



**POLITECNICO  
DI MILANO**

## CODE INSPECTION

*Hasancan Sayılan*

## Contents

1. INTRODUCTION .....	3
1.1 Purpose.....	3
1.2 Scope .....	3
1.3 Reference Documents .....	3
1.4 Overview.....	3
2. CLASSES ASSIGNED .....	4
3. FUNCTIONAL ROLE .....	4
4. LIST OF ISSUES CONSIDERING CHECKLIST .....	4
4.1 Naming Conventions .....	4
4.2 Idention .....	6
4.3 Braces .....	6
4.4 File Organization.....	6
4.5 Wrapping Lines.....	7
4.6 Comments .....	7
4.7 Java Source Files .....	8
4.8 Package and Import Statements .....	9
4.9 Class and Interface Declarations .....	10
4.10 Initialization and Declerations.....	11
4.11 Method Calls.....	12
4.12 Arrays.....	12
4.13 Object Comparison.....	13
4.14 Output Format.....	13
4.15 Computation, Comparisons and Assignments .....	13
4.16 Exceptions .....	14
4.17 Flow of Control .....	14
4.18 Files.....	15
5. ADDITIONAL NOTES.....	16
HOURS OF WORK.....	16
REFERENCES .....	16
Used Tools .....	16

# **1. INTRODUCTION**

## **1.1 Purpose**

This document is made with the purpose of analyzing the source code of Apache Ofbiz project and according to the checklist, finding the mistakes and reporting them.

## **1.2 Scope**

The analysis is made using the checklist provided by the course professors as a base document to find mistakes and point them out.

## **1.3 Reference Documents**

- Apache Ofbiz Project
- Code Inspection Assignment Task Description

## **1.4 Overview**

The document is made of five main sections which are :

- Introduction : This section explains the purpose and the scope of the project.
- Classes Assigned : The source code that the team checks is given in this section.
- Functional Role : Functional roles are checked in this section.
- List of Issues by Checklist : Code fragments which are not completely in use or have mistakes are checked in this section.

## 2. CLASSES ASSIGNED

This document explains and check out the source code of the project Apache Ofbiz project with the source code :

```
./apache-ofbiz-  
16.11.01/framework/minilang/src/main/java/org/apache/ofbiz/minilang/method/conditi  
onal/CombinedCondition.java
```

## 3. FUNCTIONAL ROLE

This section explains the CombinedCondition class.

The CombinedCondition.class is a java class which is related to MiniLanguage under the project named Apache Ofbiz. MiniLanguage is a project that aims to help business in terms of software by simplifying the process of software developments and projects. With this project, people with little experience in software departments can understand and even contribute to the development of software projects and this will increase the effectiveness of the development process.

CombinedCondition.java is a subclass of this project. It implements some conditions on the environment where the Ofbiz might be in use. It validates Ofbiz, and check the conditions using the simple real-life relations “AND”, “IF”, and “NOT”.

## 4. LIST OF ISSUES CONSIDERING CHECKLIST

In this section the codes of the project will be examined and the errors will be pointed out according to their different patterns.

### 4.1 Naming Conventions

*1) All class names, interface names, method names, class variables, method variables, and constants used should have meaningful names and do what the name suggests.*

```

55     protected void prettyPrint(StringBuilder messageBuffer, MethodContext methodContext, String combineText) {
56         messageBuffer.append("(");
57         for (Conditional subCond : subConditions) {
58             subCond.prettyPrint(messageBuffer, methodContext);
59             messageBuffer.append(combineText);
60         }

```

- In this example, prettyPrint is not a meaningful name.

2) *If one-character variables are used, they are used only for temporary “throwaway” variables, such as those used in for loops.*

- No one-character variable is used in this code.

3) *Class names are nouns, in mixed case, with the first letter of each word in capitalized.*

- All class names are nouns and the first letter of each word in each class are capitalized.

4) *Interface names should be capitalized like classes.*

- No interface is used in this code.

5) *Method names should be verbs, with the first letter of each additional word capitalized.*

- Even though some method names violate the first rule of naming conventions, all of them are verbs with the first letter of each additional word is capitalized.

6) *Class variables, also called attributes, are mixed case, but might begin with an underscore (‘\_’) followed by a lowercase first letter. All the remaining words in the variable name have their first letter capitalized.*

- There are no class variables starting with an underscore but this rule is not restricted like the others hence no attribute name violates this rule.

7) *Constants are declared using all uppercase with words separated by an underscore.*

- Not applicable for this source file.

## 4.2 Indention

1) *Three or four spaces are used for indention and done so consistently.*

- Indention fits the conditions of this rule.

2) *No tabs are used to indent.*

- Indention fits the condition of this rule.

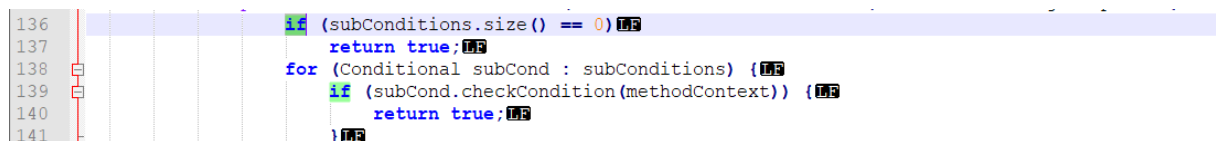
## 4.3 Braces

1) *Consistent bracing style is used, either the preferred “Allman” style (first brace goes underneath the opening block) or the “Kernighan and Ritchie” style (first brace is on the same line of the instruction that opens the new block).*

- All braces fits the condition of this rule.

2) *All if, while, do-while, try-catch, and for statements that have only one statement to execute are surrounded by curly braces.*

- There are some “IF” statements that violate this rule such as in the example.



```
136 if (subConditions.size() == 0) {
137     return true;
138 }
139 for (Conditional subCond : subConditions) {
140     if (subCond.checkCondition(methodContext)) {
141         return true;
142     }
143 }
```

- The differences between the two “IF” statements in the picture is obvious. The first one has no curly braces in the beginning and the end of its statement but the second one does.

## 4.4 File Organization

1) *Blank lines and optional comments are used to separate sections (beginning comments, package/import statements, class/interface declarations which include class variable/attributes declarations, constructors, and methods).*

- Blank lines are used to separate sections like packages, import statements, class declarations etc. There are beginning comments to start the file and comments to separate sections like blank lines hence file organization fits the condition of this rule.

2) *Where practical, line length does not exceed 80 characters.*

- There are some lines that exceed 80 characters hence those lines does not fit this condition and some changes at those lines could be a necessity.

```
132 | public CombinedCondition createCondition(Element element, SimpleMethod simpleMethod) throws MiniLangException {
```

```
100 | public CombinedCondition createCondition(Element element, SimpleMethod simpleMethod) throws MiniLangException {
```

3) *When line length must exceed 80 characters, it does NOT exceed 120 characters.*

- There are no lines exceeding 120 characters in practical codes. There are some comments that exceed 120 characters but since they are not practical lines they do not violate this condition.

## 4.5 Wrapping Lines

1) *Line break occurs after a comma or an operator.*

- Each line breaks after each comma.

2) *Higher-level breaks are used.*

- There are no higher-level breaks in the code.

3) *A new statement is aligned with the beginning of the expression at the same level as the previous line.*

- This condition cannot be considered because there are no higher-level breaks in the code as mentioned in 4.5.3

## 4.6 Comments

1) *Comments are used to adequately explain what the class, interface, methods, and blocks of code are doing.*

- The comments which are used in the code are partially not understandable. Some comments are just abbreviations of the actual code and some comments are just directing to a link which does not open a website or a resource.

```
33 | /**
34 |  * Implements the &lt;and>, &lt;or>, &lt;not>, and &lt;xor> elements.
35 |  *
36 |  * @see <a href="https://cwiki.apache.org/confluence/display/OFBADMIN/Mini-language+Reference#Mini-languageReference-Conditional%2FLoo
37 |  */
```

2) *Commented out code contains a reason for being commented out and a date it can be removed from the source file if determined it is no longer needed.*

- There are no comments fitting this condition.

## 4.7 Java Source Files

1) Each Java source file contains a single public class or interface.

- There are many public classes using in the source file.

```

161 public static final class XorConditionFactory extends ConditionalFactory<CombinedCondition> {
162     @Override
163     public CombinedCondition createCondition(Element element, SimpleMethod simpleMethod) throws MiniLangException {
164         return new CombinedCondition(element, simpleMethod);
165     }
166     @Override
167     public boolean checkCondition(MethodContext methodContext) throws MiniLangException {
168         if (subConditions.size() == 0)
169             return true;
170         boolean trueFound = false;
171         for (Conditional subCond : subConditions) {
172             if (subCond.checkCondition(methodContext)) {
173                 if (trueFound)
174                     return false;
175                 else
176                     trueFound = true;
177             }
178         }
179         return trueFound;
180     }
181     @Override
182     public void prettyPrint(StringBuilder messageBuffer, MethodContext methodContext) {
183         prettyPrint(messageBuffer, methodContext, " XOR ");
184     }
185 }
186
187
188 @Override
189 public String getName() {
190     return "xor";
191 }
192
193
194

```

```

38 public abstract class CombinedCondition extends MiniLangElement implements Conditional {
39
40     protected final List<Conditional> subConditions;
41
42     public CombinedCondition(Element element, SimpleMethod simpleMethod) throws MiniLangException {
43         super(element, simpleMethod);
44         List<? extends Element> childElements = UtilXml.childElementList(element);
45         if (MiniLangValidate.validationOn() && childElements.isEmpty()) {
46             MiniLangValidate.handleError("No conditional elements.", simpleMethod, element);
47         }
48         List<Conditional> conditionalList = new ArrayList<Conditional>(childElements.size());
49         for (Element conditionalElement : UtilXml.childElementList(element)) {
50             conditionalList.add(ConditionalFactory.makeConditional(conditionalElement, simpleMethod));
51         }
52         this.subConditions = Collections.unmodifiableList(conditionallist);
53     }
54
55     protected void prettyPrint(StringBuilder messageBuffer, MethodContext methodContext, String combineText) {
56         messageBuffer.append("(");
57         for (Conditional subCond : subConditions) {
58             subCond.prettyPrint(messageBuffer, methodContext);
59             messageBuffer.append(combineText);
60         }
61         messageBuffer.append(")");
62     }
63
64     /**
65      * A &lt;and&gt; element factory.
66      */

```

2) The public class is the first class or interface in the file.

- The first class of the file is the public class hence the file fits this condition. The first class of the source code is given in the following figure.



```

38 public abstract class CombinedCondition extends MiniLangElement implements Conditional {
39     protected final List<Conditional> subConditions;
40
41     public CombinedCondition(Element element, SimpleMethod simpleMethod) throws MiniLangException {
42         super(element, simpleMethod);
43         List<? extends Element> childElements = UtilXml.childElementList(element);
44         if (MiniLangValidate.validationOn() && childElements.isEmpty()) {
45             MiniLangValidate.handleError("No conditional elements.", simpleMethod, element);
46         }
47         List<Conditional> conditionalList = new ArrayList<Conditional>(childElements.size());
48         for (Element conditionalElement : UtilXml.childElementList(element)) {
49             conditionalList.add(ConditionalFactory.makeConditional(conditionalElement, simpleMethod));
50         }
51         this.subConditions = Collections.unmodifiableList(conditionallist);
52     }
53
54     protected void prettyPrint(StringBuilder messageBuffer, MethodContext methodContext, String combineText) {
55         messageBuffer.append("(");
56         for (Conditional subCond : subConditions) {
57             subCond.prettyPrint(messageBuffer, methodContext);
58             messageBuffer.append(combineText);
59         }
60         messageBuffer.append(")");
61     }
62 }

```

3) Check that the external program interfaces are implemented consistently with what is described in the javadoc.

- It is not possible to detect external program interfaces because javadoc does not point them out completely.

4) Check that the javadoc is complete (i.e., it covers all classes and files part of the set of classes assigned to you).

- There are some missing sections in Javadoc such as some methods hence it is not complete.

## 4.8 Package and Import Statements

1) If any package statements are needed, they should be the first noncomment statements. Import statements follow.

- There is only one package statement in the source file and it is the first noncomment statement. After it, there are import statements hence the file fits this condition.

```

1  /*****
2  * Licensed to the Apache Software Foundation (ASF) under one
3  * or more contributor license agreements. See the NOTICE file
4  * distributed with this work for additional information
5  * regarding copyright ownership. The ASF licenses this file
6  * to you under the Apache License, Version 2.0 (the
7  * "License"); you may not use this file except in compliance
8  * with the License. You may obtain a copy of the License at
9  *
10 * http://www.apache.org/licenses/LICENSE-2.0
11 *
12 * Unless required by applicable law or agreed to in writing,
13 * software distributed under the License is distributed on an
14 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
15 * KIND, either express or implied. See the License for the
16 * specific language governing permissions and limitations
17 * under the License.
18 *****/
19 package org.apache.ofbiz.minilang.method.conditional;
20
21 import java.util.ArrayList;
22 import java.util.Collections;
23 import java.util.List;
24
25 import org.apache.ofbiz.base.util.UtilXml;
26 import org.apache.ofbiz.minilang.MinilangElement;
27 import org.apache.ofbiz.minilang.MinilangException;
28 import org.apache.ofbiz.minilang.MinilangValidate;
29 import org.apache.ofbiz.minilang.SimpleMethod;
30 import org.apache.ofbiz.minilang.method.MethodContext;
31 import org.w3c.dom.Element;
32

```

## 4.9 Class and Interface Declarations

1) The class or interface declarations shall be in the following order:

- a) class/interface documentation comment;
- b) class or interface statement;
- c) class/interface implementation comment, if necessary;
- d) class (static) variables;
  - I. first public class variables;
  - II. next protected class variables;
  - III. next package level (no access modifier);
  - IV. last private class variables.
- e) instance variables;
  - I. first public instance variables;
  - II. next protected instance variables;
  - III. next package level (no access modifier);
  - IV. last private instance variables.
- f) constructors;
- g) methods.

- All declarations are in that order thus the file fits this condition.

2) *Methods are grouped by functionality rather than by scope or accessibility.*

- All methods are public and fits the condition of this rule.

3) *Check that the code is free of duplicates, long methods, big classes, breaking encapsulation, as well as if coupling and cohesion are adequate.*

- There are some duplicates in the codes between the similar part of different methods used in the file but the size of methods and classes are proper but still critical.

```

67 public static final class AndConditionFactory extends ConditionalFactory<CombinedCondition> {
68     @Override
69     public CombinedCondition createCondition(Element element, SimpleMethod simpleMethod) throws MiniLangException {
70         return new CombinedCondition(element, simpleMethod);
71     }
72     @Override
73     public boolean checkCondition(MethodContext methodContext) throws MiniLangException {
74         if (subConditions.size() == 0)
75             return true;
76         for (Conditional subCond : subConditions) {
77             if (!subCond.checkCondition(methodContext))
78                 return false;
79         }
80         return true;
81     }
82     @Override
83     public void prettyPrint(StringBuilder messageBuffer, MethodContext methodContext) {
84         prettyPrint(messageBuffer, methodContext, " AND ");
85     }
86 }
87
88
89 @Override
90 public String getName() {
91     return "and";
92 }
93
94 }

```

```

98 public static final class NotConditionFactory extends ConditionalFactory<CombinedCondition> {
99     @Override
100     public CombinedCondition createCondition(Element element, SimpleMethod simpleMethod) throws MiniLangException {
101         return new CombinedCondition(element, simpleMethod);
102     }
103     @Override
104     public boolean checkCondition(MethodContext methodContext) throws MiniLangException {
105         if (subConditions.size() == 0)
106             return true;
107         Conditional subCond = subConditions.get(0);
108         return !subCond.checkCondition(methodContext);
109     }
110     @Override
111     public void prettyPrint(StringBuilder messageBuffer, MethodContext methodContext) {
112         messageBuffer.append(" NOT ");
113         if (subConditions.size() > 0) {
114             Conditional subCond = subConditions.get(0);
115             subCond.prettyPrint(messageBuffer, methodContext);
116         }
117         messageBuffer.append(" ");
118     }
119 }
120
121 @Override
122 public String getName() {
123     return "not";
124 }

```

## 4.10 Initialization and Declarations

1) *Check that variables and class members are of the correct type. Check that they have the right visibility (public/private/protected).*

- All variables have the right visibility and of the correct type.

2) *Check that variables are declared in the proper scope.*

All variables are declared in the proper scope and used properly in the source code.

3) *Check that constructors are called when a new object is desired.*

- The source code fits the condition of this rule.

4) *Check that all object references are initialized before use.*

- All object references are initialized before use.

5) *Variables are initialized where they are declared, unless dependent upon a computation.*

- All variables are initialized when they are declared and there are no null variables thus the source code fits this condition.

6) *Declarations appear at the beginning of blocks (A block is any code surrounded by curly braces '{' and '}'). The exception is a variable can be declared in a for loop.*

- All declarations appear at the beginning of the blocks.

#### 4.11 Method Calls

1) *Check that parameters are presented in the correct order.*

- All parameters are presented in the correct order.

2) *Check that the correct method is being called, or should it be a different method with a similar name.*

- All methods are given a different, understandable and distinctable names from each other. Some may similar but they are still distinct and can be understood.

3) *Check that method returned values are used properly.*

- All values returned from methods are used properly.

#### 4.12 Arrays

1) *Check that there are no off-by-one errors in array indexing (that is, all required array elements are correctly accessed through the index).*

- There are not much arrays to be considered in the code but all required elements of the existing arrays are correctly accessed through the index.

*2) Check that all array (or other collection) indexes have been prevented from going out-of-bounds.*

- *No indexes are going out-of-bounds.*

*3) Check that constructors are called when a new array item is desired.*

- As mentioned above there are not much arrays are in use in the code but the existing arrays are created after the constructors are called.

#### **4.13 Object Comparison**

*1) Check that all objects (including Strings) are compared with equals and not with ==.*

- This rule is not applicable for this source code.

#### **4.14 Output Format**

*1) Check that displayed output is free of spelling and grammatical errors.*

- There are no spelling or grammatical errors in the output.

*2) Check that error messages are comprehensive and provide guidance as to how to correct the problem.*

- The exception codes in the source file are proper and this leads to the proper error messages.

*3) Check that the output is formatted correctly in terms of line stepping and spacing.*

- The output is formatted correctly.

#### **4.15 Computation, Comparisons and Assignments**

*1) Check that the implementation avoids “brutish programming”: (see <http://users.csc.calpoly.edu/~jdalbey/SWE/CodeSmells/bonehead.html>).*

- The implementations may not enough to avoid “brutish programming”.

*2) Check order of computation/evaluation, operator precedence and parenthesizing.*

- The computation process is not in the source code hence it cannot be considered. Parenthesizing on the other hand is proper.

3) *Check the liberal use of parenthesis is used to avoid operator precedence problems.*

- Not applicable for this source code.

4) *Check that all denominators of a division are prevented from being zero.*

- Not applicable for this source code.

5) *Check that integer arithmetic, especially division, are used appropriately to avoid causing unexpected truncation/rounding. 49. Check that the comparison and Boolean operators are correct.*

- Not applicable for this source code.

6) *Check that the comparison and Boolean operators are correct.*

- The comparison and boolean operators are correct.

7) *Check throw-catch expressions, and check that the error condition is actually legitimate.*

- The throw-catch expressions are proper and based on legitimate error conditions.

8) *Check that the code is free of any implicit type conversions.*

- The code is free of implicit type conversions.

#### 4.16 Exceptions

1) *Check that the relevant exceptions are caught.*

- All exceptions are proper and relevant exceptions are caught.

2) *Check that the appropriate action are taken for each catch block.*

- There are no catch block in the source code.

#### 4.17 Flow of Control

1) *In a switch statement, check that all cases are addressed by break or return.*

- There are no switch statements.

2) *Check that all switch statements have a default branch.*

- There are no switch statements.

*3) Check that all loops are correctly formed, with the appropriate initialization, increment and termination expressions.*

- The file mostly formed of “FOR” and “IF” statements and they are used properly.

#### **4.18 Files**

*1) Check that all files are properly declared and opened.*

- There are no file operations that can be done during examination of source file.

*2) Check that all files are closed properly, even in the case of an error.*

- There are no file operations that can be done during examination of source file.

*3) Check that EOF conditions are detected and handled correctly.*

- There are no file operations that can be done during examination of source file.

*4) Check that all file exceptions are caught and dealt with accordingly.*

- There are no file operations that can be done during examination of source file.

## **5. ADDITIONAL NOTES**

The source file is very hard to read and understand since the comment lines are not descriptive and the lines are not distictly seperated from each other.

## **HOURS OF WORK**

HASANCAN SAYILAN

- 04/07/2017 : 3h
- 05/07/2017 : 8h
- 06/07/2017 : 10h
- 07/07/2017 : 4h

## **REFERENCES**

### **Used Tools**

- Github : To upload the document
- NotePad++ and Netbeans : As the environment to inspect the project.