Politecnico di Milano

PowerEnjoy Service - Code Inspection Assignment

February 4, 2017

Version 1.1

Authors:

- Domenico FAVARO (Mat. 837995)
- Matheus FIM (Mat. 876069)
- Caio ZULIANI (Mat. 877266)

Prof. Elisabetta DI NITTO

Contents

1 Class assigned - SeoConfigUtil **2**
1.1 Functional Role of SeoConfigUtil 2

2 Issues - Code Inspection Checklist **2**
2.1 Naming Conventions 2
2.2 Identation 3
2.3 Braces 3
2.4 File Organization 4
2.5 Wrapping Lines 4
2.6 Comments 4
2.7 Java Source Files 4
2.8 Package and Import Statements 4
2.9 Class and Interface Declarations 4
2.10 Initialization and Declarations 5
2.11 Method Calls 5
2.12 Arrays 5
2.13 Object Comparison 5
2.14 Output Format 5
2.15 Computation, Comparisons and Assignments 6
2.16 Exceptions 6
2.17 Flow of Control 6
2.18 Files 6

3 Others Problems **6**

4 References **8**
4.1 Used Tools 8
4.2 Effort Spent 9

5 Changelog **10**

Class assigned - SeoConfigUtil

The Java Class assigned to this document is `SeoConfigUtil`, a class that is part of the `org.apache.ofbiz.product.category` package of the Apache OFBIZ (Open For Business) Framework.

Functional Role of SeoConfigUtil

The function of this Class as its name states is SEO Configuration File Utility, SEO stands for Search Engine Optimization. Its main function is to parse a XML Cofiguration File `SeoConfig.xml` and store all the elements, URL patters, Categories and filters stated on that configuration document, using a Perl Compiler for Regular Expressions, and then offers methods that answer wether the Category URLs are enabled, wether they can be accesed by Users or Anonymous IDs on the Server and other important values for the SEO of the Web Page.

For this the Class consists on an `init()` method that initializes the class by parsing the XML file, and other access methods that return, if the initialization was successful, the elements of the SEO Configuration.

Issues - Code Inspection Checklist

Naming Conventions

SeoConfigUtil respects naming conventions throughout the file. To illustrate some important points:

1. All method names are meaningful and describe exactly what the method does. Ex:

   - (l.409) `isCategoryNameEnabled()` checks whether category name is enabled.
   - (l.436) `isJSessionIdUserEnabled()` checks whether jsessionid is enabled for user.

   With the exception of maybe the `init()` method that consists on the initialization of the class, but does a lot of parsing of the Configuration File, in fact a 230 lines method that does not call any other methods inside the Class could be considered for break up on sub methods. However this is a method that should be called once for the entire Class and having an `init()` method that just calls an equally long `parseConfigFile()` may be over complicating.

2. One character variables are used only in temporary variables, inside `for` loops (`i,j`) or to catch exceptions (`e`). Other temporary variables that do not include loops or exceptions have meaningful names inside the context they are used, like `pattern` that is initialized as the compiled URL pattern of the Perl Compiler.

3. The only Class name declared in this file is `SeoConfigUtil` and it respects the capitalized letter camelCase for its noun.

4. No Naming Conventions for Interfaces for No Interfaces are declared in this file.

5. All methods are verbs, almost all are getters like `getSpecialProductId()`. Those that return a `boolean` value are posed as a question like `isInitialed()` with the exception of `checkUseUrlRegexp()` and `checkCategoryUrl()` that to respect the naming convention should be called `canUseUrlRegexp()` and `isCategoryUrlEnabled()`

6. All class variables are declared as camelCase. That is mixed case, starting with a lowercase first letter and all theremaining words in the variable name have their first letter capitalized. Ex: `userExceptionPatterns`.

7. All constants, mostly Strings are declared with the all Caps snake case or `SCREAMING_SNAKE_CASE` Ex: `ELEMENT_URL_CONFIG`.

Identation

For identation four spaces are used consistently and no tabs are used to indent.

Braces

The Braces ident style used in this Class is the K&R ("Kernighan and Ritchie's") style, first brace is on the same line of the instruction that opens the new block, with the Java variant, that is also for class or method declarations. Ex:

```
public static boolean isInitialed() {
    return isInitialed;
}
```

File Organization

The Class follows the separation of each method by comments and even inside the `init()` method comments are used to separate the different parts.
It does not respect however the 80 or either the 120 characters line length limit, due to the highly explicative variable names and numerous Debug logs in the `init()` method, making it not an easy method to read.

Wrapping Lines

As no limit is respected for the line lenght, no wrapping of lines is used, no line breaks. All lines finish at the end of their current expression with ; or { }

Comments

The Comments in the file are adequately explaning the function of the classes. They are also used to separate and explain the diferentes parts of the class `init()` .

Java Source Files

The file has one first top-level public class named as the file's name. It also has consistency when becomes of the javadoc in such parameters as that all the classes of the file have the parameter that the javadoc expects also in the reference of the external program interfaces.

Package and Import Statements

The class assigned requires one package and as was expected it is placed as the first non-coment statement of the file.

Class and Interface Declarations

The classes of the file follows the order that was stated. For example the principal class "SeoConfigUtil" has one comment on the begining explaning the function of the class; the only class static variable is of the type private and is positioned in the after the class statement and they are followed by the methods as this class don't have contructors. The others classes of the file also follow the pattern that was stated.

Initialization and Declarations

All the variables were declared in accordance with the classes types and were declared in the correct scope. Moreover when a object is needed there are constructors positioned in the correct form also the variables used for the these objects were initialized in the beggining of the code as is recomended. All variables were declared and initialized in the beggining of the blocks.

Method Calls

The use of Methods in the file are in accordance with the expected. The ones that have parameters are presented in a correct order as well as the ones that are void don't have any parameters, in other words void methods are not passing parameters. In the calling of the methods they are correct, they refer to existing methods with correspondent assignature. All the returns of the methods are returning values as expected as the following example points were the return 'specialProductIds.containsKey(productId)' were declared as boolean.

```
public static boolean isSpecialProductId(String productId) {
        return specialProductIds.containsKey(productId);
    }
```

Arrays

The file assigned does not deal with arrays.

Object Comparison

Overall object comparison is handled in the right way. One may argue that the comparison of the string contextPath in the isCategoryUrlEnabled method is not according to the rule of using **equals** instead of ==. But, in this specific case, the comparison is not in regard to the value saved in the variable but wheter or not it is a null pointer. Therefore, both approaches can be considered as correct.

Output Format

This class does not have a formated output, as it is a parser from a XML configuration file that stores the settings stated in the document on the program context.
A number of error messages and warnings are displayed accordingly to the situation that caused the undesired behavior to occur. A good practice adopted with the warning messages is showing the information in detail,

specifying the type of warning and in which part of the XML file it happened. As for the errors, only the kind of exception that was raised is shown, but it is already enough to identify and correct the issue.

Computation, Comparisons and Assignments

It is a fairly simple class as there are no significant logical statements, causing little concern in this area. But it does make frequent use of booleans and throw-catch expressions.
The booleans are not a reason for concern as, instead of being used for operations, are mostly set in the parsing process and returned in the access methods.
The throw-catch expressions on the other hand could be a problem as in the parsing process many different exceptions can occur. Fortunately, also in this point the class is well designed and the correct error condition are checked. As an example the MalformedPatternException is always catch after a pattern is handled by the perl compiler.

Exceptions

The class checks for parsing specific exceptions. Examples are the perl compiler not finding what it was intended and problems with the XML file structure.
Also, it looks for common exceptions such as incorrect usage of a null pointer, inappropriate conversion of string to integer and input/output errors.
The catch blocks does no action besides of notifying what was the undesired behavior that occurred. Which is in accordance with the comportment expected from a parser.

Flow of Control

No switch statement is used and the four for loops are correctly formed.

Files

The class uses only one XML file, and takes care of making the correct procedure of opening, closing, and catching file related exceptions.

Others Problems

Even though this is not a complex class, a good amount of time is needed to understand how it functions. If, together with the code, it was pro-

vided information on the structure of SeoConfig.xml this time would drop significantly.

References

Used Tools

- **GitHub:** for Version Control
- **Notepad++:** for Java Code Editing
- **TeXworks:** for LaTex editing of this Document

Effort Spent

Date	Domenico	Caio	Matheus
30/01/17	1h	1h	1h
31/01/17	2h	-	-
01/02/17	-	-	-
02/02/17	2h	-	-
03/02/17	-	3h	2h
04/02/17	-	-	3h
05/02/17	-	-	-

Changelog

As the project and design decisions may change during the development this document is also prone to change. We'll document every version in this part.

- **Version 1.1:** 05/02/2017