Politecnico di Milano

A.A 2016-2017

Code Inspection Document

Version 1.0

PowerEnjoy

Instructor : Prof. Di Nitto

Authors:
Amico Simone
Chianella Claudia Beatrice
Giovanakis Yannick

CONTENTS

1. INTRODUCTION

The code inspection is a systematic examination of computer source. It is intended to find mistakes overlooked during the initial development phase, with the aim of developing the overall quality of the software.
This document indexes all coding mistakes found following a standard code inspection check-list for the requested classes listed in the underling section 1.1

1.1 Class overview

The Apache OFBiz Project is an open source product for the automation of enterprise processes that includes framework components and business applications for ERP,CRM and other business-oriented functionalities.[1]
Among the many, the following classes were assigned for code inspection:

- **ProductStoreCartAwareEvents.java**
  - *Location*: apache-ofbiz-16.11.01/applications/order/src/main /java/org/apache/ofbiz/order/shoppingcart/product /ProductStoreCartAwareEvents.java
  - *Class role*: see section 2.1
  - *Total number of issues found*: TBD
- **ProductDisplayWorker.java**
  - *Location*: apache-ofbiz-16.11.01/applications/order/src/main /java/org/apache/ofbiz/order/shoppingcart/product /ProductDisplayWorker.java
  - *Class role*: see section 2.2
  - *Total number of issues found*: TBD

[1]Full source code: `http://mirror.nohup.it/apache/ofbiz/apache-ofbiz-16.11.01.zip`

2. FUNCTIONAL ROLE OF CLASSES

This section analyses the meaning and functionality for each of the inspected classes.

2.1 ProductStoreCartAwareEvents.java

This java class is composed of two static methods. The first static method takes a `HttpServletRequest` and `HttpServletResponse` as input. The method extracts the `productStoreId` String parameter from the request and tries to invoke the second static method. Depending on the success or failure of the second method , the the first method return either the string ”success” or ”error”
The second static method takes extracted `productStoreID` String and the `HttpServletRequest` as input. The method than :

- returns immediately if the `productStoreID` is `null` or if it equals the `HttpSession`'s old `productStoreID`.
- throws an exception if `productStoreID` is not related to a valid productStore , if the `webSite` cannot be found in relation to the request or if `webSite` exists but does not allow changing of the `productStoreID`.
- if the points above are not matched , sets the `productStoreID` of the `session` to the one passed as parameter ,clears the old product catalogue and eventually sets the session's `currency`, `locale` and `time-zone` depending on the `productStore`. Finally the method creates a new `shoppingCart` for the session without erasing the old one.

2.2 ProductDisplayWorker.java

This class is composed of three static methods and a private inner class. The first method , `getRandomCartProductAssoc` , receives a **ServletRequest** and a boolean as input. The methods than extracts the **ShoppingCart** object from the request and for each item in the cart looks for products that are associated with that product.The associated products found are memorized in a `cartAssocs` object .From this object all items but three are eliminated randomly. The `cartAssocs` is then returned.
The `getQuickReorderProducts` generates `Map<String, Object>` object containing ordered products.
The method calls the third static method ,`prodcutOrderByMao` whose function is to order the products according to the `Map` passed as parameter. The sorting process uses the static `sort` method of the Java `Collections` .
The private inner class `ProductByMapComparator` implements `Comparator<Object>` and is used by the `Collections.sort` to sort the list.

3. List of Issues

3.1 ProductStoreCartAwareEvents.java

3.1.1 Naming conventions

- Line 99: The method setCurrencyUomIfNone doesn't have a meaningful name.
- Line 50: The only one-character variable (Exception e) is a temporary throwaway variable.
- All class names are nouns, in mixed case, with the first letter of each word in capitalized.
- There are no interfaces.
- All method names are verbs, with the first letter of each addition word capitalized.
- All attributes, are mixed case, no one begin with an underscore. All the remaining words in the variable name have their first letter capitalized.
- There are no constants.

3.1.2 Indention

- Four spaces are used for indentation and it is done consistently.
- No tabs are used to indent.

3.1.3 Braces

- In all the class is used the Kernighan and Ritchie style (first brace is on the same line of the instruction that opens the new block).
- All if, while, do-while, try-catch, and for statements that have only one statement to execute are surrounded by curly braces.

3.1.4 File Organisation

- Either blank lines or comments are used to separate sections.
- Line length exceeds 80 characters many times:
  - Line 42
  - Line 44
  - Line 49
  - Line 60
  - Line 66
  - Line 76
  - Line 78
  - Line 84
  - Line 86
  - Line 88
  - Line 91
  - Line 94
  - Line 97
  - Line 98
  - Line 99
  - Line 100
  - Line 101
  - Line 103
  - Line 104
  - Line 105
  - Line 108
  - Line 110
- Seven times it exceeds also 120 characters:
  - Line 78

  - Line 84
  - Line 88
  - Line 94
  - Line 97
  - Line 98
  - Line 110

3.1.5 Wrapping Lines

- Line break occurs only after a comma or an operator.
- Higher-level breaks are not used.
- A new statement is aligned with the beginning of the expression at the same level as the previous line.

3.1.6 Comments

- Comments are used to adequately explain what the class, interface, methods, and blocks of code are doing.
- No issues found about commented out code.

3.1.7 Java Source Files

- The file contains a single public class
- The public class is the first class in the file
- javadoc is not used
- javadoc is not used

3.1.8 Package and Import Statements

- Package is the first non comment line in the file
- All import statements follow the package

3.1.9 Class and Interface Declarations

- Class or interface declarations order:

  1. There is no class documentation comment
  2. Class statement
  3. There is class implementation comment
  4. class (static) variables:
     - First we have class variable
     - There are no protected variables
     - No package level
     - There are no private class variables
  5. istance variables:
     - public instance variable is the first
     - There are no protected instance variable
     - No package level
     - There are no private instance variable
  6. There is no constructor
  7. Methods declaration

- Methods are grouped by functionality then by scope or accessability
- The code is free of duplicates, the method "public static void setSessionProductStore(String productStoreId, HttpServletRequest request) is too long method, the class is not big.

3.1.10 Initialisation and Declaration

- Variables and class members are of the correct type and they have the right visibility
- Variables are declared in the proper scope
- Constructor is not used
- Object references are initialized before use

- Variables are initialized when they are declared
- Declarations:
  - Line 65
  - Line 66
  - Line 73
  - Line 76
  - Line 82
  - Line 86
  - Line 107

3.1.11 Method Calls

- Parameters are presented in the correct order
- The class contains two method with the same name
- Returned values are used properly

3.1.12 Arrays

Arrays are not used

3.1.13 Object Comparison

- Line 61 : '==' used instead of `equals` on a `String`.
- Line 77 : '==' used instead of `equals` on a `GenericValue`. object.
- Line 83 : '==' used instead of `equals` on a `GenericValue`.

3.1.14 Output Format

- No output issues found.

3.1.15 Computation , Comparisons and Assignments

- Line 48 : try and catch condition is legitimate and handled correctly.
- Line 61 : operators handled correctly.
- Line 68 : operators handled correctly.
- Line 77 : operators handled correctly.
- Line 61 : operators handled correctly.
- Line 83 : operators handled correctly.
- Line 87 : operators handled correctly.
- Line 109 : operators handled correctly.

3.1.16 Exceptions

- Line 48 : Try and catch block correctly handled.

3.1.17 Flow Control

- No issues found

3.1.18 Files

- No issues found

3.2 ProductDisplayWorker.java

3.2.1 Naming conventions

- Line 268: nqdbl has no meaningful name
- Line 208: nqint has no meaningful name
- Line 84: curpcms has no meaningful name
- Line 86: curpcm has no meaningful name
- Line 137: the only one-character variable (Exception e) is a temporary throwaway variable.
- All class names are nouns, in mixed case, with the first letter of each word in capitalized.
- There are no interfaces.
- All method names are verbs, with the first letter of each addition word capitalized.
- All attributes, are mixed case, no one begin with an underscore. All the remaining words in the variable name have their first letter capitalized, except one (cartiter, which should have been cartIter).
- There are no constants.

3.2.2 Indention

- Four spaces are used for indentation and it is done consistently.
- No tabs are used to indent.

3.2.3 Braces

- In all the class is used the Kernighan and Ritchie style (first brace is on the same line of the instruction that opens the new block).
- All while, do-while, try-catch, and for statements that have only one statement to execute are surrounded by curly braces. There are 10 if statements which contain onlu one statement and are written in one line without braces.

3.2.4 File Organisation

- Either blank lines or comments are used to separate sections.
- Line length exceeds 80 characters many times:
  - Line 61
  - Line 64
  - Line 70
  - Line 76
  - Line 79
  - Line 81
  - Line 82
  - Line 83
  - Line 84
  - Line 87
  - Line 88
  - Line 89
  - Line 99
  - Line 100
  - Line 101
  - Line 116
  - Line 117
  - Line 121
  - Line 148
  - Line 151
  - Line 154
  - Line 158
  - Line 159
  - Line 160
  - Line 162

- Line 165
- Line 168
- Line 169
- Line 170
- Line 175
- Line 176
- Line 178
- Line 183
- Line 185
- Line 187
- Line 190
- Line 192
- Line 193
- Line 194
- Line 198
- Line 203
- Line 204
- Line 208
- Line 210
- Line 214
- Line 215
- Line 216
- Line 220
- Line 221
- Line 225
- Line 237
- Line 239
- Line 241
- Line 256

  - Line 260
  - Line 261
  - Line 262
  - Line 265
  - Line 268
  - Line 288
  - Line 295
  - Line 303
  - Line 332
- 18 times it exceeds also 120 characters:
  - Line 76
  - Line 77
  - Line 79
  - Line 84
  - Line 117
  - Line 121
  - Line 158
  - Line 159
  - Line 160
  - Line 169
  - Line 198
  - Line 214
  - Line 215
  - Line 216
  - Line 237
  - Line 260
  - Line 268
  - Line 288

3.2.5 Wrapping Lines

- Line break occurs only after a comma or an operator.
- Higher-level breaks are not used.
- A new statement is aligned with the beginning of the expression at the same level as the previous line.

3.2.6 Comments

- Comments are used to adequately explain what the class, interface, methods, and blocks of code are doing.
- No issues found about commented out code.

3.2.7 Java Source Files

- The file contains a single public class
- The public class is the first class in the file
- javadoc is not used
- javadoc is not used

3.2.8 Package and Import Statements

- Package is the first non comment line in the file
- All import statements follow the package

3.2.9 Class and Interface Declarations

- Class or interface declarations order:
    1. There is no class documentation comment
    2. Class statement
    3. There is no class implementation comment
    4. Class (static) variables:
        - First we have class variable

- There are no protected variables
- No package level
- There are no private class variable

5. istance variables:
   - public instance variable is the first
   - There are no protected instance variable
   - No package level
   - The class ”ProductByMapComparator” contains private instance variables
6. There is the constructor
7. Methods declaration

- Methods are grouped by functionality then by scope or accessability
- The code is free of duplicates, there are long methods, the class is big.

3.2.10 Initialisation and Declaration

- Variables and class members are of the correct type and they have the right visibility
- Variables are declared in the proper scope
- Constructor is not used
- Object references are initialized before use
- Variables are initialized when they are declared
- Declarations:
  - Line 169
  - Line 170
  - Line 187
  - Line 190
  - Line 195
  - Line 225

- Line 236
- Line 237
- Line 249
- Line 253
- Line 254
- Line 255
- Line 292

3.2.11 Method Calls

- Parameters are presented in the correct order
- Methods have different names
- Returned values are used properly

3.2.12 Arrays

Arrays are not used

3.2.13 Object Comparison

- Line 66 : '==' used instead of `equals` on a `ShoppingCart` object.
- Line 74 : '==' used instead of `equals` on a `ShoppingCart` object.
- Line 80 : '==' used instead of `equals` on a `List`.
- Line 85 : '==' used instead of `equals` on a `List`.
- Line 109 : '==' used instead of `equals` on a `Iterator`.
- Line 118: '==' used instead of `equals` on a `String`.
- Line 127 : '==' used instead of `equals` on a `List`.
- Line 61 : '==' used instead of `equals` on a `String`.
- Line 154 : '==' used instead of `equals` on a `GenericValue`.

- Line 155 : '==' used instead of `equals` on a `GenericValue`.
- Line 162 : '==' used instead of `equals` on `Maps`.
- Line 172 : '==' used instead of `equals` on a `Iterator`.
- Line 178 : '==' used instead of `equals` on a `Iterator`.
- Line 189 : '==' used instead of `equals` on a `BigDecimal`.
- Line 192 : '==' used instead of `equals` on a `BigDecimal`.
- Line 189 : '==' used instead of `equals` on a `BigDecimal`.
- Line 197 : '==' used instead of `equals` on a `Integer`.
- Line 288 : '==' used instead of `equals` on a `List`.
- Line 289 : '==' used instead of `equals` on a `Integer`.
- Line 322 : '==' used instead of `equals` on `Objects`.
- Line 328 : '==' used instead of `equals` on a `Object`.

3.2.14 Output Format

- No issues found.

3.2.15 Computation , Comparisons and Assignments

- Line 141: return `cartAssocs` would have sufficed.
- Line 208: division by 0 could occur.
- Line 280: wrong Exception handling

3.2.16 Exceptions

- Line 280: wrong Exception handling. Debugger used instead of error solving.

3.2.17 Flow Control

- All for and while loops are well formed.

3.2.18 Files

- No issues found

4. OTHER ISSUES

Section dedicated to other issues found

- The code is generally poorly documented and **ProductDisplayWorker** is hard to understand.
- Javadoc was not used : methods and classes functionality is not always obvious and therefore hard to understand.

5. APPENDICES

5.1 References

The following tools where used in the creation of this document:

- *TexMaker 4.5* as Editor

5.2 Effort Spent

- Simone Amico 6h
- Chianella Claudia Beatrice 6 h
- Giovanakis Yannick 6 h