**Politecnico di Milano**

**Software Engineering 2**

**CODE INSPECTION**

**PowerEnjoy**

**Authors:**

**Simone Bruzzechesse**

**Luca Franceschetti**

**Gian Giacomo Gatti**

# Index

[Index 2](#_Toc471469809)

[1 Introduction 4](#_Toc471469810)

[1.1 Purpose 4](#_Toc471469811)

[1.2 Table of Content 4](#_Toc471469812)

[1.2.1 Suggested Structure 4](#_Toc471469813)

[1.2.2 Document Structure 5](#_Toc471469814)

[1.3 Project and Assigned Classes 6](#_Toc471469815)

[1.3.1 Apache OFBiz 6](#_Toc471469816)

[1.3.2 Assigned Class 6](#_Toc471469817)

[1.4 Functional role of assigned set of classes 6](#_Toc471469818)

[1.5 Reference Documents 6](#_Toc471469819)

[1.6 Notation 6](#_Toc471469820)

[2 List of issues 7](#_Toc471469821)

[2.1 Naming Conventions 7](#_Toc471469822)

[2.2 Indention 7](#_Toc471469823)

[2.3 Braces 7](#_Toc471469824)

[2.4 File Organization 7](#_Toc471469825)

[2.5 Wrapping Lines 8](#_Toc471469826)

[2.6 Comments 8](#_Toc471469827)

[2.7 Java Source Files 8](#_Toc471469828)

[2.8 Package and Import Statements 9](#_Toc471469829)

[2.9 Class and Interface Declarations 9](#_Toc471469830)

[2.10 Initialization and Declarations 9](#_Toc471469831)

[2.11 Method Calls 10](#_Toc471469832)

[2.12 Arrays 10](#_Toc471469833)

[2.13 Object Comparison 10](#_Toc471469834)

[2.14 Output Format 11](#_Toc471469835)

[2.15 Computation, Comparisons and Assignments 11](#_Toc471469836)

[2.16 Exceptions 11](#_Toc471469837)

[2.17 Flow of Control 11](#_Toc471469838)

[2.18 Files 12](#_Toc471469839)

[3 Other Problems 13](#_Toc471469840)

[4 References 14](#_Toc471469841)

[4.10 Used Tools 14](#_Toc471469842)

[4.11 Effort 14](#_Toc471469843)

[4.12 References 14](#_Toc471469844)

# 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 for 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. This checklist includes, for example, some techniques for code evaluation: we can check the quality of the source code through some aesthetics aspect, like code indentation, brackets stile, comments, wrapping lines or naming conventions. Moreover, we can turn our attention on other detail (not aesthetics this time) like class and interface declaration, arrays utilization, object comparisons, and so on. We will see the whole set of checklist in the specific section, later in the document.

## Table of Content

### Suggested Structure

1. **Front** **page**: Include at least the project title, the version of the document, your names and the release date.
2. **Table of content**: Include the table of content of your document.
3. **Classes that were assigned to the group**
4. **Functional role** of assigned set of classes
5. **List of issues** found by applying the checklist
6. Any other problem you have highlighted

### Document Structure

|  |  |
| --- | --- |
| Introduction | 1. Purpose 2. Table of Content 3. Project and Assigned Class 4. Functional role of Class 5. Reference documents 6. Notation |
| List of Issues | 1. Checklist    1. Naming Convention    2. Indention    3. Braces    4. File Organization    5. Wrapping Lines    6. Comments    7. Java Source File    8. Package and Import statements    9. Class declaration    10. Initialization and declaration    11. Method Calls    12. Arrays    13. Object Comparisons    14. Output Format    15. Computation, Comparisons and Assignment    16. Exception    17. Flow of Control    18. Files |
| Other Problems |  |
| References | 1. Used Tools 2. Effort 3. References |

## Project and Assigned Classes

### Apache OFBiz

Download link: <http://mirror.nohup.it/apache/ofbiz/apache-ofbiz-16.11.01.zip>

Documentation: <https://ofbiz.apache.org/documentation.html>



### Assigned Class

|  |  |
| --- | --- |
| Name | InvoiceWorker |
| Path | ../apache-ofbiz-16.11.01/  applications/accounting/src/main/java/org/apache/ofbiz/  accounting/invoice/InvoiceWorker.java |
| Total Lines | **757** |

## Functional role of assigned set of classes

**//TODO together**

## 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.

# List of issues

## Naming Conventions

* **L.504**: variable otherCurrencyUomId: is not clear what Uom stand for
* **L.528**: acctgTransEntries not so clear
* **L.130**: Maybe “entry set” it’s not a verb.
* First occurrence **L.304**: “NowTimestamp”, but it’s an external class from   
  **import** java.sql.Timestamp;
* **L.52-57:** excluding the “ZERO” variable

## Indention

* **Three or four spaces** are used for indentation
* **No tabs** are used to indent

## Braces

* **“Kernighan and Ritchie”** style (first brace is on the same line of the instruction that opens the new block).
* **L.220**
* **L.263**
* **L.305**
* **L.307**
* **L.657**

## File Organization

* **L.359-370:** no blank lines between methods
* **L.500:** no blank line between two methods
* 80 characters are exceeded a lot of time

##### 

* **L.1-350**: it does exceed 120 characters at least 10 time
* **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
* **L.615-756:** it does exceed 120 characters at least 11 times

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

## Wrapping Lines

* **L.119:** line breaks after a dot
* **L.170-171-172:** line breaks after a bracket
* **L.310**
* **L.320**
* **L.410**
* **L.419**
* **L.662**
* **L.663**
* **L.664**
* **L.700**
* **L.701**
* **L.702**
* **L.703**
* **L.721**
* **L.722**
* **L.723**
* **L.665**
* **L.702**
* **L.703**
* **L.704**
* **L.722**
* **L.723**
* **L.724**

## Comments

* **L.689:** inthe method “getInvoiceTaxTotalForTaxAuthPartyAndGeo” the comment doesn’t explain the function’s role.
* There aren’t dates to indicate when it can be removed from the source file if determined it is no longer needed.

## Java Source Files

* No javadoc documentation for:
  + “getInvoiceTaxTotal”
  + “getInvoiceNoTaxTotal”
  + “getInvoiceAddressByType”
  + “getInvoiceCurrencyConversionRate”
  + “getTaxTotalForInvoiceItems”

## Package and Import Statements

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

Checklist respected in this project.

## Class and Interface Declarations

* “getInvoiceTotal” is not grouped well

All this classes are duplicate. We can evince from the code that the duplicates are “override” method for different situation where some parameters are required or not. Moreover, those methods differ only for initial parameter most of time, and not for internal code. We provide a list:

* “GetInvoiceTotal”
* “getInvoiceAddressByType”
* “getInvoiceNotApplied”
* “getInvoiceApplied”
* “getInvoiceItemApplied”
* “getInvoiceCurrencyConversionRate”

Moreover, the method “getInvoiceTaxByTaxAuthGeoAndParty” is probably too complex since it involves almost 70 lines of code.

## Initialization and Declarations

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

* **L.594**
* **L.642:** in these two lines we have a declaration of a variable which is a “Map<String, Object>” but instead of Object in the second position there should be “BigDecimal” since it should contains BigDecimal and not Objects in general.

**Other problem:** in the line 635 there is an error since the point comes before the declaration of the type of the variables so the order of the declaration and the method calls is not correct, this is an error.

* All objects are initialized before use, but sometimes the initialization is to NULL. It’s not problematic but, if we won’t use that variable during the execution (and it remain equal to NULL) at the end, maybe, we cannot understand completely the flow of the program. We report this as a “warning”.
* All objects are initialized before use, but sometimes the initialization is to NULL. It’s not problematic but, if we won’t use that variable during the execution (and it remain equal to NULL) at the end, maybe, we cannot understand completely the flow of the program. We report this as a “warning”.
* **L.85**
* **L.109**
* **L.205**
* **L.248**
* **L.331-332**
* **L.401**
* **L.642**
* **L.740**

## Method Calls

* **L.401:** fourth parameter is not presented
* As we can see from the documentation and from the source code, the correct method is being called everywhere in “InvoiceWorker” class.
* From line **1** to **350** almost every function return “BigDecimal” or “GenericValue”. Where there is a calling chain between function, the whole return statements are correct.

## Arrays

* Not accessed through index, correctly accessed through “*for*” statement
* No array indexes, only list surrounded by try-catch statements
* isNotEmpty() method is used as well as “*for*” statement in the form   
  for(Object o : Objects)

## Object Comparison

* **L.81**
* **L.92**
* **L.106**
* **L.110**
* **L.532**

## Output Format

* Error message are too restrictive and does not contains any guidance to problem resolution
* **L.403**: maybe is better to specify which method, for example.
* **L**.**667:** in the output of the error condition we don’t refer to InvoiceItem but to InvoiceTaxItems.
* **L**.**706:** the same of before
* **L.726:** the same of before

## Computation, Comparisons and Assignments

**//TODO BETTER**

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 legitimate.

51. Check that the code is free of any implicit type conversions.

## Exceptions

* The appropriate action isn’t taken in the catch block because the programmers actions is always to debug the error in a “logError” object. Never retry to do the statement after some changes or some new input. Debugging is correct, but maybe it’s better doing some new action that can execute successfully the “try” statement

## Flow of Control

* No switch statements
* No switch statements
* All the “for” statements are constructed in the “for-each” form. The “itemCollection” used for iterate is always well declared. At line 178 there is a flow control where we check if “invoiceItems” is not NULL.

## Files

* No files are declared, opened, closed or generally handled in this class.

# Other Problems

# References

## Used Tools

## Effort

## References