POLITECNICO DI MILANO
MSc COMPUTER SCIENCE AND ENGINEERING

SOFTWARE ENGINEERING 2
ACADEMIC YEAR 2016-2017

Code Inspection Document

PowerEnJoy

Authors:

Melloni Giulio   876279
Renzi Marco   878269
Testa Filippo   875456

Reference Professor:
MOTTOLA Luca

Release Date: February 5th, 2017
Version 1.0

Table of Contents

1 | Code description

1.1 Assigned class

The assigned class, named ***ShoppingListEvents***, is part of the *Apache OFBiz* project which focuses on enterprises processes and includes specific frameworks for ERP, CRM and other business-oriented functionalities; in particular the interested class is part of the ***org.apache.ofbiz.order.shoppinglist*** *package.*
The Code Inspection activity will be carried on the previously stated class with the final intent to discover any possible error that affects the quality and functional behaviour of piece of the software.

1.2 Functional role

2 | Code issues

This chapter reports all the issues found into the class implementation with respect to the check list provided within the *Code Inspection Assignment Task Description* document. The first section just explain what is the adopted notation for reporting the various issues, while the second one, the core of this document, goes methodically through every single point of the checklist and if a problem of that kind is found, using a table, the line(s) affected and a brief rationale are reported.

2.1 Notation

The notation adopted is very simple: if any issue of a specific type isn't found, just a short comment is provided to motivate the absence of a that problem, otherwise with the help of a table, are reported the line(s) where the issues have been found with a related short explanation.

Moreover, in order to maintain the same structure of the given checklist, each point is analysed in a different subsection.

2.2 Checklist issues

2.2.1 Naming Conventions

Line	*Issue*
602 610 611	Variables without a meaningful names
67 68	Constants are not declared using all upper case with words separated by an underscore

Other variables names, methods names and class name are used properly and have a meaningful name.

Only *throwaway* are sometimes composed by a one-character word.

Class name is written with the first letter in capitalized and method names are verbs with the first letter of each addition word capitalized.

Variables, methods and class are written with the camel notation.

The other constant is written using all upper case with words separated by an underscore.

2.2.2 Indention

Throughout all the class, four spaces are used consistently to properly indent the code instructions; moreover, any *tabs* command are used to indent the lines.

2.2.3   Braces

No significant issues are found with respect to bracing usage. The "Kernighan and Ritchie" style is adopted and it is consistent throughout the entire class. In general, curly braces are used for blocks within *If, while, do-while, try-catch* and *for* clauses with only one statement too with the following exception:

Line	Issue
337	*If* clause with only one statement is devoid of curly braces

2.2.4   File Organization

Blank lines and optional comments are properly used to separate different sections of the code.
The following table reports the lines that exceed the maximum length of 80 characters, along with a rationale.

Line	Issue

Similarly to the previous table, the following one reports the lines that exceed the maximum length of 120 characters, along with a brief rationale.

Line	Issue
85, 99, 107, 110, 126, 151, 161, 206, 214, 216, 236, 283, 294, 299, 304, 342, 344, 369, 402, 479, 503, 656	All these lines are characterized by an object assignment with a method call, which in most cases has a long list of parameters
95, 201, 360, 548	In this case the lines refer to method signatures (composed of qualifiers, name, parameters, exceptions)
143, 148, 188, 289, 291, 297, 371, 414, 524	The lines represent simply method calls and the maximum allowed length is broken because of long list of parameters. In particular, in *line 188* is also present an in-line if which lengthens the numbers furthermore the length
277	This line refers to an if clause which is quite long because are used different method calls
490 491	These are comment lines simply explaining an else branch of an if statement

2.2.5   Wrapping Lines

In general the code is neat, tidy and statements on consecutive lines are properly aligned. Typically line breaks after commas (such as in definition of the input parameters of methods) and operators are respected with this exception:

Line	Issue
110	
126	
143	
161	
206	
216	In method *UtilProperties.getMessage(...)* the second parameter is attached to a comma
236	
242	
294	
297	
299	
304	

2.2.6   Comments

Line	Issue
204	
240	
287	Comments used doesn't explain anything more than what the code says
313	
314	

The remaining part of the code is not commented sufficiently: there could be more lines which explain the function of a code block, especially in the critical parts.
The other lines of code are meaningful and explain what the code is doing correctly.

2.2.7   Java Source File

The Java Source File contains only a single *public class* which is, obviously, the assigned *ShoppingListEvents* one. For this reason, the unique public class is also the first one in the file.
Since there is no complete javadoc of the class (except for a few words that do not add any extra information with respect to the class name) it is impossible to check if the external interfaces are implemented properly because their descriptions in the javadoc is totally missing.
Furthermore, the javadoc of the methods is frequently absent while in the case it is present, it is very short and never covers the input parameters, the returned output (if present) and the exceptions thrown of the related method.

2.2.8   Package and Import Statements

No issues of this type. Package declaration (*package org.apache.ofbiz.order.shoppinglist;*) is the very first non comment statement and *import* statements follow.

2.2.9   Class and Interface Declarations

Line	Issue
393 621 684	Methods aren't grouped by functionality, scope or accessibility to the other close to it
95 201 443	Methods are long. They should be divided into more sub-methods that are maybe useful to reuse in future

Other class, variables and methods positions are respected and are grouped by functionality rather than by scope or accessibility.
Moreover, the code doesn't contain any duplicates.

2.2.10   Initialization and Declarations

No critical issues are found in this field. In particular, variables and class members type are consistent with respect to their declarations.
The scope of variables is aligned with the purpose of the block of code in which the variables are declared: since the class in consideration is essentially a list of methods for almost every variable the scope is limited to the method in which it is declared.
Objects are always initialized before use or if a computation is needed they are set to *null*. Most times declarations of variables occur at the beginning of the blocks in which they will be used, thus making possible to easily look up for a particular variable.
Event though the overall class is well structured with respect to initializations and declarations, one may question about these arguments:

Line	Issue
67 68 69	The three *public static final* variables may be set to *private* since they are unlikely to be used outside this class
81 85 121	String variables *shoppingListId* and *selectedCartItems* are reassigned after initialization in previous lines. A legal operation, but a new object is created.
293 332 365 366	Variables declarations should be put at the beginning of the corresponding blocks and not in these positions

2.2.11   Method Calls

There is no error to underline.
Every method has parameters presented in the correct order.
When a method is called, it is called the right one, although there are similar names.
Return value of the method is used properly in each case.

2.2.12   Arrays

Arrays problems is not an issue in this class since there are no off-by-one errors (i.e. array elements are accessed without indexing problems and loops are executed a right number

of times) and out-of-bounds elements.

2.2.13   Object Comparison

Every object comparison is done with the *Java* method **equals()** and not with the $==$ or the **!=**.
The object is correctly compared to something with the $==$ when it is important to see if the variable is null.

2.2.14   Output Format

In the code there aren't instructions that directly display text and messages to the users (such as, for example, *System.out.println(..)*); however the code in some cases deals with variables of type *string*: for each of them the strings correctly spelled and free of grammatical errors. Also, they are always correctly formatted, in terms of both line stepping and spacing.
More specifically, error messages (some of them write into the log others instead are simply returned strings) are syntactically correct, however in some cases they are not totally clear and does not provide any specific guidance on how to correct the problem. This situation can be found at the following lines:

Line	Issue
109	These instructions write into the log just a generic error message, which seems
160	to be poorly detailed in order to provide a precise way on how to correct the
235	problem

2.2.15   Computation, Comparisons and Assignments

The class implementations is devoid of any kind of *brutish programming*.

2.2.16   Exceptions

Line	Issue
142	Exception used isn't meaningful
328	

Other exceptions are used correctly and have a correct meaning that helps to find the problem of different program routine.

2.2.17   Flow of Control

The class has no particular complex control flow structures but is indeed quite redundant with *if* statements that weigh down the reading of the code. Typically these *if* clauses are not followed by *else* counterparts and so the default branches in these cases are trivially the instructions that follow the *if* block.
For what concerns loops most of the times the preferred pattern is the *for-each* loop, consequently initialization,increment and termination are not issues. The standard *for* loops are well formed too.

2.2.18   Files

The *ShoppingListEvents* class doesn't use any external file, for this reason there aren't instructions involving file management.

2.3   Other issues

This section underline problems that are not specified in the *Checklist issues* because of their importance.

Line	Issue
285	TODO must be commented differently with the notation $\backslash\backslash TODO$ so that
367	every IDE can find it easily and recognize it
360	It is better to have near the same type of parameters. In this example, *String* ones are divided
694	Eliminate wrapping lines that aren't used to visualize the code better

3 | Effort Spent

In order to complete this document, each author worked for XX hours.

4 | Revision History

Version	Date	Summary
1.0	05/02/2017	First release of the CI document