Software Engineering 2: "PowerEnJoy"
Code Inspection (V. 1.0)

Authors:
Sergio CAPRARA
Soheil GHANBARI
Erica TINTI

Milan, Italy
05/02/2017

Table of contents

1. Introduction

1.1 Revision History

Version	Date	Authors	Description
1.0	05/02/2017	S. Caprara, S. Ghanbari, E. Tinti	First release

1.2 Purpose and Scope

The objective of this document is to provide details about the inspection of the lines of code of some specific class contained in the Apache OFBiz (Open For Business) release.

The code inspection aims at finding possible issues inside the code and tries to provide solutions. All the aspects of the code are reviewed, such as the name of the methods, attributes and their definition, their usage, the presence of comments and descriptions, the length of LOC and so on. A check-list helps in doing the code analysis.

The Apache OFBiz is an open source product for the automation of enterprise processes that includes framework components and business applications, as stated on the official website.

1.3 Definitions and Abbreviations

- **LOC:** lines of code.
- **DB:** Database.

1.4 Reference Documents

The documents used as a reference to provide the design document are:

- Code Inspection Assignment Task Description.pdf
- OFBIZ Technical Documentation (***https://cwiki.apache.org/confluence/display/OFBIZ/OFBiz+Technical+Documentation+-+Home+Page***)

1.5 Document Structure

The sections of the document are:

- **Introduction:** this chapter contains the purpose and scope of this document and introduces the code inspection.
- **Classes:** here, we describe the classes of this study and their respective methods.
- **Functional role:** this section contains an overview of the role that the classes we are inspecting play in the entire framework.
- **List of Issues:** this chapter contains all the issues that have been found in the inspected classes and provides the line number to make it easier to find them.
- **Other problems:** in this section, we describe other issues not included in the check-list and possible solutions.

2. Classes

The target classes of this code inspection are BillingAccountWorker and PaymentWorker. The methods of each class are listed below.

2.1 BillingAccountWorker

Namespace: org.apache.ofbiz.accounting.payment

Methods:

- makePartyBillingAccountList(GenericValue, String, String, Delegator, LocalDispatcher)
- getBillingAccountOpenOrders(Delegator, String)
- getBillingAccountAvailableBalance(GenericValue)
- getBillingAccountAvailableBalance(Delegator, String)
- getBillingAccountNetBalance(Delegator, String)
- availableToCapture(GenericValue)
- calcBillingAccountBalance(DispatchContext, Map<String, ? extends Object>)

2.2 PaymentWorker

Namespace: org.apache.ofbiz.accounting.payment

Methods:

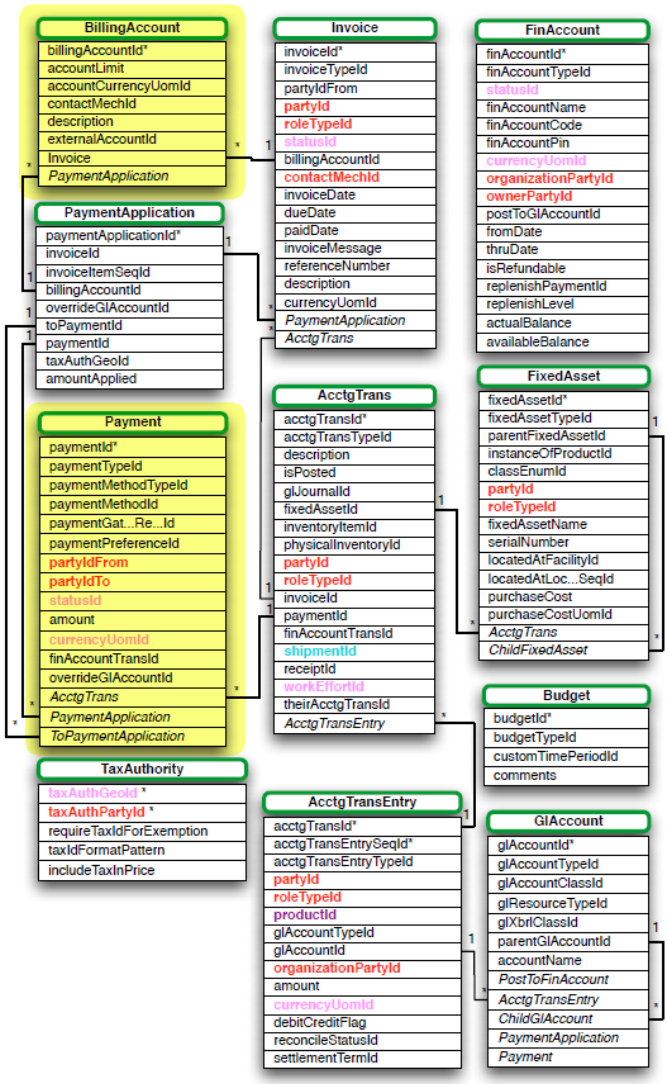
- getPartyPaymentMethodValueMaps(Delegator, String)
- getPartyPaymentMethodValueMaps(Delegator, String, Boolean)
- getPaymentMethodAndRelated(ServletRequest, String)
- getPaymentAddress(Delegator, String)
- getPaymentsTotal(List<GenericValue>)
- getPaymentApplied(Delegator, String)
- getPaymentApplied(Delegator, String, Boolean)
- getPaymentAppliedAmount(Delegator, String)
- getPaymentApplied(GenericValue)
- getPaymentApplied(GenericValue, Boolean)
- getPaymentNotApplied(GenericValue)

- getPaymentNotApplied(GenericValue, Boolean)
- getPaymentNotApplied(Delegator, String)
- getPaymentNotApplied(Delegator, String, Boolean)

3. Functional role

The two classes ***PaymentWorker*** and ***BillingAccountWorker*** are part of the Accounting section of the Apache OFBiz project. The JavaDoc details are missing in their description, as the only annotations are respectively “Worker methods for Payments” and “Worker methods for BillingAccounts”.

The diagram shown below is part of the online documentation concerning the Data Model. It may clarify the main structure of the Accounting package.



The two classes discussed in this document refer to the highlighted elements and are also used to build queries to control the presence of payments or to retrieve the status of a billing account (balance and open orders) on the DB.

As you can remark by reading the documentation, the billing account is used to provide a more structured organisation of Invoices and Payments, but its role is not essential.
This component is also a way to allow the customers to consolidate several invoices into an account that is paid off later.

The class PaymentWorker provides some utility methods for two forms of payment services: one that does credit card authorization internally (through a gateway) and one that does the payment process externally.

4. List of Issues

4.1 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.***

   BillingAccountWorker

   Line 67: String currencyUomId may be not clear enough.

   Line 85: billingAccountVO variable has a name not clear.

   Line 88: thruDate not clear the meaning of "thru".

   Line 156: pAi stands for paymentApplicationIterator but not clear.

   Line 180: availableToCapture() method name could be more meaningful.

   Line 230: variable delegator may be not clear enough.

   Line 262: actualCurrencyUomId variable has a name not clear.

2. ***If one-character variables are used, they are used only for temporary throwaway variables, such as those used in for loops.***

   Everything ok.

3. ***Class names are nouns, in mixed case, with the first letter of each word in capitalized.***

   Everything ok.

4. ***Interface names should be capitalized like classes.***

   Everything ok.

5. ***Method names should be verbs, with the first letter of each addition word capitalized.***

   BillingAccountWorker

   Line 180: method availableToCapture() has not a verb as a name.

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.***

Everything ok.

7. ***Constants are declared using all uppercase with words separated by an underscore.***

   BillingAccountWorker

   Lines 54-55: declared constants are not all uppercase.

   Line 56: variable ZERO is uppercase but it is not a constant.

   PaymentWorker

   Lines 50-52: declared constants are lowercase, instead of being uppercase.

4.2 Indention

8. ***Three or four spaces are used for indentation and done so consistently.***

   Everything ok.

9. ***No tabs are used to indent.***

   Everything ok.

4.3 Braces

10. ***Consistent bracing style is used.***

    Everything ok.

11. ***All if, while, do-while, try-catch, and for statements that have only one statement to execute are surrounded by curly braces.***

    BillingAccountWorker

    Line 64: the if condition has only one statement without braces.

    PaymentWorker

    Line 66: the if condition has only one statement without braces.

    Line 103: if statement is inline.

    Line 107: if statement has no braces.

4.4 File Organization

12. Blank lines and optional comments are used to separate sections.

Code is poorly commented.

PaymentWorker

Adding some blank lines to method getPaymentApplied() would help readability.

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

A possible solution is breaking the lines after commas or when a method call takes place.

BillingAccountWorker

Lines 67-68, 70-71, 73, 75, 77-79, 85, 88-89, 91-92, 94-95, 98, 103, 111, 113-116, 121-122, 127-130, 133, 138-139, 144-145, 151, 154-156, 158, 161-162, 175, 180-181, 187, 194, 196, 198, 202-205, 210, 212, 216-218: limit of 80 characters is exceeded.

PaymentWorker

Lines 51-52, 57, 61-62, 64, 66, 69, 73-90, 99, 128-132, 159, 161, 163, 165, 176, 180-181, 184, 191, 193, 214, 225, 229, 231, 236, 242, 252, 256, 259-263, 274, 283, 285, 288, 293-294, 296-297, 300-305, 308, 319, 324-326, 328, 331, 335, 337, 342, 348, 350: limit of 80 characters is exceeded.

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

A possible solution is breaking the lines after commas or when a method call takes place.

BillingAccountWorker

Lines 67, 70, 71, 73, 111, 121-122, 127, 130, 138-139, 144, 151, 154-155, 162, 175, 181, 187, 194, 218: limit of 120 characters is exceeded.

PaymentWorker

Lines 61, 64, 99, 128-132, 159, 161, 163, 165, 181, 191, 193, 214, 225, 229, 231, 236, 242, 252, 256, 259-263, 288, 293, 294, 296, 297, 300- 305, 324-326, 328, 331, 335, 337, 342, 348, 350: limit of 120 characters is exceeded.

4.5 Wrapping Lines

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

    No issues found.

16. ***Higher-level breaks are used.***

    No issues found.

17. ***A new statement is aligned with the beginning of the expression at the same level as the previous line.***

    No issues found.

4.6 Comments

18. ***Comments are used to adequately explain what the class, interface, methods, and blocks of code are doing.***

    No issues found.

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.***

    No issues found.

4.7 Java Source Files

20. ***Each Java source file contains a single public class or interface.***

    No issues found.

21. ***The public class is the first class or interface in the file.***

    No issues found.

22. ***Check that the external program interfaces are implemented consistently with what is described in the javadoc.***

    No issues found.

23. Check that the javadoc is complete (i.e., it covers all classes and files part of the set of classes assigned to you).

BillingAccountWorker

Line 67: method makePartyBillingAccountList() Javadoc is missing

Line 138: method getBillingAccountAvailableBalance() Javadoc is missing

Line 187: method calcBillingAccountBalance() Javadoc is missing

PaymentWorker

Line 61: method getPartyPaymentMethodValueMaps() Javadoc is missing

Line 99: method getPaymentMethodAndRelated() Javadoc is missing

Line 176: method getPaymentAddress() Javadoc is missing

Line 229: method getPaymentApplied() Javadoc is missing

Line 317: method getPaymentNotApplied() Javadoc is missing

Line 324: method getPaymentNotApplied() Javadoc is missing

Line 331: method getPaymentNotApplied() Javadoc is missing

Line 335: method getPaymentNotApplied () Javadoc is missing

4.8 Package and Import Statements

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

No issues found.

4.9 Class and Interface Declaration

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. last private class variables.

(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.

BillingAccountWorker

Line 59: package variables are declared after private class variables (lines 57-58)

26. Methods are grouped by functionality rather than by scope or accessibility.

PaymentWorker

Method getPaymentAppliedAmount() should be moved at the end or before the first getPaymentNotApplied()

27. Check that the code is free of duplicates, long methods, big classes, breaking encapsulation, as well as if coupling and cohesion are adequate.

No issues found.

4.10 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).

BillingAccountWorker

Line 127: method visibility could be private

Line 151: method visibility could be private

Line 180: method visibility could be private

Lines 106, 111, 120, 121, 122, 123, 124, 156 should be moved on the top of the method.

29. Check that variables are declared in the proper scope.

No issues found.

30. ***Check that constructors are called when a new object is desired.***

    No issues found.

31. ***Check that all object references are initialized before use.***

    No issues found.

32. ***Variables are initialized where they are declared, unless dependent upon a computation.***

    No issues found.

33. ***Declarations appear at the beginning of blocks (A block is any code surrounded by curly braces `f' and `g'). The exception is a variable can be declared in a for loop.***

    No issues found.

4.11 Method Calls

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

    No issues found.

35. ***Check that the correct method is being called, or should it be a different method with a similar name.***

    No issues found.

36. ***Check that method returned values are used properly.***

    No issues found.

4.12 Arrays

37. ***Check that there are no one-by-one errors in array indexing (that is, all required array elements are correctly accessed through the index).***

    No issues found.

38. ***Check that all array (or other collection) indexes have been prevented from going out-of-bounds.***

No issues found.

39. Check that constructors are called when a new array item is desired.

No issues found.

4.13 Object Comparison

40. Check that all objects (including Strings) are compared with equals and not with ==.

No issues found.

4.14 Output Format

41. Check that displayed output is free of spelling and grammatical errors.

No issues found.

42. Check that error messages are comprehensive and provide guidance as to how to correct the problem.

PaymentWorker

Lines 238, 268: the message "Problem getting Payment" is not clear enough for the user.

Line 337: the message "Null delegator is not allowed in this method" is not very clear.

43. Check that the output is formatted correctly in terms of line stepping and spacing.

No issues found.

4.15 Computation, Comparisons, and Assignments

44. Check that the implementation avoids brutish programming: see http://users.csc.calpoly.edu/~jdalbey/SWE/CodeSmells/bonehead.html

PaymentWorker

Lines 143-152 and 158-165 performs the same checks. Group them.

45. ***Check order of computation/evaluation, operator precedence and parenthesizing.***

    No issues found.

46. ***Check the liberal use of parenthesis is used to avoid operator precedence problems.***

    No issues found.

47. ***Check that all denominators of a division are prevented from being zero.***

    PaymentWorker

    Line 263: possible division by zero (actualCurrencyAmount), could be solved by checking the value of the element.

48. ***Check that integer arithmetic, especially division, are used appropriately to avoid causing unexpected truncation/rounding.***

    No issues found.

49. ***Check that the comparison and Boolean operators are correct.***

    No issues found.

50. ***Check throw-catch expressions, and check that the error condition is actually legitimate.***

    No issues found.

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

    No issues found.

4.16 Exceptions

52. ***Check that the relevant exceptions are caught.***

    No issues found.

53. ***Check that the appropriate action are taken for each catch block.***

    No issues found.

4.17 Flow of Control

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

No issues found.

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

No issues found.

56. Check that all loops are correctly formed, with the appropriate initialization, increment and termination expressions.

No issues found.

4.18 Files

57. Check that all files are properly declared and opened.

No issues found.

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

No issues found.

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

No issues found.

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

No issues found.

5. Other problems

Methods of the class ***BillingAccountWorker*** are only called internally (or never called) and no methods are called outside the class. According to this, we can tell that this class is currently not useful and may be removed. Moreover, the methods listed below are never used:

- makePartyBillingAccountList(GenericValue, String, String, Delegator, LocalDispatcher), line 67
- getBillingAccountOpenOrders(Delegator, String), line 111
- getBillingAccountAvailableBalance(Delegator, String), line 138

The following methods of the class ***PaymentWorker*** are never used and may be removed:

- getPartyPaymentMethodValueMaps(Delegator, String), line 57
- getPaymentMethodAndRelated(ServletRequest, String), line 99
- getPaymentApplied(Delegator, String), line 225
- getPaymentAppliedAmount(Delegator, String), line 252
- getPaymentNotApplied(GenericValue, Boolean), line 324
- getPaymentNotApplied(Delegator, String), line 331

6. Hours of work

To make this document we have spent:

- Sergio Caprara, 15 hours
- Soheil Ghanbari, 5 hours
- Erica Tinti, 15 hours