

POLITECNICO DI MILANO  
Scuola di Ingegneria Industriale e dell'Informazione  
M. Sc. in Computer Science and Engineering  
Dipartimento di Elettronica, Informatica e Bioingegneria

# Power EnJoy

Software Engineering 2 - Project

## CI Code inspection

Version 1.0

Authors:

Andrea BATTISTELLO

Matr: 873795

William DI LUIGI

Matr: 864165

Academic Year 2016 - 2017

<b>1. Introduction</b>	<b>4</b>
1.1. Purpose	4
1.2. Scope	4
1.3. Reference documents	4
1.4. Document structure	4
<b>2. Assigned class</b>	<b>5</b>
2.1. Functional role of the class	5
2.2. Methods	6
2.3. Inner classes	8
<b>3. Issues</b>	<b>9</b>
3.1. Naming conventions	9
3.2. Indentation	9
3.3. Braces	9
3.4. File organization	10
3.5. Wrapping lines	14
3.6. Comments	14
3.7. Java Source Files	14
3.8. Package and Import Statements	15
3.9. Class and Interface Declarations	15
3.10. Initialization and Declarations	16
3.11. Method Calls	16
3.12. Arrays	16
3.13. Object Comparison	17
3.14. Output Format	17
3.15. Computation, Comparisons and Assignments	17
3.16. Exceptions	18
3.17. Flow of Control	19
3.18. Files	19
<b>4. Observations</b>	<b>20</b>
<b>5. Appendix</b>	<b>21</b>
5.1. Used tools	21
5.2. Hours of work	21
5.3. Revision history	21



# 1. Introduction

## 1.1. Purpose

The purpose of the code inspection is to thoroughly analyze a piece of software in order to identify defects and, if necessary, improve the quality of the code. Code inspection is a specific case of static analysis as part of the V&V (Verification and Validation) process.

We will systematically go through all the points of the code inspection checklist [2] and we will mark them with a tick if that aspect is fulfilled in the code. Otherwise, we will mark them with a cross and point out the specific lines that do not fulfill the constraint and a brief explanation of how to make them compliant.

## 1.2. Scope

Apache OFBiz® is an open source product for the automation of enterprise processes that includes framework components and business applications for ERP (Enterprise Resource Planning), CRM (Customer Relationship Management), E-Business / E-Commerce, SCM (Supply Chain Management), MRP (Manufacturing Resource Planning), MMS/EAM (Maintenance Management System/Enterprise Asset Management).

## 1.3. Reference documents

- [1] Code inspection assignment task description.
- [2] Code inspection checklist for Java.
- [3] Apache OFBiz main page: <https://ofbiz.apache.org/>

## 1.4. Document structure

The document will be organized in 4 sections and an appendix.

- The introduction section, this one, describes the purpose of the document.
- In the second section, we will analyze the assigned class and its context (the OFBiz framework) from a general point of view.
- The third part will go through all the points of the code inspection checklist and see if the assigned class complies with all the points or not.
- The fourth part gives some observations and suggestions to improve the quality of the code
- The appendix describes the software we have used and the effort spent to make this document.

## 2. Assigned class

The class assigned to us is the Compare class inside the package  
/apache-ofbiz-16.11.01/framework/minilang/src/main/java/org/apache/ofbiz  
/minilang/method/conditional/Compare.java

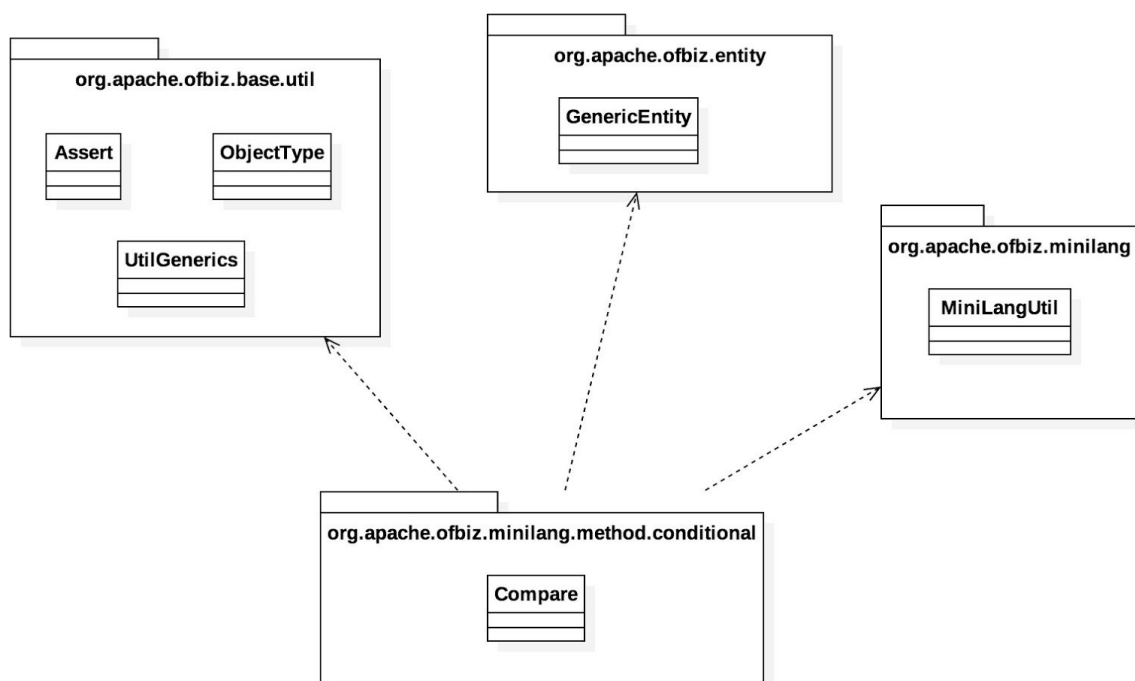
### 2.1. Functional role of the class

This class is used as a utility class to compare Objects. The main class is Compare, which is an abstract class with a public static method getInstance that returns one of the concrete instances of Compare (specified by means of a String).

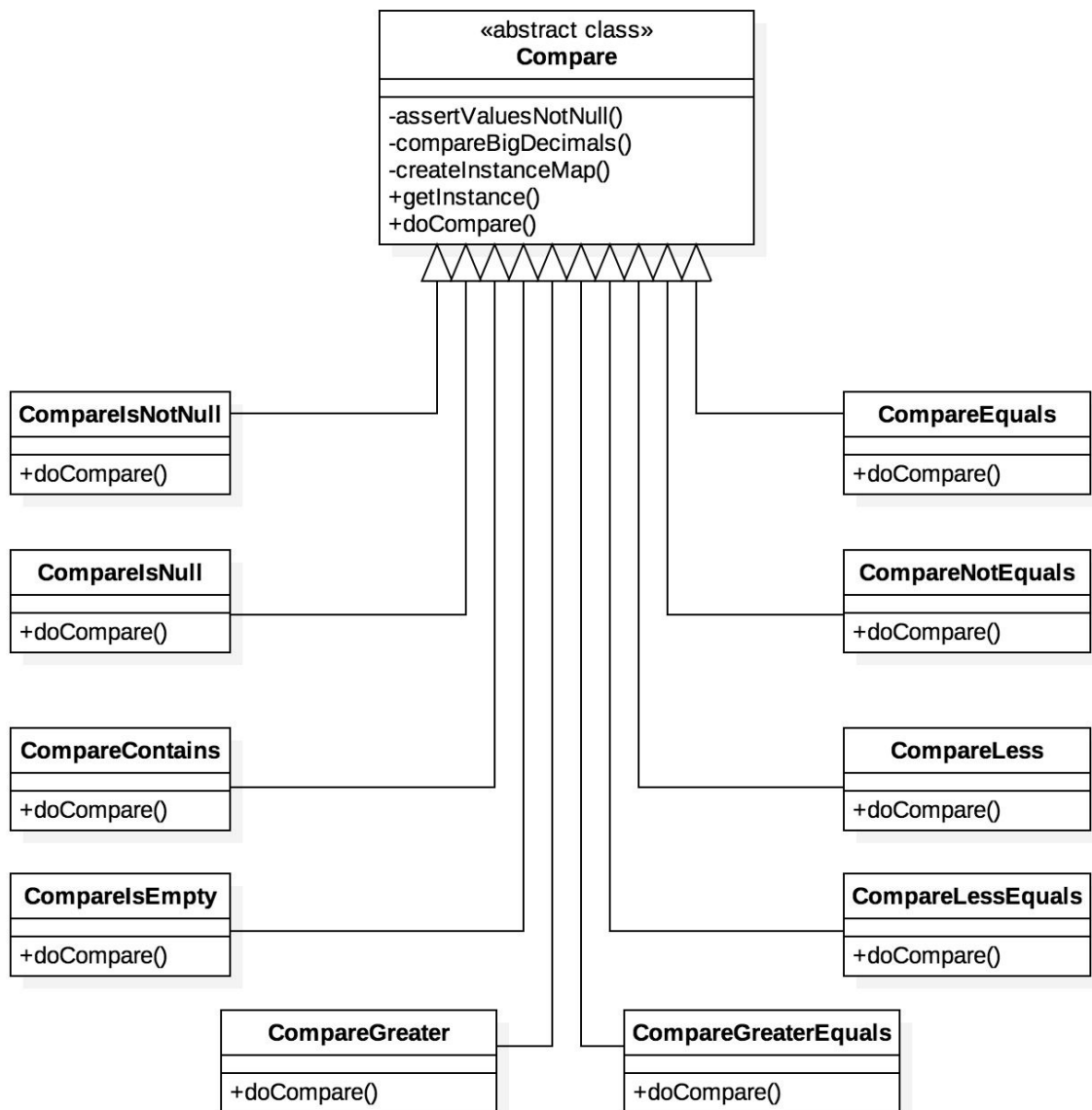
An example of a simple usage is shown:

```
// elem1 and elem2 are both instances of BigDecimal or both implements  
// Comparable interface.  
Compare cmp = Compare.getInstance("greater-equals");  
if (cmp.doCompare(elem1, elem2)) {  
    // one >= two  
}
```

The dependencies of Compare.java are basic utility classes, as shown below:



Compare.java has the following class hierarchy:



## 2.2. Methods

The methods defined in Compare class are the following:

Method name	assertValuesNotNull
Signature	<code>private static void assertValuesNotNull(Object lValue, Object rValue)</code>
Description	Check that the lValue and rValue are both not null or raise IllegalArgumentException otherwise.

Method name	compareBigDecimals
Signature	<code>private static int compareBigDecimals(BigDecimal lBigDecimal, BigDecimal rBigDecimal)</code>
Description	Compare BigDecimals by rounding up the number with more digits after the comma. For example, 3.4 == 3.44 because 3.44 would be rounded to 3.4, while 3.42 < 3.44 because they have the same number of digits.

Method name	createInstanceMap
Signature	<code>private static Map&lt;String, Compare&gt; createInstanceMap()</code>
Description	Creates an unmodifiable map that serves as a lookup table in order to get the correct Compare instance. The key-value pairs are the following: "contains" → CompareContains "equals" → CompareEquals "greater" → CompareGreater "greater-equals" → CompareGreaterEquals "is-empty" → CompareIsEmpty "is-not-null" → CompareIsNotNull "is-null" → CompareIsNull "less" → CompareLess "less-equals" → CompareLessEquals "not-equals" → CompareNotEquals

Method name	getInstance
Signature	<code>public static Compare getInstance(String operator)</code>
Description	Look up the table created with createInstanceMap and returns the value associated with the operator key

Method name	doCompare
Signature	<code>public abstract boolean doCompare(Object lValue, Object rValue, Class&lt;?&gt; type, Locale locale, TimeZone timeZone, String format) throws Exception;</code>
Description	This abstract method will be overridden by every subclasses and will compare the lValue and rValue Objects according to a specific operator.

## 2.3. Inner classes








All the classes defined in `Compare.java` extends the abstract class `Compare` and define the `doCompare(lValue, rValue, ... )` method.

- `CompareNotNull`: returns true if `lValue` is not null
- `CompareIsNull`: returns true if `lValue` is null
- `CompareContains`: returns true if:
  - `lValue` and `rValue` are `String` and `rValue` is a substring of `lValue`
  - `lValue` is a `Collection` and `rValue` is contained in that `Collection`
- `CompareIsEmpty`: returns true if `lValue` is empty
- `CompareGreater`: returns true if `lValue` > `rValue`
- `CompareGreaterEquals`: returns true if `lValue` ≥ `rValue`
- `CompareLessEquals`: returns true if `lValue` ≤ `rValue`
- `CompareLess`: returns true if `lValue` < `rValue`
- `CompareNotEquals`: returns true if `lValue` ≠ `rValue`
- `CompareEquals`: returns true if `lValue` = `rValue`





## 3. Issues

### 3.1. 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.
- 2-  There are no one-character variables
- 3-  The main class is named 'Compare', which is not a noun but a verb. Also, all the inner classes are named after a verb; 'CompareContains', 'CompareEquals', 'CompareGreater', 'CompareGreaterEquals', 'CompareIsEmpty', 'CompareIsNotNull', 'CompareIsNull', 'CompareLess', 'CompareLessEquals', 'CompareNotEquals'.  
  
In order to comply with the naming conventions, they should have used 'Comparator' instead of 'Compare'.
- 4-  There are no interface definitions
- 5-  All method names comply with the naming conventions.
- 6-  All variable names comply with the naming conventions.
- 7-  Constants comply with the naming conventions.

### 3.2. Indentation

- 8-  Four spaces are consistently used for indentation.
- 9-  No tabs.

### 3.3. Braces

10-  
✓

"Kernighan and Ritchie" bracing style is used consistently in the whole class.

11-  
✓

All if, for, while, do-while and try-catch block statements are surrounded with curly braces.

### 3.4. File organization

12-  
✓

All methods are preceded with a blank line. All class definitions have a blank line either before and after the definition statement.

13-  
✗

The following lines of code exceed the 80 character limit and should be shortened:

Line	Statement	Solution
52	<pre>private static int compareBigDecimals(BigDecimal lBigDecimal, BigDecimal rBigDecimal)</pre>	Break the method signature after the comma
57	<pre>lBigDecimal = lBigDecimal.setScale(rBigDecimal.scale() , RoundingMode.UP);</pre>	Break the method signature after the comma
102	<pre>public abstract boolean doCompare(Object lValue, Object rValue, Class&lt;?&gt; type, Locale locale, TimeZone timeZone, String format) throws Exception;</pre>	Break the method signature after each comma
107	<pre>public boolean doCompare(Object lValue, Object rValue, Class&lt;?&gt; type, Locale locale, TimeZone timeZone, String format) throws Exception {</pre>	Break the method signature after each comma
119	<pre>throw new IllegalArgumentException("Cannot compare: l-value is not a collection");</pre>	Declare the error message as a constant.
126	<pre>public boolean doCompare(Object lValue, Object rValue, Class&lt;?&gt; type, Locale locale, TimeZone timeZone, String format) throws Exception {</pre>	Break the method signature after each comma
127	<pre>Object convertedLvalue =</pre>	Break the method

	<code>MiniLangUtil.convertType(lValue, type, locale, timeZone, format);</code>	parameters after a comma
128	<code>Object convertedRvalue = MiniLangUtil.convertType(rValue, type, locale, timeZone, format);</code>	Break the method parameters after a comma
153	<code>public boolean doCompare(Object lValue, Object rValue, Class&lt;?&gt; type, Locale locale, TimeZone timeZone, String format) throws Exception {</code>	Break the method signature after each comma
154	<code>Object convertedLvalue = MiniLangUtil.convertType(lValue, type, locale, timeZone, format);</code>	Break the method parameters after a comma
155	<code>Object convertedRvalue = MiniLangUtil.convertType(rValue, type, locale, timeZone, format);</code>	Break the method parameters after a comma
168	<code>throw new IllegalArgumentException("Cannot compare: l-value is not a comparable type");</code>	Declare the error message as a constant.
175	<code>public boolean doCompare(Object lValue, Object rValue, Class&lt;?&gt; type, Locale locale, TimeZone timeZone, String format) throws Exception {</code>	v the method signature after each comma
176	<code>Object convertedLvalue = MiniLangUtil.convertType(lValue, type, locale, timeZone, format);</code>	Break the method parameters after a comma
177	<code>Object convertedRvalue = MiniLangUtil.convertType(rValue, type, locale, timeZone, format);</code>	Break the method parameters after a comma
190	<code>throw new IllegalArgumentException("Cannot compare: l-value is not a comparable type");</code>	Declare the error message as a constant.
197	<code>public boolean doCompare(Object lValue, Object rValue, Class&lt;?&gt; type, Locale locale, TimeZone timeZone, String format) throws Exception {</code>	Break the method signature after each comma
198	<code>Object convertedLvalue = MiniLangUtil.convertType(lValue, type, locale, timeZone, format);</code>	Break the method parameters after a comma

206	<code>public boolean doCompare(Object lValue, Object rValue, Class&lt;?&gt; type, Locale locale, TimeZone timeZone, String format) throws Exception {</code>	Break the method signature after each comma
207	<code>Object convertedLvalue = MiniLangUtil.convertType(lValue, type, locale, timeZone, format);</code>	Break the method parameters after a comma
215	<code>public boolean doCompare(Object lValue, Object rValue, Class&lt;?&gt; type, Locale locale, TimeZone timeZone, String format) throws Exception {</code>	Break the method signature after each comma
216	<code>Object convertedLvalue = MiniLangUtil.convertType(lValue, type, locale, timeZone, format);</code>	Break the method parameters after a comma
224	<code>public boolean doCompare(Object lValue, Object rValue, Class&lt;?&gt; type, Locale locale, TimeZone timeZone, String format) throws Exception {</code>	Break the method signature after each comma
225	<code>Object convertedLvalue = MiniLangUtil.convertType(lValue, type, locale, timeZone, format);</code>	Break the method parameters after a comma
226	<code>Object convertedRvalue = MiniLangUtil.convertType(rValue, type, locale, timeZone, format);</code>	Break the method parameters after a comma
239	<code>throw new IllegalArgumentException("Cannot compare: l-value is not a comparable type");</code>	Declare the error message as a constant.
246	<code>public boolean doCompare(Object lValue, Object rValue, Class&lt;?&gt; type, Locale locale, TimeZone timeZone, String format) throws Exception {</code>	Break the method signature after each comma
247	<code>Object convertedLvalue = MiniLangUtil.convertType(lValue, type, locale, timeZone, format);</code>	Break the method parameters after a comma
248	<code>Object convertedRvalue = MiniLangUtil.convertType(rValue, type, locale, timeZone, format);</code>	Break the method parameters after a comma
261	<code>throw new IllegalArgumentException("Cannot</code>	Declare the error message as a

	<code>compare: l-value is not a comparable type");</code>	constant.
268	<code>public boolean doCompare(Object lValue, Object rValue, Class&lt;?&gt; type, Locale locale, TimeZone timeZone, String format) throws Exception {</code>	Break the method signature after each comma
269	<code>Object convertedLvalue = MiniLangUtil.convertType(lValue, type, locale, timeZone, format);</code>	Break the method parameters after a comma
270	<code>Object convertedRvalue = MiniLangUtil.convertType(rValue, type, locale, timeZone, format);</code>	Break the method parameters after a comma

It is also worth to notice that many comments do not comply with the 80 character limit. In particular, the comments that do not fulfill this standard are in lines 37, 53, 54 and 108.

14-  
X

The following lines of code exceed 120 character line length, so they but be wrapped somehow.

Line	Statement	Solution
102	<code>public abstract boolean doCompare(Object lValue, Object rValue, Class&lt;?&gt; type, Locale locale, TimeZone timeZone, String format) throws Exception;</code>	Break the method signature after each comma
107	<code>public boolean doCompare(Object lValue, Object rValue, Class&lt;?&gt; type, Locale locale, TimeZone timeZone, String format) throws Exception {</code>	Break the method signature after each comma
126	<code>public boolean doCompare(Object lValue, Object rValue, Class&lt;?&gt; type, Locale locale, TimeZone timeZone, String format) throws Exception {</code>	Break the method signature after each comma
153	<code>public boolean doCompare(Object lValue, Object rValue, Class&lt;?&gt; type, Locale locale, TimeZone timeZone, String format) throws Exception {</code>	Break the method signature after each comma
175	<code>public boolean doCompare(Object lValue, Object rValue, Class&lt;?&gt; type, Locale locale, TimeZone timeZone, String format) throws Exception {</code>	Break the method signature after each comma

197	<code>public boolean doCompare(Object lValue, Object rValue, Class&lt;?&gt; type, Locale locale, TimeZone timeZone, String format) throws Exception {</code>	Break the method signature after each comma
206	<code>public boolean doCompare(Object lValue, Object rValue, Class&lt;?&gt; type, Locale locale, TimeZone timeZone, String format) throws Exception {</code>	Break the method signature after each comma
215	<code>public boolean doCompare(Object lValue, Object rValue, Class&lt;?&gt; type, Locale locale, TimeZone timeZone, String format) throws Exception {</code>	Break the method signature after each comma
224	<code>public boolean doCompare(Object lValue, Object rValue, Class&lt;?&gt; type, Locale locale, TimeZone timeZone, String format) throws Exception {</code>	Break the method signature after each comma
246	<code>public boolean doCompare(Object lValue, Object rValue, Class&lt;?&gt; type, Locale locale, TimeZone timeZone, String format) throws Exception {</code>	Break the method signature after each comma
268	<code>public boolean doCompare(Object lValue, Object rValue, Class&lt;?&gt; type, Locale locale, TimeZone timeZone, String format) throws Exception {</code>	Break the method signature after each comma

### 3.5. Wrapping lines

- 15-  
✓ No line breaks after a comma. All lines break are after a logical && operator.
- 16-  
✓ All line breaks are done correctly.
- 17-  
✓ All statements are correctly aligned.

### 3.6. Comments

- 18-  
✓ All public methods and classes are commented in Javadoc. The code itself contains only few comments at lines 53, 54 and 108, but it is mostly self-explanatory and more comments are not strictly required. The class does not contain unnecessary

comments.

19-



No commented out code.

### 3.7. Java Source Files

20-



Only one public class (Compare).

21-



The public class is the first class defined in the document.

22-



No interface used.

23-



Only public classes are documented. The following private classes are not documented:

- CompareContains (line 104)
- CompareEquals (line 123)
- CompareGreater (line 150)
- CompareGreaterEquals (line 172)
- CompareIsEmpty (line 194)
- CompareIsNotNull (line 203)
- CompareIsNull (line 212)
- CompareLess (line 221)
- CompareLessEquals (line 243)
- CompareNotEquals (line 265)

### 3.8. Package and Import Statements

24-



Package statement is the first non commented line in the document.

### 3.9. Class and Interface Declarations

25-



The order of class, method and variable declarations comply to the Java convention.

26-



Methods are grouped by functionality.

27-

All the private classes instantiated inside Compare class share quite the same structure and thus leading to code duplication.



The following example may clarify the issue:

```
@Override
public boolean doCompare(<params>){
    Object convertedLvalue = MiniLangUtil.convertType(<params>);
    Object convertedRvalue = MiniLangUtil.convertType(<params>);
    assertValuesNotNull(convertedLvalue, convertedRvalue);
    if (convertedLvalue instanceof BigDecimal &&
        convertedRvalue instanceof BigDecimal) {
        BigDecimal lBigDecimal = (BigDecimal) convertedLvalue;
        BigDecimal rBigDecimal = (BigDecimal) convertedRvalue;
        return compareBigDecimals(lBigDecimal, rBigDecimal) <OP> 0;
    }
    if (convertedLvalue instanceof Comparable &&
        convertedRvalue instanceof Comparable) {
        Comparable<Object> comparable = UtilGenerics.cast(<param>);
        return comparable.compareTo(convertedRvalue) <OP> 0;
    }
    throw new IllegalArgumentException(<same error message>);
}
```

The only difference in the structure is given by the comparison operator marked with **<OP>** in the code above. The other lines are exactly identical.

This duplicated code can be overcome by defining an Operator class with a method `compare()` that will be called rather than 'hardcoding' the **<OP>** in every class.

This problem occurs with the following methods:

- `CompareEquals.doCompare(...)` (lines 126-147)
- `CompareGreater.doCompare(...)` (lines 153-169)
- `CompareGreaterEquals.doCompare(...)` (lines 175-191)
- `CompareLess.doCompare(...)` (lines 224-240)
- `CompareLessEquals.doCompare(...)` (lines 246-261)
- `CompareNotEquals.doCompare(...)` (lines 268-288)

An observation is that even if there is duplicated code, it is quite unlikely to add the support for new comparison operators, hence this implementation should not be a critical point for maintenance in the next future.

## 3.10. Initialization and Declarations

- 28-  
✓ The types used are correct. All variables and class members are declared private, with the exception of method `doCompare` and class method `getInstance` which are correctly declared as public.
- 29-  
✓ All variables are declared in the proper scope.
- 30-  
✓ A new Compare object is created by calling the [default constructor](#).
- 31- All object references are initialized before use. There are checks for null values.





32- Variables are initialized where they are declared.



33- Declarations appear at the beginning of blocks.



### 3.11. Method Calls

34- Parameters are presented in the correct order.



35- The correct methods are being called.



36- The return values are used properly.



### 3.12. Arrays

37- No array is used.



38- There is a HashMap collection which doesn't explicitly check for missing key values. However, the standard behavior of the Map interface is that the get method will just return null. The rest of the code will need to take care of this aspect.



```
public static Compare getInstance(String operator) {  
    Assert.notNull("operator", operator);  
    return INSTANCE_MAP.get(operator);  
}
```

39- No array is used.






### 3.13. Object Comparison



40- The == operator is only ever used to compare an Object with null, or to compare primitive types (like int).



### 3.14. Output Format

- 41-  The only output displayed by this class is at lines 45, 48, 119, 168, 190, 239, 261. There are no spelling or grammatical errors.
- 42-  All error messages are comprehensive (e.g. "Cannot compare: l-value is not a comparable type").
- 43-  The output is formatted correctly.





### 3.15. Computation, Comparisons and Assignments



- 44-  The implementation avoids brutish programming.
- 45-  The order of evaluation is consistent, and the parenthesization is correct. Line 117 avoids using a temporary variable at the expense of using unneeded parentheses:

```
return ((String) lValue).contains((String) rValue);
```


Could be rewritten as:

```
String strValue = (String) lValue;  
return strValue.contains((String) rValue);
```

- 46-  There are no significant issues with operator precedence.
- 47-  No division is performed.
- 48-  The class does not present issues related to integer arithmetic.
- 49-  The comparison operators are correct. We noticed that, for the `CompareEquals` and `CompareNotEquals` specializations, special care has been put to correctly handle the case when the left/right values are null. This is handled with assertion in other specializations.

- 50-  There are no try-catch expressions.
- 51-  The code is free of implicit type conversions.

## 3.16. Exceptions

- 52-  There are numerous methods with a “throws Exception” annotation. This annotation, by not specifying a specialized exception type, can become an issue since it does not give enough information to the code that calls it. For example, a method might throw a `FileNotFoundException` but, by having specified “throws Exception”, callers won’t be warned when they fail to handle that specific exception.

This issue is present in the following lines:

Line	Statement
102	<pre>public abstract boolean doCompare(Object lValue, Object rValue, Class&lt;?&gt; type, Locale locale, TimeZone timeZone, String format) throws Exception;</pre>
107	<pre>public boolean doCompare(Object lValue, Object rValue, Class&lt;?&gt; type, Locale locale, TimeZone timeZone, String format) throws Exception {</pre>
126	<pre>public boolean doCompare(Object lValue, Object rValue, Class&lt;?&gt; type, Locale locale, TimeZone timeZone, String format) throws Exception {</pre>
153	<pre>public boolean doCompare(Object lValue, Object rValue, Class&lt;?&gt; type, Locale locale, TimeZone timeZone, String format) throws Exception {</pre>
175	<pre>public boolean doCompare(Object lValue, Object rValue, Class&lt;?&gt; type, Locale locale, TimeZone timeZone, String format) throws Exception {</pre>
197	<pre>public boolean doCompare(Object lValue, Object rValue, Class&lt;?&gt; type, Locale locale, TimeZone timeZone, String format) throws Exception {</pre>
206	<pre>public boolean doCompare(Object lValue, Object rValue, Class&lt;?&gt; type, Locale locale, TimeZone timeZone, String format) throws Exception {</pre>
215	<pre>public boolean doCompare(Object lValue, Object rValue, Class&lt;?&gt; type, Locale locale, TimeZone timeZone, String format) throws Exception {</pre>
224	<pre>public boolean doCompare(Object lValue, Object rValue, Class&lt;?&gt; type, Locale locale, TimeZone timeZone, String format) throws Exception {</pre>

246	<code>public boolean doCompare(Object lValue, Object rValue, Class&lt;?&gt; type, Locale locale, TimeZone timeZone, String format) throws Exception {</code>
268	<code>public boolean doCompare(Object lValue, Object rValue, Class&lt;?&gt; type, Locale locale, TimeZone timeZone, String format) throws Exception {</code>

53- There are no try-catch expressions.



### 3.17. Flow of Control

54- The code does not contain switch statements.



55- The code does not contain switch statements.



56- The code does not contain loops.



### 3.18. Files

57~60 The code does not employ files.



## 4. Observations

During the code inspection we have identified some problems with the assigned piece of code.

- Why do they declare an `assertNotEquals` function inside the class? They should have used the `Assert.NotNull( ... )` method.
- `CompareIsNotNull`, `CompareIsEmpty` and `CompareIsNull` performs a check only on the `IValue`. In fact these are unary operators, and shouldn't be implemented in a binary method interface.
- `CompareContains` performs two logically different operations: substring search or Collection membership and this can lead to confusion.
- This code makes heavy use of the `instanceof` operation in order to manage different type comparison. Using generic methods may help making the code cleaner by removing explicit casts and `instanceof` operations.
- We haven't found unit tests for the `Compare` class. Since this class is quite self-contained, it shouldn't be hard to write unit tests for it.

The overall observation is that the class is self-contained, well written and easy to read.

## 5. Appendix

### 5.1. Used tools

- Google Docs (<http://www.docs.google.com>) for this document
- StarUML (<http://staruml.io/>) for the diagrams.

### 5.2. Hours of work

The following are the total number of hours spent by each member of the group:

- Andrea Battistello: 8h
- William di Luigi: 5h

### 5.3. Revision history

<i>Version</i>	<i>Date</i>	<i>Revision description</i>	<i>Revision notes</i>
1.0	04-02-2016	First release	-