools have been used:

- LaTeX and TeXMaker editor: to redact and to format this document

4.2 Working Hours

Last Name	First Name	Total Hours
Blanco	Federica	30 h
Casasopra	Fabiola	2 h