**Politecnico di Milano**

**Software Engineering 2**

**CODE INSPECTION**

**PowerEnjoy**

**Authors:**

**Simone Bruzzechesse**

**Luca Franceschetti**

**Gian Giacomo Gatti**

# Index

[Index 2](#_Toc470617697)

[1 Introduction 3](#_Toc470617698)

[1.1 Purpose 3](#_Toc470617699)

[1.2 Assigned Classes 3](#_Toc470617700)

[1.3 Functional role of assigned set of classes 3](#_Toc470617701)

[1.4 Reference Documents 3](#_Toc470617702)

[1.5 Notation 3](#_Toc470617703)

[1.6 Document Structure 4](#_Toc470617704)

[2 List of issues 5](#_Toc470617705)

[2.1 Naming Conventions 5](#_Toc470617706)

[2.2 Indention 5](#_Toc470617707)

[2.3 Braces 5](#_Toc470617708)

[2.4 File Organization 6](#_Toc470617709)

[2.5 Wrapping Lines 6](#_Toc470617710)

[2.6 Comments 6](#_Toc470617711)

[2.7 Java Source Files 6](#_Toc470617712)

[2.8 Package and Import Statements 6](#_Toc470617713)

[2.9 Class and Interface Declarations 7](#_Toc470617714)

[2.10 Inizialization and Declarations 7](#_Toc470617715)

[2.11 Method Calls 7](#_Toc470617716)

[2.12 Arrays 8](#_Toc470617717)

[2.13 Object Comparison 8](#_Toc470617718)

[2.14 Output Format 8](#_Toc470617719)

[2.15 Computation, Comparisons and Assignments 8](#_Toc470617720)

[2.16 Exceptions 9](#_Toc470617721)

[2.17 Flow of Control 9](#_Toc470617722)

[2.18 Files 9](#_Toc470617723)

[3 Other Problems 10](#_Toc470617724)

# Introduction

## Purpose

Code inspection is the systematic examination (often known as peer review) of computer source code. It is intended to find mistakes overlooked during the initial development phase, with the aim of improving both the overall quality of software and the developers' skills. We 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 Apache OFBiz project, 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.

Our scope is to perform the inspection reporting on the quality status of selected code extracts using the checklist for Java code inspection.

## Assigned Classes

## Functional role of assigned set of classes

## Reference Documents

* Our RASD document
* Specification Document: Assignments AA 2016-2017.pdf
* Structure of the design document.pdf
* Sample Design Deliverable Discussed on Nov. 2

## Notation

* A specific line of code will be referred as follows: L.123.
* An interval of lines of code will be referred as follows: L.123-456.

## Document Structure

# List of issues

## 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.

L.504 variable otherCurrencyUomId: is not clear what Uom stand for

L.528 acctgTransEntries not so clear

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.

## 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).

L.500 no blank line between two methods

L.359-370 no blank lines between methods

13. Where practical, line length does not exceed 80 characters.

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

L.291 is 134 characters

L.296 is 136 characters

L.430 is 125 characters.

L.496 is 123 characters

L.507 is 134 characters.

L.545 is 124 characters.

L.552 is 129 characters

Other problems: line 507 and 509 there are two spaces between variable name and ‘=’.

## Wrapping Lines

15. Line break occurs after a comma or an operator.

EntityConditionList<EntityExpr> dateCondition = EntityCondition.makeCondition(UtilMisc.toList(

EntityCondition.makeCondition("effectiveDate", EntityOperator.EQUALS, null),

L.419

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).

L.501 🡪 method not explained

L.567 🡪 methods are not explained

## 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:

1. class/interface documentation comment;
2. class or interface statement;
3. class/interface implementation comment, if necessary;
4. class (static) variables;
   * 1. first public class variables;
     2. next protected class variables;
     3. next package level (no access modi\_er);
     4. last private class variables.
5. instance variables;
   * 1. first public instance variables;
     2. next protected instance variables;
     3. next package level (no access modi\_er);
     4. last private instance variables.
6. constructors;
7. 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.

## Inizialization 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.

L.401 variables are declared after an IF statement.

## Method Calls

34. Check that parameters are presented in the correct order.

L.401 fourth parameter is not presented

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 o\_-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.

isNotEmpty() method is used as well as for statement in the form for(Object o : Objects)

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 ==.

L.532 if (origAmount.compareTo(ZERO) == 1) {

## 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.

L.403 maybe is better to specify which method.

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 actions are taken for each catch block.

## Flow of Control

54. In a switch statement, check that all cases are addressed by break or return.

No switch statements

55. Check that all switch statements have a default branch.

No switch statements

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.

No files

58. Check that all files are closed properly, even in the case of an error.

No files

59. Check that EOF conditions are detected and handled correctly.

No files

60. Check that all file exceptions are caught and dealt with accordingly.

No files

# Other Problems

# Information

## Used Tools

## Effort

## References