POLITECNICO
MILANO 1863

Software engineering 2 Project

Code Inspection Document

Ofbiz

Perugini Alex	876359
Re Marco	873564
Scotti Vincenzo	875505

Table of Contents

1 Introduction

1.1 Purpose

The purpose of this document is to provide a complete report of the code inspection analysis applied to Compare.java and CompareCondition.java, two classes from the automation of enterprise system product Apache Ofbiz.

1.2 Scope

In this document we will provide both a syntactical and a semantical analysis. In fact there is a section in which we will explain what are the main functionalities of the two classes in the Apache Ofbiz. The analysis will be guided by the checklist provided inside the “Code Inspection Assignment Task Description”.

1.3 Reference documents

1. Code Inspection Assignment Task Description
2. Minilang documentation at “https://cwiki.apache.org/confluence/display/OFBADMIN/Mini+Language+-+minilang+-+simple-method+-+Reference”
3. Example documents:
    -

1.4 Document structure

The document is organized as follows:

- Section 1: Introduction, provides a general description of how the inspection of the source code from the selected classes of Ofbiz will proceed.
- Section 2: Class analysis, contains the description of the analyzed classes and their detailed inspection.
- Section 3: Used tools, contains the list of the tools used to realize the CID and the scope they were used for.
- Section 4: Hours of work, report of work time for each member.

2 Class analysis

2.1 Classes

The analyzed classes are:

- ../apache-ofbiz-16.11.01/framework/minilang/src/main/java/org/apache/ofbiz/minilang /method/conditional/CompareCondition.java

- ../apache-ofbiz-16.11.01/framework/minilang/src/main/java/org/apache/ofbiz/minilang /method/conditional/Compare.java

2.2 Functional role

2.2.1 Compare.java

Compare class uses the abstract factory pattern to provide its concrete implementation through the method getInstance(). This method returns a different instance depending on the required operator passed as a string. The right instance is found mapping the value to the INSTANCE_MAP attribute, that contains the mapping between the name of the operator and the corresponding class. Each subclass overrides the doCompare() method to perform the required operation.
Here follows a list of the classes returned by the factory pattern and a description of the overridden method:

- CompareContains: performs a check for the type and if it is a collection, checks if the collection on the left contains the collection on the right or if it is a string checks if the string on the right is a substring of the string on the left.
- CompareIsEmpty: checks if the left value does not contain something.
- CompareIsNotNull: checks if the left value is not null.
- CompareIsNull: checks if the left value is null.

The following ones perform all the same type check to verify if the input is a BigDecimal or Comparable, and then perform the same operation for both types, in particular if it is a BigDecimal the comparison is performed using the compareBigDecimals() static and private method provided by the abstract class.

- CompareEquals: checks if the values are equals.
- CompareNotEquals: checks if the values are not equals.
- CompareGreater: checks if the left value is greater than the right one.
- CompareGreaterEquals: checks if the left value is greater or equal than the right one.
- CompareLess: checks if the left value is lesser than the right one.
- CompareLessEquals: checks if the left value is lesser or equal than the right one.

Finally there are the createInstanceMap() method, used to fill the INSTANCE_MAP attribute, and the assertValuesNotNull() method, this one is used before every compare that involves both left and right values to ensure they're not null.

2.2.2 CompareCondition.java

CompareCondition class is used to evaluate a conditional expression for the minilang scripting language. An instance of this class can be built through the CompareConditionFactory where two different methods can be called: createCondition() and createMethodOperation(), the former overrides the equivalent abstract method of the abstract class and the latter implements the method declared in the interface, both these methods invoke the constructor of this class. However the constructor of the class is public so an instance of the class can also be obtained calling it directly.
This constructor initializes all the attributes by parsing an xml file. Among these there is the compare attribute which is initialized invoking the getInstance() method of the Compare class passing as parameter the operator to have the correct comparator. Also the targetClass is initialized here to the value of the parsed attribute 'type', checking if the class type is compatible with the operation. Then both subOps and elseSubOps are initialized parsing the xml element passed as input to the constructor. The subOps attribute is initialized as a unmodifiable list or set to null (if the first child is null or "else"), the elseSubOps is initialized in an analogous way reading the elseElement (child of element).
Here follows an analysis of the methods implemented in CompareCondition:

- checkCondition(): applies the doCompare() using this.compare and returns its value if no exception was thrown, otherwise adds the error message of the exception to the simpleMethod and returns false.
- exec(): given the methodContext conditions in input, performs the subOps operations if the checkCondition() is true otherwise performs the elseSubOps.
- gatherArtifactInfo(): for each methodOperation in subOps and elseSubOps invokes the gatherArtifactInfo() in order to update the ArtifactInfoContext.
- prettyPrint(): appends a string to the messageBuffer according to the operation to be done.
- toString(): builds an xml element <if-compare> with field, operator, value, type and format as attributes.

2.3 Found issues

2.3.1 Compare.java

- Naming conventions: all naming conventions are respected.
- Indention: the four spaces convention is always applied.
- Braces: the "Kernighan and Ritchie" style is maintained consistent for the whole file.
- File organization: although the blank spaces separation convention is respected. All of the doCompare() methods headers exceed the 80 and then 120 characters limit, every time the abstract method, that exceeds too, is overridden.

- Wrapping lines: at lines 109 and 116, in the if statements is not respected the line break convention after the “||” operator in the former line and the “&&” in the latter.
- Comments: although there is a valid reason for the comments to exist, they aren’t adequately detailed. Also there is no commented out code.
- Java source files: the first class in the file is public and all the others are private, so this convention is respected. Instead for what concerns the javadoc it isn’t completely detailed, in particular the description of the abstract class isn’t complete, the public method getInstance() doesn’t specify what is the parameter “operator” and the public method doCompare() doesn’t specify anything about the exceptions it throws, moreover the order of the parameters in the javadoc doesn’t coincide with the method input parameters.
- Package and import statements: the convention for the order of package and imports is respected.
- Class and interface declarations: all the class and interface declarations conventions are respected. There are some lines of duplicated code in the private classes CompareEquals, CompareGreater, CompareGreaterEquals, CompareLess, CompareLessEquals and CompareNotEquals in the inherited method doCompare(), in particular the two “if” conditions with the respective “then” branches differs only from each other for the operator (==, >, >=, <, <=, !=).
- Initialization and declarations: the visibility is supposed to be right, in fact since the class is a factory it is right for the final inner classes to be private and for the method getInstance() to be public and static. All the variables are declared in the proper scope. The constructors for the final inner classes are called once since they’re static objects. Variables are always initialized when declared, but for what concerns the BigDecimals inside the doCompare() the convention of declaring it before a block is not respected.
- Method calls: all the rules concerning method calls are respected.
- Arrays: there are no arrays in the class.
- Object comparison: there is no misuse of the “==” operator.
- Output format: the only outputs of class are error messages, that are written correctly.
- Computation, comparison and assignments: there are no brutish programming issues and all the other conventions for this point are respected.
- Exceptions: there are no “try-catch” blocks.
- Flow of control: there are no switches or loops.
- Files: there is no use of files in this class.

2.3.2 CompareCondition.java

- Naming conventions: all naming conventions are respected.
- Indention: the four spaces convention is always applied.
- Braces: the “Kernighan and Ritchie” style is maintained consistent for the whole file.
- File organization: although the blank spaces separation convention is respected. The CompareConditionFactory header in line 194 exceed the 80 and then 120 characters limit, as for line 78. In all the other cases when the 80 characters limit is exceeded the 120 is respected.

- Wrapping lines: at line 91, in the if statements is not respected the line break convention after the “&&”.
- Comments: the comment in the method checkCondition() at line 117 isn’t much clear and in line 153 there is no reason for the comment since it is empty. All the remaining comments are correct and clear. Also there is no commented out code.
- Java source files: there are two public classes in the source file, the CompareCondition class and the inner class CompareConditionFactory. The javadoc for the two classes is clearly incomplete and not specific, plus the javadoc of the CompareCondition refers to a not working web page. The javadoc for the 2 non overridden methods (prettyPrint() and createMethodOperation()) is missing (this is probably due to the absence of “@Override”). All external interfaces are implemented consistently.
- Package and import statements: the convention for the order of package and imports is respected.
- Class and interface declarations: all the class and interface declarations conventions are respected. There is no duplicated code. All the other methods doesn’t need to be grouped in a better way since there would be no criteria, with exception for prettyPrint() and toString() that are correctly grouped.
- Initialization and declarations: Almost in every method the declaration convention is not respected, in fact it happens quite often that variables are declared and initialized in the middle of a code block. Plus since there is a factory method the constructor of the class should be private.
- Method calls: all the rules concerning method calls are respected.
- Arrays: there are no arrays in the class.
- Object comparison: all the “==” and “!=” are used only to check if a variable is null or not and in all the other cases is used the equals() method, so the convention is respected.
- Output format: all the rules for the output format are respected, even the ones of the xml in the toString().
- Computation, comparison and assignments: there are no brutish programming issues and all the other conventions for this point are respected. In fact there are two “try-catch” blocks, each one is consistent and is handled without re-throwing.
- Exceptions: as stated before exceptions are correctly handled in both cases.
- Flow of control: the only 2 for loops are in the gatherArtifactInfo() method, they are generalized for loops and the not null condition is checked before beginning. There are no switch statements.
- Files: there is no use of files in this class.

2.4 Other problems

In the Compare.java class, the inner final classes could be grouped in a better way, like:

- CompareContains
- CompareIsEmpty
- CompareIsNull
- CompareIsNotNull
- CompareEquals
- CompareNotEquals
- CompareGreater
- CompareGreaterEquals
- CompareLess
- CompareLessEquals

in this way the inner classes are disposed according to their functionality.

In the CompareCondition.java, in the prettyPrint() method there is a call to the deprecated method expandString().

3. Used tools

- Google Drive: documents sharing
- Google Docs: word processor, concurrent work platform
- GitHub: control version

4. Hours of work

Perugini Alex:

- 19/01/17 30m
- 20/01/17 1h 30m
- 26/01/17 2h
- 03/02/17 3h

Re Marco:

- 19/01/17 30m
- 20/01/17 1h 30m
- 26/01/17 2h
- 03/02/17 3h

Scotti Vincenzo

- 19/01/17 30m
- 20/01/17 1h 30m
- 26/01/17 2h
- 03/02/17 3h