Software Engineering 2: PowerEnJoy

Code Inspection Document

Nardo Loris, Osio Alberto

Politecnico di Milano

1 Description of the code

1.1 Assigned classes

The classes assigned to our group are:

- `HtmlTreeRenderer` located in the package `org.apache.ofbiz.widget.renderer.html` of Apache OFBiz.

1.2 Functional role of classes

The class to review is part of the OFBiz project.

> 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)

The role of the package is to render widgets referenced in XML files representing the User Interface of the system into HTML. In particular `HtmlTreeRenderer` renders a widget named tree into a classical expandeable tree structure.

2 Issues

2.1 Naming conventions

2.1.1 Condition 7

Constants are declared using all uppercase with words separated by an underscore. Examples: **MIN_WIDTH**; **MAX_HEIGHT**.
The following lines are not compliant:

- 52: module is static final so it is a constant

2.2 Braces

2.2.1 Condition 10

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).
The following lines are not compliant:

- 54: The right curly brace must be on a new line

2.3 File Organization

2.3.1 Condition 12

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).
The following lines are not compliant:

- 183-184: Left two lines instead of one as a separator, this is non consistent with the whole file.

2.3.2 Condition 13

Where practical, line length does not exceed 80 characters.
The following lines exceed 80 characters:

- 18
- 61
- 88
- 101
- 119
- 146
- 223
- 240
- 312
- 335

- 49
- 62
- 93
- 102
- 120
- 150
- 224
- 268
- 313
- 342

- 56
- 72
- 94
- 107
- 129
- 158
- 227
- 308
- 314
- 343

- 58
- 81
- 95
- 112
- 134
- 185
- 228
- 309
- 319

This could have been avoided breaking lines on method declaration, using less nested code and shorter variable names.

2.3.3 Condition 14

When line length must exceed 80 characters, it does **not** exceed 120 characters.
The following lines exceed 120 characters:

- 56
- 150
- 185
- 268
- 134
- 158
- 227
- 342

2.4 Wrapping Lines

2.4.1 Condition 17

A new statement is aligned with the beginning of the expression at the same level as the previous line.
The following lines are not compliant:

- 227-228: Line 228 has less indentation than expected.
- 342-343: Line 343 has less indentation than excepted.

2.5 Comments

2.5.1 Condition 18

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

Apart from copyright and a minimal class description there is no helpful comment in the whole file, nor in the interface it implements.
The following comments are useless:

Line	Comment	Reason
159, 186, 269	open tag	It states something obvious
257	the text	It states something wrong, probabily this is due to a copy paste from renderLabel
178, 264	close tag	It states something obvious
85	FIXME	Since the solution is simple, instead of writing the comment, one should have written the fix

2.5.2 Condition 19

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.
The following lines have been commented out, but no date is present:

- 92
- 106

2.6 Java Source Files

2.6.1 Condition 22

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

No javadoc is present in the implemented interface.

2.6.2 Condition 23

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

No javadoc is present in the class assigned.

2.7 Class and Interface Declarations

2.7.1 Condition 25

The class or interface declarations shall be in a standard order.
The following lines are not compliant:

- 52: public static variable must be before line 51

2.7.2 Condition 27

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

The assigned class contains lot of duplicated code, in particular the following pattern

```
if (UtilValidate.isNotEmpty({attributeValue})) {
  writer.append(" {attributeName}=\"");
  writer.append({attributeValue});
  writer.append("\"");
}
```

is repeated several times, in the following methods:

- `renderLabel`: 2 times
- `renderLink`: 5 times immediate + 1 time where a long computation is embedded
- `renderImage`: 5 times immediate + 1 time where a long computation is embedded

The following methods are overly complex and long, they should have used private or utility methods

- `renderNodeBegin`
- `renderLink`
- `renderImage`

2.8 Initialization and Declarations

2.8.1 Condition 28

Check that variables and class members are of the correct type. Check that they have the right visibility (public / private / protected).
The following lines are not compliant:

- 51 : The declared field must be private, since a getter is defined and no other class references this field directly.

2.8.2 Condition 29

Check that variables are declared in the proper scope.
The following lines are not compliant:

- 57: The declared variable is used only inside some blocks, so it is pointless to declare it outside those blocks.
- 305: Since the variable is never modified and only accessed, the constant value should have been propagated and the variable removed.
- 306: Since the variable is never modified and only accessed, the constant value should have been propagated and the variable removed.

- 307: Since the variable is never modified and only accessed, the constant value should have been propagated and the variable removed.

2.8.3 Condition 33

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.
The following lines are not compliant:

- 68
- 69
- 70
- 76
- 86
- 87
- 94
- 95
- 107
- 112
- 113
- 129
- 161
- 167
- 188
- 194
- 200
- 206
- 212
- 218
- 221
- 222
- 223
- 224
- 258
- 271
- 277
- 283
- 289
- 295
- 301
- 304
- 305
- 306
- 307
- 308

2.9 Other found problems and considerations

2.9.1 Legacy Java code

In several places the java standard used seems to be pre Java 5, and this makes the code more verbose than what should be. Since Java 5 there is a concept called autoboxing and unboxing, which moves to the compiler some typical conversion from object to primitive values and vice-versa. The source code instead uses explicit boxing and unboxing in the following lines:

- 105: It is used `Boolean.TRUE` instead of `true`
- 128: It is used `Boolean.FALSE` instead of `false`
- 136,152: It is used `{variable}.booleanValue()` instead of `{variable}`.

Since Java 6 `@Override` annotations could also be put on implemented interface methods, and this is also a good practice. None of the implemented methods have an `@Override` annotation.

2.9.2 Desugaring of String concatenation operator

The desugaring of String concatenation operator is made whenever one uses the `StringBuilder` instead of the `+` operator with no particular performance reasons. The Java compiler already translates the `+` operator to a `StringBuilder` pattern, for example the above two example are equals also on the bytecode produced by the compiler:

```
String string = "b";
String result = "a " + string + " c";
```

This code use the standard concatenation operator.

```
String string = "b";
String result = (new StringBuilder().append(" a").append(string).append("
c")).toString();
```

This code use the `StringBuilder` class.

The following lines uses such desugaring for no reason:

- 94
- 112
- 235: Even if the `StringBuilder` is passed to a method, there is an alternative methods that gets the same result as a `String` object.
- 321: Even if the `StringBuilder` is passed to a method, there is an alternative methods that gets the same result as a `String` object.

2.9.3 Mixed responsibilities

Mixed responsibilities happens when a task logically belonging to a unit is made by another unit. This can lead to errors, in fact in the class assigned to us the mixed resposibility for the encoding of the text lead to some errors in particular:

- When the code need to get an attribute from a `ModelNode` subclass it calls a method belonging to the subclass which is resposable to encode the text according to the renderer used and then returns this text, this task should have been done by the renderer itself and in fact not all attribute values are properly encoded for example the `getStyle`, `getId`, `getName`, `getTitle` and others do not encode the text.
- When building the target destination of a link, the `externalLoginKey` is not encoded.

This can lead to possible vectors to XSS or to unexpected behaviours.

3 Appendix

3.1 Effort spent

- Nardo Loris: 14 hours of work
- Osio Alberto: 10 hours of work