



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
1.1	Assigned class	1
1.2	Functional role	1
2	Code issues	2
2.1	Notation	2
2.2	Checklist issues	2
2.2.1	Naming Conventions	2
2.2.2	Indention	2
2.2.3	Braces	2
2.2.4	File Organization	2
2.2.5	Wrapping Lines	3
2.2.6	Comments	3
2.2.7	Java Source File	3
2.2.8	Package and Import Statements	3
2.2.9	Class and Interface Declarations	4
2.2.10	Initialization and Declarations	4
2.2.11	Method Calls	4
2.2.12	Arrays	5
2.2.13	Object Comparison	5
2.2.14	Output Format	5
2.2.15	Computation, Comparisons and Assignments	5
2.2.16	Exceptions	5
2.2.17	Flow of Control	5
2.2.18	Files	5
2.3	Other issues	5
3	Effort Spent	7
4	Revision History	8

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

2.1 Notation

2.2 Checklist issues

2.2.1 Naming Conventions

<i>Line</i>	<i>Issue</i>
602 610 611	Variables without a meaningful names
67 68	Constants are not declared using all uppercase 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 uppercase with words separated by an underscore.

2.2.2 Indention

<i>Line</i>	<i>Issue</i>
-------------	--------------

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:

<i>Line</i>	<i>Issue</i>
337	<i>If</i> clause with only one statement is devoid of curly braces

2.2.4 File Organization

<i>Line</i>	<i>Issue</i>
-------------	--------------

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:

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

2.2.6 Comments

<i>Line</i>	<i>Issue</i>
204	Comments used doesn't explain anything more than what the code say
240	
287	
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

<i>Line</i>	<i>Issue</i>
-------------	--------------

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

<i>Line</i>	<i>Issue</i>
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 initilizations and declarations, one may question about these arguments:

<i>Line</i>	<i>Issue</i>
67 68 69	The three <i>public static final</i> variables may be set to <i>private</i> since they are unlikely to be used outside this class
81 85 121	String variables <i>shoppingListId</i> and <i>selectedCartItems</i> 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

<i>Line</i>	<i>Issue</i>
-------------	--------------

2.2.15 Computation, Comparisons and Assignments

<i>Line</i>	<i>Issue</i>
-------------	--------------

2.2.16 Exceptions

<i>Line</i>	<i>Issue</i>
142 328	Exception used isn't meaningful

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

<i>Line</i>	<i>Issue</i>
-------------	--------------

2.3 Other issues

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

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

3 | Effort Spent

4 | Revision History