Politecnico di Milano

Computer Science and Engineering's master degree course

Department of Electronics and Information

Apache OFBiz
Code Inspection

Version 1.0

Release Date: 05/02/2017

Customer: Eng. Elisabetta DI NITTO

Authors:

Eng. Marco FERNI	Id. 877712
Eng. Angelo Claudio RE	Id. 877808
Eng. Gabriele TERMIGNONE	Id. 877645

Contents

Chapter 1

Introduction

1.1 Purpose

The purpose of this document is to show all the problems found during the inspection of a small amount of code of a specific version of Apache OFBiz. The process of inspecting source code has two main purposes. The first and most obvious one is to enhance the quality of the code and eventually to identify the remaining bugs. The second purpose is to improve the coding skills of the team. The inspectors improve themselves by analysing the code made by others and eventually discovering coding methods that they did not know. The original authors of the code receive a list of possible mistakes that they could have done, which of course help them improving themselves.

Each group of the project has different methods assigned of a specific version of Apache OFBiz. We have to analyse these methods by checking that they are in agreement with every point of a given checklist. We also have to find other problems, then we have to report the problems found in this document.

1.2 Scope

Apache OFBiz is the official implementation of JEE. It is an open source project that uses svn as version system, in fact we used it to retrieve a specific version of Apache OFBiz 16.11.01 (of 27 Nov 2016).
This version is the version required by the assignment since we have been assigned some of its methods to check.
Apache OFBiz is a maven project, in fact we imported the java file into intellij IDEA and we used it, SonarLint and JUnit to verify some checks of the checklist.

1.3 List of Definitions and Abbreviations

- *JEE* Java enterprise edition
- *Context:* Contextual Information: it is a design pattern that consists in storing the main information inside one object and this object is used to pass everything.
- *Servlet:* A java program that runs on a dedicated web server that is able to elaborate http requests and reply to them.
- *Guest:* a person who is not already registered to the system.
- *User:* a registered customer.
- *DD:* Design Document
- *DB:* DataBase
- *RASD:* Requirement Analysis and Specification Document
- *ITPD:* Integration Test Plan Document
- *CI:* Code Inspection Document
- *SVN:* OFBiz public Subversion repository.
- *Maven:* a software used primarily for Java project0s management and build automation,
- *POM:* Project Object Model, it's the configuration file that contains all the details of a project (dependancies, structure, plugins,..)
- *Intellij IDEA:* a IDE (Integratd Development Enviroment)
- *SonarLint:* an open source platform used by development teams to manage source code quality.

1.4 List of Reference Documents

- Code Inspection Assignment Task Description.pdf
- Apache OFBiz docs
- CI assignments specification
- Document of all assignments classes
- The PowerEnjoy's RASD
- PowerEnjoy's DD
- The project's Assignments PDF
- Old years projects
- The ITPD standards
- Software Engineering course slides
- Specific tools' tutorials and documentation

1.5 Document Structure

- **Introduction:** this section introduces the CI document. It contains a justification of its utility and indications on which parts are covered in this document.
- **Classes:** this section describes the classes and the methods that have been inspected.
- **Functional Role:** this section describes the functional role of the class which methods assigned belong to.
- **List of issues found by applying the checklist:** this section describes the issues found applying the checklist given.
- **Other problems:** this section describes other problems found that are not strictly related to the checklist.

Chapter 2

Classes

All methods assigned to us belong to the same class.

2.1 StandardContext

- **Namespace:** org.apache.ofbiz.accounting.thirdparty.sagepay
- **Extends:** /
- **Implements:** /

2.1.1 Attributes

- Line 47: module
- Line 48: resource

2.1.2 Methods

- Line 50: Map<String, String> buildCustomerBillingInfo(Map<String, Object> context)
- Line 150: Map<String, Object> ccAuth(DispatchContext dctx, Map<String, Object> context)
- Line 169: Map<String, Object> processCardAuthorisationPayment(DispatchContext ctx, Map<String, Object> context)
- Line 236: Map<String, Object> ccCapture(DispatchContext ctx, Map<String, Object> context)

- Line 250: Map<String, Object> processCardCapturePayment(DispatchContext ctx, Map<String, Object> context)
- Line 291: Map<String, Object> ccRefund(DispatchContext ctx, Map<String, Object> context)
- Line 353: Map<String, Object> processCardRefundPayment(DispatchContext ctx, Map<String, Object> context)
- Line 402: Map<String, Object> processCardVoidPayment(DispatchContext ctx, Map<String, Object> context)
- Line 448: Map<String, Object> ccRelease(DispatchContext ctx, Map<String, Object> context)
- Line 465: Map<String, Object> processCardReleasePayment(DispatchContext ctx, Map<String, Object> context)

Chapter 3

Functional Role

The class to review is part of the "SagePay" implementation, which is a third-party service used to handle electronic payments; in particular, this specific implementation manages the online payments performed with credit cards.
The class provides four public method, called:

- ccAuth
- ccCapture
- ccRefund
- ccRelease

The purpose of the class is to check if the context in which it is running satisfies the conditions to:

- proceed with the authentication of the credit card
- proceed with a payment action
- proceed with a refund action
- release the credit card

Chapter 4

List of issues found by applying the checklist

4.1 NamingConventions

- Line 150: method ***ccAuth*** should start with a verb (Hint: ***authorizeCreditCard***).
- Line 236: method ***ccCapture*** should start with a verb (Hint: ***captureCreditCard***).
- Line 291: method ***ccRefund*** should start with a verb (Hint: ***refundCreditCard***).
- Line 448: method ***ccRelease*** should start with a verb (Hint: ***releaseCreditCard***).

4.2 Indention

Four spaces are used for indentation.

- Line 157 start with a mismatching number of spaces (8 instead of 0).

4.3 Braces

Everything ok.

Kernighan and Ritchie style is used.

4.4 File Organization

- Line 1 can be easily rewritten not to exceed 80 columns.
- Line 18 can be easily rewritten not to exceed 80 columns.
- Line 50 can be easily rewritten not to exceed 80 columns.
- Line 51 can be easily rewritten not to exceed 80 columns.
- Line 52 can be easily rewritten not to exceed 80 columns.
- Line 69 can be easily rewritten not to exceed 80 columns.
- Line 71 can be easily rewritten not to exceed 80 columns.
- Line 73 can be easily rewritten not to exceed 80 columns.
- Line 74 can be easily rewritten not to exceed 120 columns.
- Line 81 can be easily rewritten not to exceed 80 columns.
- Line 87 can be easily rewritten not to exceed 80 columns.
- Line 92 can be easily rewritten not to exceed 80 columns.
- Line 119 can be easily rewritten not to exceed 80 columns.

- Line 124 can be easily rewritten not to exceed 80 columns.
- Line 128 can be easily rewritten not to exceed 120 columns.
- Line 150 can be easily rewritten not to exceed 80 columns.
- Line 156 can be easily rewritten not to exceed 80 columns.
- Line 159 can be easily rewritten not to exceed 120 columns.
- Line 169 can be easily rewritten not to exceed 120 columns.
- Line 174 can be easily rewritten not to exceed 80 columns.
- Line 178 can be easily rewritten not to exceed 80 columns.
- Line 191 can be easily rewritten not to exceed 80 columns.
- Line 192 can be easily rewritten not to exceed 80 columns.
- Line 195 can be easily rewritten not to exceed 80 columns.
- Line 197 can be easily rewritten not to exceed 80 columns.
- Line 207 can be easily rewritten not to exceed 80 columns.
- Line 208 can be easily rewritten not to exceed 120 columns.
- Line 210 can be easily rewritten not to exceed 120 columns.
- Line 214 can be easily rewritten not to exceed 80 columns.
- Line 215 can be easily rewritten not to exceed 120 columns.
- Line 217 can be easily rewritten not to exceed 80 columns.
- Line 218 can be easily rewritten not to exceed 120 columns.
- Line 220 can be easily rewritten not to exceed 80 columns.

- Line 221 can be easily rewritten not to exceed 120 columns.
- Line 223 can be easily rewritten not to exceed 80 columns.
- Line 224 can be easily rewritten not to exceed 120 columns.
- Line 226 can be easily rewritten not to exceed 80 columns.
- Line 227 can be easily rewritten not to exceed 120 columns.
- Line 230 can be easily rewritten not to exceed 80 columns.
- Line 231 can be easily rewritten not to exceed 120 columns.
- Line 237 can be easily rewritten not to exceed 80 columns.
- Line 239 can be easily rewritten not to exceed 80 columns.
- Line 240 can be easily rewritten not to exceed 80 columns.
- Line 250 can be easily rewritten not to exceed 80 columns.
- Line 254 can be easily rewritten not to exceed 80 columns.
- Line 255 can be easily rewritten not to exceed 80 columns.
- Line 264 can be easily rewritten not to exceed 80 columns.
- Line 274 can be easily rewritten not to exceed 80 columns.
- Line 278 can be easily rewritten not to exceed 80 columns.
- Line 279 can be easily rewritten not to exceed 120 columns.
- Line 281 can be easily rewritten not to exceed 80 columns.
- Line 282 can be easily rewritten not to exceed 120 columns.
- Line 285 can be easily rewritten not to exceed 80 columns.

- Line 286 can be easily rewritten not to exceed 120 columns.
- Line 291 can be easily rewritten not to exceed 80 columns.
- Line 295 can be easily rewritten not to exceed 80 columns.
- Line 296 can be easily rewritten not to exceed 80 columns.
- Line 298 can be easily rewritten not to exceed 120 columns.
- Line 300 can be easily rewritten not to exceed 80 columns.
- Line 303 can be easily rewritten not to exceed 80 columns.
- Line 305 can be easily rewritten not to exceed 120 columns.
- Line 306 can be easily rewritten not to exceed 120 columns.
- Line 311 can be easily rewritten not to exceed 80 columns.
- Line 313 can be easily rewritten not to exceed 80 columns.
- Line 314 can be easily rewritten not to exceed 80 columns.
- Line 316 can be easily rewritten not to exceed 80 columns.
- Line 318 can be easily rewritten not to exceed 80 columns.
- Line 327 can be easily rewritten not to exceed 80 columns.
- Line 338 can be easily rewritten not to exceed 80 columns.
- Line 353 can be easily rewritten not to exceed 80 columns.
- Line 357 can be easily rewritten not to exceed 80 columns.
- Line 358 can be easily rewritten not to exceed 80 columns.
- Line 366 can be easily rewritten not to exceed 80 columns.

- Line 373 can be easily rewritten not to exceed 80 columns.
- Line 374 can be easily rewritten not to exceed 80 columns.
- Line 375 can be easily rewritten not to exceed 80 columns.
- Line 376 can be easily rewritten not to exceed 80 columns.
- Line 379 can be easily rewritten not to exceed 80 columns.
- Line 387 can be easily rewritten not to exceed 80 columns.
- Line 388 can be easily rewritten not to exceed 120 columns.
- Line 390 can be easily rewritten not to exceed 80 columns.
- Line 391 can be easily rewritten not to exceed 120 columns.
- Line 396 can be easily rewritten not to exceed 120 columns.
- Line 402 can be easily rewritten not to exceed 80 columns.
- Line 406 can be easily rewritten not to exceed 80 columns.
- Line 407 can be easily rewritten not to exceed 80 columns.
- Line 412 can be easily rewritten not to exceed 80 columns.
- Line 415 can be easily rewritten not to exceed 80 columns.
- Line 417 can be easily rewritten not to exceed 80 columns.
- Line 422 can be easily rewritten not to exceed 80 columns.
- Line 428 can be easily rewritten not to exceed 80 columns.
- Line 429 can be easily rewritten not to exceed 120 columns.
- Line 431 can be easily rewritten not to exceed 80 columns.

- Line 432 can be easily rewritten not to exceed 120 columns.
- Line 434 can be easily rewritten not to exceed 80 columns.
- Line 435 can be easily rewritten not to exceed 120 columns.
- Line 437 can be easily rewritten not to exceed 80 columns.
- Line 438 can be easily rewritten not to exceed 120 columns.
- Line 443 can be easily rewritten not to exceed 120 columns.
- Line 448 can be easily rewritten not to exceed 80 columns.
- Line 452 can be easily rewritten not to exceed 80 columns.
- Line 454 can be easily rewritten not to exceed 80 columns.
- Line 456 can be easily rewritten not to exceed 120 columns.
- Line 465 can be easily rewritten not to exceed 80 columns.
- Line 470 can be easily rewritten not to exceed 80 columns.
- Line 473 can be easily rewritten not to exceed 80 columns.
- Line 478 can be easily rewritten not to exceed 80 columns.
- Line 489 can be easily rewritten not to exceed 80 columns.
- Line 495 can be easily rewritten not to exceed 80 columns.
- Line 496 can be easily rewritten not to exceed 120 columns.
- Line 498 can be easily rewritten not to exceed 80 columns.
- Line 499 can be easily rewritten not to exceed 120 columns.
- Line 503 can be easily rewritten not to exceed 80 columns.
- Line 504 can be easily rewritten not to exceed 120 columns.

4.5 Wrapping Lines

- Line 74 higher-level break isn't used.
- Line 128 higher-level break isn't used.
- Line 159 higher-level break isn't used.
- Line 169 higher-level break isn't used.
- Line 208 higher-level break isn't used.
- Line 210 higher-level break isn't used.
- Line 214 higher-level break isn't used.
- Line 215 higher-level break isn't used.
- Line 217 higher-level break isn't used.
- Line 218 higher-level break isn't used.
- Line 220 higher-level break isn't used.
- Line 221 higher-level break isn't used.
- Line 223 higher-level break isn't used.
- Line 224 higher-level break isn't used.
- Line 226 higher-level break isn't used.
- Line 227 higher-level break isn't used.
- Line 231 higher-level break isn't used.
- Line 279 higher-level break isn't used.
- Line 281 higher-level break isn't used.

- Line 282 higher-level break isn't used.
- Line 286 higher-level break isn't used.
- Line 298 higher-level break isn't used.
- Line 305 higher-level break isn't used.
- Line 306 higher-level break isn't used.
- Line 313 higher-level break isn't used.
- Line 327 higher-level break isn't used.
- Line 338 higher-level break isn't used.
- Line 338 higher-level break isn't used.
- Line 388 higher-level break isn't used.
- Line 390 higher-level break isn't used.
- Line 391 higher-level break isn't used.
- Line 396 higher-level break isn't used.
- Line 429 higher-level break isn't used.
- Line 432 higher-level break isn't used.
- Line 435 higher-level break isn't used.
- Line 438 higher-level break isn't used.
- Line 443 higher-level break isn't used.
- Line 456 higher-level break isn't used.
- Line 496 higher-level break isn't used.

- Line 498 higher-level break isn't used.
- Line 499 higher-level break isn't used.
- Line 504 higher-level break isn't used.

4.6 Comments

Class methods aren't commented. In fact, there are no comment at all in the file.

4.7 Java Source Files

Everything ok. This is the link to the javadoc page for the assigned class:
[Class SagePayPaymentServices Javadoc]

4.8 Package and Import Statements

Everything ok

4.9 Class and Interface Declarations

Everything ok

4.10 Initialization and Declarations

Everything ok

4.11 Method Calls

Everything ok

4.12 Arrays

Everything ok

4.13 Object Comparison

Everything ok

4.14 Output Format

Everything ok

4.15 Computation, Comparisons and Assignments

Everything ok

4.16 Exceptions

Everything ok

4.17 Flow of Control

Everything ok (there are no switches).

4.18 Files

Everything ok, no files.

Chapter 5

Other problems

The SonarLint auto-inspection detected the following problems:

- Line 45 - *SagePayPaymentServices*

  Add a private constructor to hide the implicit public one

- Line 47 - *module*

  Rename this constant name to match the regular expression $'\hat{}[A-Z][A-Z0-9]*(\_[A-Z0-9]+)*\$'$.

- Line 48 - *resource*

  Rename this constant name to match the regular expression $'\hat{}[A-Z][A-Z0-9]*(\_[A-Z0-9]+)*\$'$.

- Line 48 - *static*

  Reorder the modifiers to comply the Java Language Specification

- Line 50 - *buildCustomerBillingInfo*

  Refactor this method to reduce its Cognitive Complexity from 34 to the 15 allowed.

- Line 54 - *<String, String>*

Replace the type specification in this constructor call with the diamond operator ("<>").

- Line 69 - *orderPaymentPreference*

  Define a costant instead of duplicating this literal 6 times.

- Line 74(67-74) - *try-if*

  Refactor this code do not nest more than 3 if/for/while/switch/try statements.

- Line 81 - *billingAddress*

  Define a costant instead of duplicating this literal 4 times.

- Line 84 - *if*

  Refactor this code do not nest more than 3 if/for/while/switch/try statements.

- Line 90 - *cardNumber*

  Define a costant instead of duplicating this literal 4 times.

- Line 94 - *if*

  Refactor this code do not nest more than 3 if/for/while/switch/try statements.

- Line 98 - *cardType*

  Define a costant instead of duplicating this literal 4 times.

- Line 99 - *if*

  Refactor this code do not nest more than 3 if/for/while/switch/try statements.

- Line 119 - *orderId*

  Define a costant instead of duplicating this literal 5 times.

- Line 121 - *currency*

  Define a costant instead of duplicating this literal 5 times.

- Line 127 - *GenericEntityException ex*

  Either log or rethrown this expection.

- Line 133 - *amount*

  Define a costant instead of duplicating this literal 6 times.

- Line 135 - *description*

  Define a costant instead of duplicating this literal 4 times.

- Line 137 - *cardHolder*

  Define a costant instead of duplicating this literal 3 times.

- Line 138 - *expiryDate*

  Define a costant instead of duplicating this literal 3 times.

- Line 141 - *billingPostCode*

  Define a costant instead of duplicating this literal 3 times.

- Line 153 - *Map<String, Object> response = null;*

  Remove this useless assignment to local variable *"response"*.

- Line 155 - *locale*

  Define a costant instead of duplicating this literal 8 times.

- Line 174 - *paymentGatewayConfigId*

  Define a costant instead of duplicating this literal 10 times.

- Line 181 - *vendorTxCode*

  Define a costant instead of duplicating this literal 6 times.

- Line 198 - *status*

  Define a costant instead of duplicating this literal 5 times.

- Line 199 - *statusDetail*

  Define a costant instead of duplicating this literal 5 times.

- Line 200 - *vpsTxId*

  Define a costant instead of duplicating this literal 5 times.

- Line 201 - *securityKey*

  Define a costant instead of duplicating this literal 4 times.

- Line 202 - *txAuthNo*

  Define a costant instead of duplicating this literal 5 times.

- Line 213 - *INVALID*

  Define a costant instead of duplicating this literal 4 times.

- Line 216 - *MALFORMED*

  Define a costant instead of duplicating this literal 4 times.

- Line 226 - *SagePay - Invalid status*

  Define a costant instead of duplicating this literal 4 times.

- Line 226 - *received for order :*

  Define a costant instead of duplicating this literal 4 times.

- Line 227 - *ERROR*

  Define a costant instead of duplicating this literal 4 times.

- Line 231 - *errorString*

  Define a costant instead of duplicating this literal 5 times.

- Line 241 - *authTransaction*

  Define a costant instead of duplicating this literal 4 times.

- Line 257 - *altReference*

  Define a costant instead of duplicating this literal 6 times.

- Line 258 - *referenceNum*

  Define a costant instead of duplicating this literal 4 times.

- Line 259 - *gatewayFlag*

  Define a costant instead of duplicating this literal 4 times.

- Line 260 - *gatewayCode*

  Define a costant instead of duplicating this literal 4 times.

- Line 309 - *captureTransaction*

  Define a costant instead of duplicating this literal 3 times.

- Line 330 - *Map<String, Object> response = null;*

  Remove this useless assignment to local variable *"response"*.

- Line (100 - 111) - (*if - else if - else*)

  We could use a switch statement.

- Line (206 - 238) - (*if - else if - else*)

  We could use a switch statement.

- Line (427 - 439) - (*if - else if - else*)

  We could use a switch statement.

Chapter 6

Conclusion

This SagePayPaymentService.java has been developed by a team of professionals and qualified programmers. Still, in this document it is possible to see that this code is not perfect. Using a checklist is an efficient, systematic, and rigorous way to point out some remaining bugs to make the code as coherent as possible.
Unfortunately, code inspection is a very time consuming activity and it is not envisageable to apply it totally for large projects like Apache OFBiz but fortunately it is possible for large projects to use tools such as Sonar to perform some of the checkings in an automated way. Moreover, even if using a checklist is a good way to improve the code, this is by no means a guarantee for making a perfect code. Indeed, some of the bugs or possible improvements shown in this document are not related to points of the checklist. For example we can see that some of the methods implemented in the java document are never used in the project. This is not a real bug but it is more a conceptual mistake. Time has been wasted making these methods and disk space is wasted on every machine hosting the project.

Chapter 7

Tools Used

- **Github:** to publicize our work
- **Texpad:** to design CI with LaTeX
- **IntelliJ Idea:** to test the java class

Chapter 8

Effort Spent

In this section you will include information about the number of hours each group member has worked towards the fulfillment of this deadline.

Date	Marco (h)	Angelo (h)	Gabriele (h)
03/01/2017		1.5	
08/01/2017		3.5	
17/01/2017	2		2
21/01/2017		1	
24/01/2017		1	
30/01/2017	1		
31/01/2017	1		
03/02/2017	3		4
04/02/2017	3		4
05/02/2017		3	
TOTAL	10	10	10

Chapter 9

Changelog

Version 1.0 - First Release Version