Software engineering 2 project
PowerEnJoy

Code Inspection Document (CID)

Authors:
Madaffari Federico (matriculation number: 792873)(Computer Science)
Mandrioli Claudio (matriculation number: 849973)(Automation and Control)

Table of Contents

Introduction

This is the code inspection document and presents the results of the inspection activity made on the assigned portion of code. The code is part of the application OFBiz that is an open source planning system by Apache.

The document is structured in three sections:

- The Assigned class section that reports the class to be inspected.
- The Functional role section that describes the functionality provided by the assigned class and describes how those information where retrieved.
- The Code inspection that describes point by point the inspection of the code made going through the provided checklist.

Assigned class

This document presents the inspection of the class UtilCodec.java in the package org.apache.ofbiz.base.util of the application Apache OFBiz.

Functional role

The retrieval of the functionality of the class was performed completely by reading the code and manually executing some parts of it. This was necessary as comments or javadoc that present this kind of information are completely absent. According to who performed the inspection this strong lack of comments is critical.

The class, as the name suggests, offers a set of useful static methods that can be used for encoding and decoding strings. In the following we list those methods and briefly introduce them.

- getEncoder: that returns a class that manages the encoding of a string of the type passed as parameter to the getEncoder method. Available types are: html, xml, url, string.
- getDecoder: that returns a class that manages the decoding of a string of the type passed as parameter to the getDecoder method. As of now only the url decoder is implemented because of this it returns the url decoder if it is the required one or null in the other cases.
- canonicalize: that retrieves the decoding of the string passed as input and especially evaluates the presence of nested codifications. It returns the decoded string. Moreover the method points out the presence multiple or mixed encodings and according to two Boolean parameters it can throw an exception or display a warning.
- checkStringForHtmlStrictNone: that in first place decodes the input string using the canonicalize method (with the mentioned Boolean parameters set to true) and then verifies the presence of char “<” or “>” in order to spot html code. A

comment suggest to extend this research also to other char typical of html code. If no html code is found the decoded string is simply returned, otherwise before regularly return the string an error message is added to a error list passed as input.

Moreover in this class is defined another static class HtmlEncodingMapWrapper that is very poorly and badly commented and it wasn't possible in this context to find out its effective utility.
In this class through a static method (getHtmlEncodingMapWrapper) is possible to initialize the class variables: internalMap of type Map<Key, Object> and encoder of type SimpleEncoder. All methods except one provide some information over the map calling predefined methods of java. The only exception is the get method: this method does the encoding of the value associated to the key passed as input.

Code inspection

The inspection of the document was performed following the checklist and reading the whole code for each group of points.
Here a number preceded by C refers to a point of the code inspection checklist and a number preceded by a L refers to a line of code.

Naming conventions

C1: The following list presents the attributes that have a name that could be more meaningful:

- module, L41: the variable containing the class name;
- working, L188: a variable referencing a copy of a string passed as a parameter;
- mixedCount, L190: the number of different codecs found to be used for an encoded string;
- foundCount, L191: the number of times a string has been encoded.
- clean, L192: a variable used as a boolean in a while loop, if true it means that a string has been fully decoded;
- old, L200: a variable referencing a string before its decode;

C2: The only one-character variables are found at L197, an "i" character is used as an identifier for an iterator, and at L251, an "e" character is used as an identifier for an error exception in a catch block.
C3: Every class names are nouns, in mixed case, with the first letter of each word in capitalized.
C4: Interface names are capitalized like classes.
C5: Every method names is a verb in camel case.
C6: All class variables are camel case.
C7: All constants are declared using all uppercase with words separated by an underscore (L71, L102).

Indention

C8: Tabs are used in place of spaces for indentation.

C9: Tabs are used to indent code throughout the whole class and project, buti t is done consistently.

Braces

C10: The bracing style is consistent except for the code blocks that have only one statement (L292-L295, L309-L317). The Kernighan and Ritchie style is used.
C11: Every block of code that has only one statement to execute is surrounded by curly braces except on L246 where an 'If' condition has its statement on the same line without it being surrounded by curly braces.

File organization

C12: Blank lines and optional comments are used to separate sections .
C13/C14: The following lines do not respect the prescribed standards (number of characters per line is reported):

- L83, 139 char
- L84, 89 char
- L179, 91 char
- L183, 98 char
- L220, 121 char
- L226, 89 char
- L232, 86 char
- L245, 114 char
- L253, 200 char
- L254, 167 char
- L259, 113 char
- L271, 120 char
- L295, 92 char
- L305, 115 char
- L309, 83 char
- L311, 96 char

Wrapping lines

C15: Every Line break occurs after a comma or an operator.
C16: Higher level breaks are used.
C17: Every new statement is aligned with the beginning of the expression at the same level as the previous line.

Comments

C18: The comments provided are very limited and are not used to adequately explain what the class, blocks of code and methods do.
C19: There is no commented out code.

Java source files

C20/C21: The inspected file contains 2 public interfaces and 7 public classes.
C22/C23: Only one method is presented with the javadoc and it isn't complete at all.

Package and import statements

C24: The package statement is the first non-comment statement.

Class and interface declarations

C25: The method at L270 does not respect the canonical structure of java code. Other methods are fine.
C26: Methods are correctly grouped.
C27: The only long method is canonicalize at L183 that consists of 50 lines of code, anyhow it is well structured and easy to comprehend. The cohesion of the inspected code is high as methods frequently call each other.

Initializations and declarations

C28: Data types are correctly defined. Variables and class members visibilities are also correctly defined: almost all visibilities are public as the methods will be called in other classes, the few private variables are only used in the related methods.
C29: All variables are declared in the proper scope.
C30: Constructors are always called when new objects are desired.
C31: All object references are initialized before the use.
C32 : codecs at L46 is the only variable that is not initialized immediately when declared and this is correct as its definition depends on a computation.
C33: In the method canonicalize at L183 the variables are declared only after checking whether the input is null or not. In the method HtmlEncodingMapWrapper at L270 variables are declared after a method definition. In the method at L271 variables are declared only after checking whether the input are null or not.

Method calls

C34: Method parameters are correctly called.
C35: Even though there are many methods with similar names in this portion of code no errors of this type where found.
C36: Values returned by the methods are used properly.

Arrays

C37/C38: Arrays are always treated as a whole and therefore there cannot be issues related to the indexing.
C39: The only allocated list is at line 48 and the constructor is correctly called.

Object comparison

C40: All objects are compared with equals except when the comparison is made with null and the == operator is correctly used.

Output format

C41: Output is free of spelling and grammatical errors.
C42: Any of the error and warning messages do not provide any information about how to correct the code.
C43: The output is formatted correctly.

Computation, comparisons and assignments

C44: No brutish programming is present in the code.
C45/C46: No issues related to parenthesizing and precedents are present in the code.
C47: No divisions are present in the code.
C48: As there are no divisions no truncation or rounding problems can be present.
C49: Comparison and Boolean operators are correct.
C50: In the code block try-catch at L129 the try block may throw and exception that is not catch.
C51: In every assignments there is not implicit cast of any variable.

Exceptions

C52: The method canonicalize at L183 throw an IntrusionException which is correctly catched by the method checkStringForHtmlStrictNone at L245.
C53: Every time an exception is catched, a debug message is printed to the console.

Flow of control

C54/C55: There are no switch statements anywhere in the code
C56: There are two nested while-loop at L193, both correctly formed.

Files

C57/C58/C59/C60: Our class does not work with files.

Hours of work

Madaffari Federico

- 25/01/17 3h group work
- 26/01/17 2h
- 04/02/17 6h group work

Mandrioli Claudio

- 25/01/17 3h group work
- 27/01/17 0.5h
- 04/02/17 5h group work