



POLITECNICO
MILANO 1863

5 February 2017

PowerEnJoy

Code Inspection

Blanco	Federica	875487
Casasopra	Fabiola	864412

Software Engineering 2 Project
2016/2017

Contents

1	Introduction	1
1.1	Purpose	1
1.2	List of Definitions and Abbreviations	1
1.3	List of Reference Documents	1
1.4	Document overview	1
2	Coding Conventions and Issue Checklist	3
2.1	Coding Conventions	3
2.2	Issue Checklist	3
2.2.1	Naming Convention	3
2.2.2	Indentation	4
2.2.3	Braces	4
2.2.4	File Organization	4
2.2.5	Wrapping Lines	4
2.2.6	Comments	5
2.2.7	Java Source File	5
2.2.8	Package and Import Statements	5
2.2.9	Class and Interface Declarations	5
2.2.10	Initialization and Declarations	6
2.2.11	Method Calls	6
2.2.12	Arrays	6
2.2.13	Object Comparison	7
2.2.14	Output Format	7
2.2.15	Computation, Comparisons and Assignments	7
2.2.16	Exceptions	7
2.2.17	Flow of Control	8
2.2.18	Files	8
3	Code Inspection Process	9
3.1	Assigned Classes	9
3.2	Functional Role of the Assigned Set of Classes	9
3.3	List of Issue	9
4	Appendix	10
4.1	Used Tools	10
4.2	Working Hours	10

List of Figures

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 analyze 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 developers 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 their 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 acronyms 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 structure 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 compiler, 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, 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.

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

3.2 Functional Role of the Assigned Set of Classes

3.3 List of Issue

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	? h
Casasopra	Fabiola	? h