***PowerEnJoy***



**Code Inspection Document**

Authors:

Emanuele Chilà

Giorgio Lazzarinetti

January 22th 2017

SUMMARY

[1. INTRODUCTION 3](#_Toc472888321)

[1.1. PURPOSE AND SCOPE 3](#_Toc472888322)

[1.2. LIST OF REFERENCE DOCUMENT 3](#_Toc472888323)

[2. CLASS ASSIGNED 4](#_Toc472888324)

[3. FUNCTIONAL ROLE OF THE ASSIGNED CLASS 4](#_Toc472888325)

[4. CODE INSPECTION 4](#_Toc472888326)

[4.1. NAMING CONVENTIONS 4](#_Toc472888327)

[4.2. INDENTION 5](#_Toc472888328)

[4.3. BRACES 5](#_Toc472888329)

[4.4. FILE ORGANIZATION 5](#_Toc472888330)

[4.5. WRAPPING LINES 6](#_Toc472888331)

[4.6. COMMENTS 6](#_Toc472888332)

[4.7. JAVA SOURCE FILE 7](#_Toc472888333)

[4.8. PACKAGE AND IMPORT STATEMENTS 7](#_Toc472888334)

[4.10. INITIALIZATION AND DECLARATION 9](#_Toc472888335)

[4.11. METHOD CALLS 9](#_Toc472888336)

[4.12. ARRAYS 10](#_Toc472888337)

[4.13. OBJECT COMPARISON 10](#_Toc472888338)

[4.14. OUTPUT FORMAT 10](#_Toc472888339)

[4.15. COMPUTATION, COMPARISONS AND ASSIGNMENTS 11](#_Toc472888340)

[4.16. EXCEPTIONS 11](#_Toc472888341)

[4.17. CONTROL FLOW 11](#_Toc472888342)

[4.18. FILES 12](#_Toc472888343)

[5. EFFORT SPENT 12](#_Toc472888344)

## **INTRODUCTION**

### **PURPOSE AND SCOPE**

In this document, we are going to analyze a piece of code assigned and looking for errors on it by applying the inspection approach. The inspection approach consists in checking for certain properties in the code listed in a set of points, it is a systematic approach that allows to always applying the same criterion in the analysis and debugging of the code, and always in a certain way to detach the psychological friction between analysts and programmers who developed the code.

### **LIST OF REFERENCE DOCUMENT**

Specification Document: Assignment 4 CodeInspection.pdf

## **CLASS ASSIGNED**

We have been assigned to the code inspection of the class:

**apache-ofbiz-16.11.01\framework\widget\src\main\java\org\apache\ofbiz\widget\model\HtmlWidjet.java**

In this source code, we identified

The main public class:

* HtmlWidget;

Some nested public static classes:

* ExtendedWrapper
* StringHtmlWrapperForFtl
* CollectionHtmlWrapperForFtl
* HtmlTemplate
* HtmlTemplateDecorator
* HtmlTemplateDecorator

This is a class coming from the 16.11.01 version of Apache OFBiz Project.

## **FUNCTIONAL ROLE OF THE ASSIGNED CLASS**

The HtmlWidget seems to be a widget for an html page with the purpose to show graphic components on the screen.

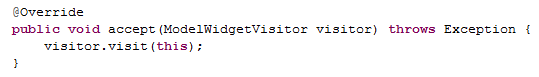
We reached this conclusion by analyzing the very explicit class name, the fact that it extends the class ModelScreenWidget and the fact that it saves a configuration about the component as a parameter.

The purpose of this class could be that the programmer may be interested to develop a generic class that is able to handle all types of widget that you want to show to screen.

## **CODE INSPECTION**

## **NAMING CONVENTIONS**

1. All class names, interface names, method names, class variables, method variables, and constants used have meaningful names and do what the name suggests. Exception made for the following method appearing in some classes of code, by the way this is an override method so it is unavoidable in any way except by changing the name on the class from which we extend the class.



1. There aren’t one-character variables even in the loops present in the code they are declared with meaningful names.
2. All the Class names are correctly capitalized.
3. All the Interfaces are correctly capitalized (by the way in the code no interface is defined and not even used to extend some classes).
4. All the methods respect the correct indentation.
5. All the class variables respect the correct indentation.
6. There are no constant declares.

## **INDENTION**

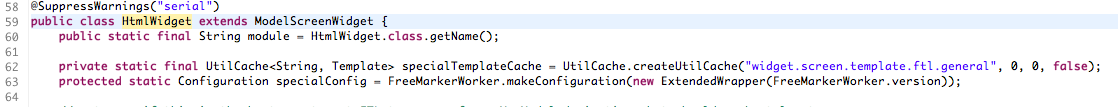
1. Four spaces are used for indention and this is done consistently.
2. No tabs are used for indention.

## **BRACES**

1. All the braces follow the “Kernighan and Ritchie” style.
2. All the loops constructors are correctly indented.

## **FILE ORGANIZATION**

1. Blank lines are used to separate package and import statements correctly. Then some inconsistent uses of blank lines and comment have been found. We present the parts of the code in which there are these inconsistence:
   1. At line 61 a blank line is used to separate one final attribute of the HtmlWidget class from the other two final attributes. This is done no more in the following.



* 1. Blank lines are correctly used to separate class variables and attributes declaration from methods.
  2. In general when a class or a nested class has neither variables nor attributes, methods are not separated by a blank line from the class declaration. This is done just one time inconsistently in line 96. Moreover, methods are always separated by a blank line to each other, except from one time in line 89.



* 1. In general comments are used consistently: if they are inserted before a class or a method in order to explain the content of those, they are separated by a blank line from the previous line containing code and they are not separated by the class or method to which they refers; when they are within a nested class or a method they are not separated from the rest of the code and they are inserted before the part of the code to which they refer.

1. The code is composed of 311 lines and the 87,78% of these do not have length greater than 80 characters.
2. Of the percentage of lines that exceed the 80 characters of length, the 18,42% of lines have a length greater than 120 characters.

## **WRAPPING LINES**

1. Line break occurs after a comma or an operator
2. Higher-level breaks are used.
3. All the expression on the same level begin at the same line as the previous one.

## **COMMENTS**

1. Comments are used to explain what the code is doing and how it has been implemented.
2. There is some commented out code without any explanation.



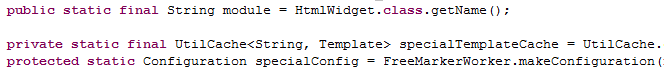
## **JAVA SOURCE FILE**

* + 1. HtmlWidget contains just one public class (in which there are nested a set of public static classes) and does not contain any interface implementation.
    2. The public class is the first and only class and interface in the file.
    3. There are no classes that implement any interface, by the way all the nested public static classes in HtmlWidget.java extends some other class in the correct way.
    4. The Javadoc is incomplete it covers just the main class but it is not extended to all of the nested classes.

## **PACKAGE AND IMPORT STATEMENTS**

* + 1. There is one package statement and it is the first non comment statement. Import statements follow.
  1. **CLASS AND INTERFACE DECLARATIONS**
     1. The class Html Widget doesn’t respect the overall list (**f** and **g** in particular) because it doesn’t group up the nested class declaration, to be more precise the structure of the code is:
* nested class declarations (for example: **public static class ExtendedWrapper** line 66);
* constructor of the external class (**public HtmlWidget(… )** line 112);
* nested class declaration (for example: **public static class HtmlTemplate** line 201);

The class Html Widget doesn’t respect the point **d** of the static variable.

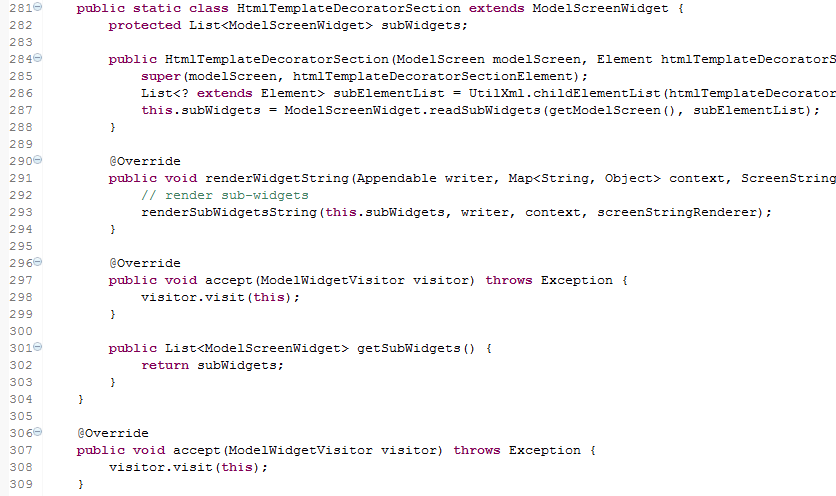


In fact, the **private static final variable specialTemplateCache** is declared before the **protected static variable specialConfig.**

* + 1. Methods are not correctly grouped, in particular the methods of the class Html Widget are scattered among the declarations of the nested classes.



It is notable that the constructor of the class is placed at line 112 and then it is followed by some of his method, but then at line 281 is declared another nested static class and on line 306 another method of the main class.



* + 1. We are not considering the subclasses as part of the code of the main class so we cannot consider HtmlWidget a big class (is a class that has to contain a lot of thing that are well nested by the way), this reasoning also applies to the duplicate check.

There aren’t any long methods.

## **INITIALIZATION AND DECLARATION**

* + 1. Variables and class members are of the correct type and they have the right visibility.
    2. Variables are declered in the proper scope.
    3. Constructors are called when a new object is desired.
    4. All object references are initialized before use.
    5. Variables are initialized where they are declared, exception done for those variables that depend upon a computation.
    6. Declaration do not always appear at the beginning of a block. Is the case of variable **private** **final** List<ModelScreenWidget> subWidgets;

## **METHOD CALLS**

* + 1. All the parameters are in the correct order.
    2. In the code, every correct method is called.
    3. All the method returned value is used properly.

## **ARRAYS**

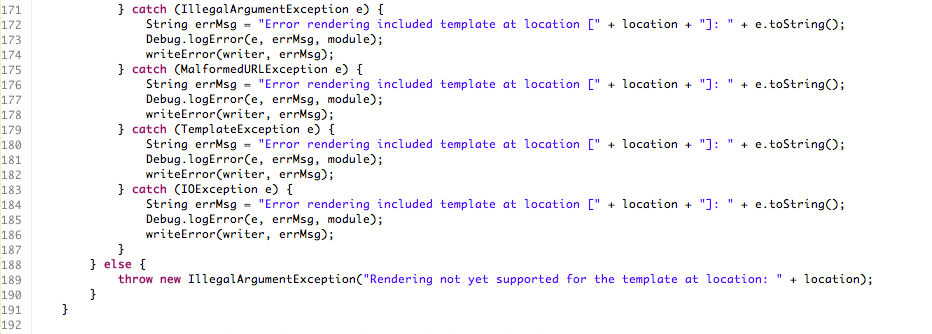
* + 1. In the code there are no array types. However several kind of collection are used (Collection, List, ArrayList). These are all implemented in the correct way and accessed correctly through indexes (or, through elements of the collection).
    2. All the collections presented have been prevented from going out-of-bound as they are initialized when declared correctly, and then they are used but never modified.
    3. Constructors are correctly called when a new array items is desired.

## **OBJECT COMPARISON**

* + 1. All the objects are compared with the equals method.

## **OUTPUT FORMAT**

* + 1. Displayed output is free of spelling and grammatical errors.
    2. Error message are comprehensive though they do not include information about how to correct the problem.



* + 1. The output is formatted correctly in terms of line stepping and spaces.

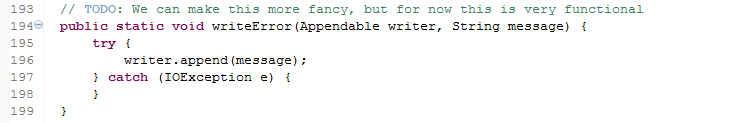
## **COMPUTATION, COMPARISONS AND ASSIGNMENTS**

* + 1. HtmlWidget itself doesn’t implements any particular algorithm and uses as loops just the for each operator which the simplest and most commonly used loop operator, for this reason we cannot consider any cases of “brutish programming” in the implementation.
    2. The correct order of computation/evaluation, operator precedence and parenthesis is respected
    3. We make notice that in this particular case that the parentheses are not necessary but make the code more readable anyway:



The parenthesis, after the “!” operator, are not needed but make the code more readable and so we should allow this notation.

* + 1. There are no arithmetic equations in the code.
    2. There are no arithmetic equations in the code.
    3. All the Boolean operators are correct.
    4. Some throw-catch expressions are not sufficiently legitimated:



Even the programmer is stating that this has to be improved, by the way we need at least to keep in mind with the correct documentation this piece of code and maybe improve it later.

* + 1. There are no implicit type conversions in the code, there are some conversion type in the code but they are explicit.

## **EXCEPTIONS**

* + 1. The relevant exception are caught.
    2. The appropriate action is taken for each catch block.

## **CONTROL FLOW**

* + 1. There are no switches in the code.
    2. There are no switches in the code.
    3. All the loops are correctly formed, by the way they are all **for each** loop so we have to check if they run on a correctly initialized list of Objects and then we are sure that there is no way they are going to generate a run-time error (caused by the loop itself).

## **FILES**

* + 1. There are no file declarations and openings.
    2. Thus, there are no file closings.
    3. EOF conditions aren’t handled.
    4. No file exceptions are handled.

## **EFFORT SPENT**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Time spent [hours]** | |  |
| **Activities** | **Emanuele Chilà** | **Giorgio Lazzarinetti** | **Total** |
| Introduction, class assigned, functional role of the class | 1 | 1 | 2 |
| Code inspection | 8 | 8 | 16 |
|  | **Total ITPD** | | 18 |

## 