Politecnico di Milano

Software Engineering 2

CODE INSPECTION

PowerEnjoy

Authors:

Simone Bruzzechesse

Luca Franceschetti

Gian Giacomo Gatti

INDEX

Index 2

1 Introduction 4

- 1.1 Purpose 4
- 1.2 Table of Content 4
  - 1.2.1 Suggested Structure 4
  - 1.2.2 Document Structure 5
- 1.3 Project and Assigned Classes 6
  - 1.3.1 Apache OFBiz 6
  - 1.3.2 Assigned Class 6
- 1.4 Functional role of assigned set of classes 6
- 1.5 Reference Documents 8
- 1.6 Notation 8

2 List of issues 9

- 2.1 Naming Conventions 9
- 2.2 Indention 9
- 2.3 Braces 9
- 2.4 File Organization 9
- 2.5 Wrapping Lines 10
- 2.6 Comments 11
- 2.7 Java Source Files 11
- 2.8 Package and Import Statements 11
- 2.9 Class and Interface Declarations 11
- 2.10 Initialization and Declarations 12
- 2.11 Method Calls 13
- 2.12 Arrays 13
- 2.13 Object Comparison 13
- 2.14 Output Format 13
- 2.15 Computation, Comparisons and Assignments 14
- 2.16 Exceptions 14
- 2.17 Flow of Control 15

2.18 Files 15
3 Other Problems 16
4 References 17
4.10 Used Tools 17
4.11 Hours Of Work 17
4.12 References 17
4.12.1 Apache OFBiz 17
4.12.2 “Brutish Programming” Hints 17

1 Introduction

1.1 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.

1.2 Table of Content

1.2.1 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 we have highlighted

1.2.2 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	2. Checklist 2.1. Naming Convention 2.2. Indention 2.3. Braces 2.4. File Organization 2.5. Wrapping Lines 2.6. Comments 2.7. Java Source File 2.8. Package and Import statements 2.9. Class declaration 2.10. Initialization and declaration 2.11. Method Calls 2.12. Arrays 2.13. Object Comparisons 2.14. Output Format 2.15. Computation, Comparisons and Assignment 2.16. Exception 2.17. Flow of Control 2.18. Files
Other Problems	List of other problems
References	1. Used Tools 2. Effort 3. References

1.3 Project and Assigned Classes

1.3.1 Apache OFBiz

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

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

1.3.2 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**

1.4 Functional Role of Assigned Set of Classes

Our class is a module of "*accounting*" section of the Ofbiz Project. More precisely, as we can see on official documentation of the project, the "**Accounting Manager Application**" is divided in some modules. In the "*invoices*" tab, the user can create a new invoice or search an old one. On the bottom of the page are shown the results of the eventual research. Main core of the class we inspect is to obtain invoice from some parameters given in input. All possible function, with relative parameters, are listed below.

We provide a list of methods composing our class "*Invoice Workers*" in order to prove its functionalities:

`getBillFromParty(GenericValue invoice)`

Convenience method to obtain the bill from party for an invoice.

`getBillToAddress(GenericValue invoice)`

Method to obtain the billing address for an invoice `getBillToParty(GenericValue invoice)`

Method to obtain the bill to party for an invoice.

`getInvoiceApplied(Delegator delegator,java.lang.String invoiceId)`

Method to return the total amount of an invoice which is applied to a payment

`getInvoiceApplied(Delegator delegator, java.lang.String invoiceId, java.sql.Timestamp asOfDateTime, java.lang.Boolean actualCurrency)`

Returns amount applied to invoice before an asOfDateTime, based on Payment.effectiveDate <= asOfDateTime

`getInvoiceApplied(GenericValue invoice)`

Method to return the total amount of an invoice which is applied to a payment

`getInvoiceApplied(GenericValue invoice, java.lang.Boolean actualCurrency)`

Return the amount applied to the invoice

`getInvoiceItemApplied(Delegator delegator, java.lang.String invoiceId, java.lang.String invoiceItemSeqId)`

Method to return the amount of an invoiceItem which is applied to a payment

`getInvoiceItemTotal(GenericValue invoiceItem)`

Method to return the total amount of an invoice item i.e. quantity * amount

`getInvoiceNotApplied(Delegator delegator, java.lang.String invoiceId, java.lang.Boolean actualCurrency)`

Method to return the total amount of an invoice which is not yet applied to a payment

`getInvoiceNotApplied(GenericValue invoice, java.sql.Timestamp asOfDateTime)`

Returns amount not applied (i.e., still outstanding) of an invoice at an asOfDate, based on Payment.effectiveDate <= asOfDateTime

`getInvoiceTaxAuthPartyAndGeos(GenericValue invoice)`

Returns a List of the TaxAuthority Party and Geos for the given Invoice.

`getInvoiceTotal(Delegator delegator, java.lang.String invoiceId)`
Return the total amount of the invoice (including tax) using the the invoiceId as input.

`getInvoiceUnattributedTaxTotal(GenericValue invoice)`
Returns the invoice tax total for unattributed tax items, that is items which have no taxAuthPartyId value.

`getSendFromAddress(GenericValue invoice)`
Method to obtain the sending address for an invoice

`getSendFromParty(GenericValue invoice)`
Method to obtain the send from party for an invoice

`getTaxableInvoiceItemTypeIds(Delegator delegator)`
Method to get the taxable invoice item types as a List of invoiceItemTypeIds.

1.5 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

1.6 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.

2 LIST OF ISSUES

2.1 NAMING CONVENTIONS

1.

- **L.504**: variable otherCurrencyUomId: is not clear what Uom stand for
- **L.528**: acctgTransEntries not so clear

5.

- **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;`

7.

- **L.52-57:** excluding the “ZERO” variable

2.2 INDENTION

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

2.3 BRACES

10.

- **“Kernighan and Ritchie”** style (first brace is on the same line of the instruction that opens the new block).

11.

- **L.220**
- **L.263**
- **L.305**
- **L.307**
- **L.657**

2.4 FILE ORGANIZATION

12.

- **L.359-370:** no blank lines between methods
- **L.500:** no blank line between two methods

13.

- 80 characters are exceeded a lot of time

14.

- **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 '='.

2.5 Wrapping Lines

15.

- **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**

17.

- **L.665**
- **L.702**
- **L.703**
- **L.704**
- **L.722**
- **L.723-724**

2.6 Comments

18.

- **L.689:** in the method “getInvoiceTaxTotalForTaxAuthPartyAndGeo” the comment doesn’t explain the function’s role.

19.

- There aren’t dates to indicate when it can be removed from the source file if determined it is no longer needed.

2.7 Java Source Files

23.

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

2.8 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.

2.9 Class and Interface Declarations

26.

- “getInvoiceTotal” is not grouped well

27.

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.

2.10 INITIALIZATION AND DECLARATIONS

28.

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.

31.

- 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”.

32.

- 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”.

33.

- **L.85**
- **L.109**
- **L.205**
- **L.248**
- **L.331-332**
- **L.401**
- **L.642**
- **L.740**

2.11 Method Calls

34.

- **L.401:** fourth parameter is not presented

35.

- As we can see from the documentation and from the source code, the correct method is being called everywhere in “InvoiceWorker” class.

36.

- 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.

2.12 Arrays

37.

- Not accessed through index, correctly accessed through “*for*” statement

38.

- 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)*

2.13 Object Comparison

40.

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

2.14 Output Format

42.

- 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

2.15 Computation, Comparisons and Assignments

21.

- From line **1** to **350** “*Brutish Programming*” is avoided, for example, when programmers have to check if an object is empty or not, they correctly use “.isEmpty”: `if (UtilValidate.isEmpty(locations)){}.`
- From line **615** to the end “*Brutish Programming*” is also avoided since, for example, to check that an element is not empty (like in line 674) it is used: “.isNotEmpty()”;

47.

- From line **615** to the end there are not divisions.

48.

- From line **615** to the end there are not divisions and all the operations are between variables that are already initialized.

49.

- From line **615** to the end there are only comparisons between the amount (a BigDecimal) and null in order to check that the variable is not null and in that case to initialize it so they are correct.

2.16 Exceptions

53.

- 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

2.17 Flow of Control

54.

- No switch statements

55.

- No switch statements

56.

- 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.

2.18 Files

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

3 Other Problems

- 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.

4 References

4.10 Used Tools

- Microsoft Office **Word**
- **Eclipse** IDE

4.11 Hours Of Work

We report approximately how many hours each member has worked on this document.

- Simone Bruzzechesse: 10 hours
- Luca Franceschetti: 10 hours
- Gian Giacomo Gatti: 10 hours

4.12 References

4.12.1 Apache OFBiz

- **Download** link: http://mirror.nohup.it/apache/ofbiz/apache-ofbiz-16.11.01.zip
- **Documentation**: https://ofbiz.apache.org/documentation.html
- **Invoice** Page: https://cwiki.apache.org/confluence/display/OFBIZ/02+Invoices

4.12.2 "Brutish Programming" Hints

- http://users.csc.calpoly.edu/~jdalbey/SWE/CodeSmells/bonehead.html