POLITECNICO
MILANO 1863

Politecnico di Milano
A.A. 2016/2017
Software Engineering 2

Code Inspection v. 1.0

Matteo Bresich (mat. 774366)

February 5, 2017

Table of Content

1 Classes Assigned

The class assigned is part of Apache OFBiz version 16.11.01.
The name of the class assigned is *OfbizAmountTransform.java* and it's located in the following path: /framework/webapp/src/main/java/org/apache/ofbiz/webapp/ftl/

2 Functional role of assigned set of classes

The OfbizAmountTransform is a class that implements TemplateTransformModel interface and it is specialized on filtering output. The methods analysed are:

- String getArg(Map args, String key) [private static method]
- Double getAmount(Map args, String key) [private static method]
- Writer getWriter(final Writer out, Map args) [factory method]

3 List of issues found by applying the checklist

The issues checklist can be found in the document "Code Inspection Assignment Task Description.pdf".

3.1 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.**
    - Class names: no issues.
    - Interface names: no issues.
    - Method names: no issues.
    - Class variables: no issues.
    - Method variables: no issues, meaningless variable names like "Object o" at line 50, 68 are temporary throwaway variables.
    - Constants: no issues.
2. **If one-character variables are used, they are used only for temporary "throwaway" variables, such as those used in for loops.**
    - No issues.
3. **Class names are nouns, in mixed case, with the first letter of each word in capitalized.**
    - No issues.
4. **Interface names should be capitalized like classes.**
    - No issues.
5. **Method names should be verbs, with the first letter of each addition word capitalized.**
    - No issues.
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.**
    - No issues.
7. **Constants are declared using all uppercase with words separated by an underscore.**
    - issue at line 45 ("module" should be "MODULE").

3.2 Indention

8. **Three or four spaces are used for indentation and done so consistently.**
    - Two spaces instead of four at line 61.
9. **No tabs are used to indent.**
    - No issues, only spaces used.

3.3 Braces

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).**
    - No issues.
11. **All if, while, do-while, try-catch, and for statements that have only one statement to execute are surrounded by curly braces.**
    - Issues at lines 52, 69 and 113.

3.4 File Organization

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).**
    - Issues at lines 75, 94 and 98.
13. **Where practical, line length does not exceed 80 characters.**
    - Issues at lines 18, 52, 69, 95, 113, 120, 128, 129 and 131.
14. **When line length must exceed 80 characters, it does NOT exceed 120 characters.**
    - No issues.

3.5 Wrapping Lines

15. **Line break occurs after a comma or an operator.**
    - No issues.
16. **Higher-level breaks are used.**
    - No issues.
17. **A new statement is aligned with the beginning of the expression at the same level as the previous line.**
    - No issues.

3.6 Comments

18. **Comments are used to adequately explain what the class, interface, methods, and blocks of code are doing.**
    - No issues.
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.**
    - No issues.

3.7 Java Source Files

20. **Each Java source file contains a single public class or interface.**
    - No issues.
21. **The public class is the first class or interface in the file.**
    - No issues.
22. **Check that the external program interfaces are implemented consistently with what is described in the javadoc.**
    - No issues, TemplateTransformModel interface is implemented consistently.
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 issues.

3.8 Package and Import Statements

24. **If any package statements are needed, they should be the first non-comment statements. Import statements follow.**
    - No issues.

3.9 Class and Interface Declarations

25. **The class or interface declarations shall be in order**
    - No issues, they are in the correct order.
26. **Methods are grouped by functionality rather than by scope or accessibility.**
    - No issues.
27. **Check that the code is free of duplicates, long methods, big classes, breaking encapsulation, as well as if coupling and cohesion are adequate.**
    - No issues.

3.10 Initialization and Declarations

28. **Check that variables and class members are of the correct type. Check that they have the right visibility (public/private/protected).**
    - No issues.
29. **Check that variables are declared in the proper scope.**
    - No issues.
30. **Check that constructors are called when a new object is desired.**
    - Some objects are not initialized by the constructor at lines: 45, 46, 49, 50, 68, 77, 81, 85, 95, 96, 97, 114, 117 and 118.
31. **Check that all object references are initialized before use.**
    - No issues.

32. **Variables are initialized where they are declared, unless dependent upon a computation.**
    - No issues.
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.**
    - Issue at line 114.

3.11 Method Calls

34. **Check that parameters are presented in the correct order.**
    - No issues.
35. **Check that the correct method is being called, or should it be a different method with a similar name.**
    - No issues.
36. **Check that method returned values are used properly.**
    - No issues.

3.12 Arrays

37. **Check that there are no off-by-one errors in array indexing (that is, all required array elements are correctly accessed through the index).**
    - No issues.
38. **Check that all array (or other collection) indexes have been prevented from going out-of-bounds.**
    - No issues.
39. **Check that constructors are called when a new array item is desired.**
    - No issues.

3.13 Object Comparison

40. **Check that all objects (including Strings) are compared with equals and not with ==.**
    - No issues. (At line 72 the == is not a comparison and it is used to check that the variable o is null).

3.14 Output Format

41. **Check that displayed output is free of spelling and grammatical errors.**
    - No issues.
42. **Check that error messages are comprehensive and provide guidance as to how to correct the problem.**
    - No issues.
43. **Check that the output is formatted correctly in terms of line stepping and spacing.**
    - No issues.

3.15 Computation, Comparisons and Assignments

44. **Check that the implementation avoids ”brutish programming”**
    - No issues.
45. **Check order of computation/evaluation, operator precedence and parenthesizing.**
    - No issues.
46. **Check the liberal use of parenthesis is used to avoid operator precedence problems.**
    - No issues.
47. **Check that all denominators of a division are prevented from being zero.**
    - No issues.
48. **Check that integer arithmetic, especially division, are used appropriately to avoid causing unexpected truncation/rounding.**
    - No issues.
49. **Check that the comparison and Boolean operators are correct.**
    - No issues.
50. **Check throw-catch expressions, and check that the error condition is actually legitimate.**
    - No issues.
51. **Check that the code is free of any implicit type conversions.**
    - No issues.

3.16 Exceptions

52. **Check that the relevant exceptions are caught.**
    - No issues.
53. **Check that the appropriate action are taken for each catch block.**
    - No issues.

3.17 Flow of Control

54. **In a switch statement, check that all cases are addressed by break or return.**
    - No issues.
55. **Check that all switch statements have a default branch.**
    - No issues.
56. **Check that all loops are correctly formed, with the appropriate initialization, increment and termination expressions.**
    - No issues.

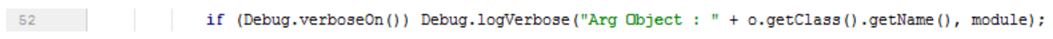
3.18 Files

57. **Check that all files are properly declared and opened.**
    - No issues.
58. **Check that all files are closed properly, even in the case of an error.**
    - No issues.
59. **Check that EOF conditions are detected and handled correctly.**
    - No issues.
60. **Check that all file exceptions are caught and dealt with accordingly.**
    - No issues.

4 Other Issues

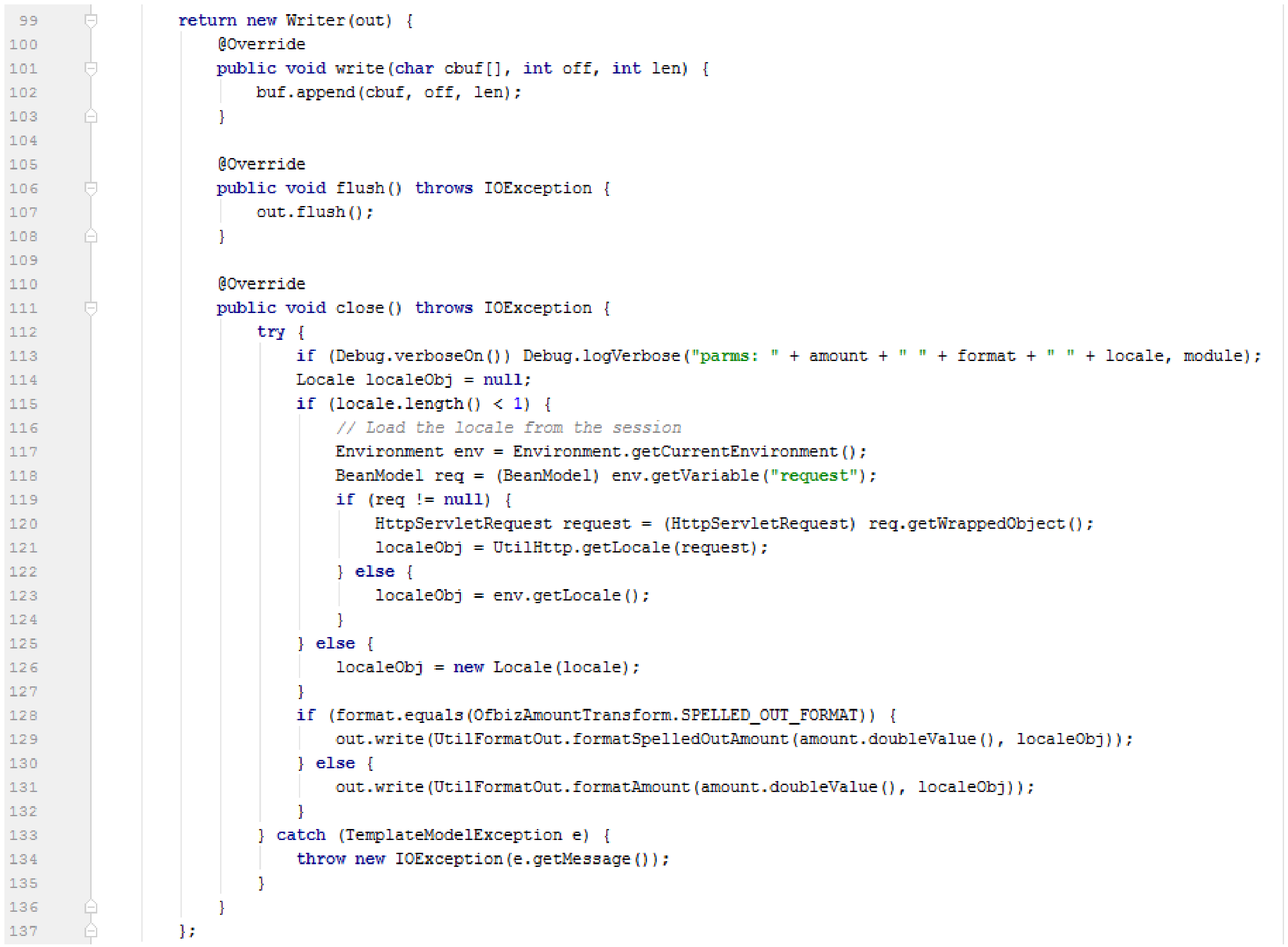
In the class additional problems were found:

- line 49: the space is doubled
- lines 52, 69, 113: two statements per line instead of one



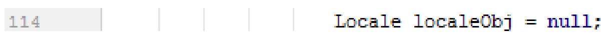
```
52              if (Debug.verboseOn()) Debug.logVerbose("Arg Object : " + o.getClass().getName(), module);
```

- line 99: the anonymous class should be better if it was a named class (too complex for an anonymous class).



```
99          return new Writer(out) {
100             @Override
101             public void write(char cbuf[], int off, int len) {
102                 buf.append(cbuf, off, len);
103             }
104
105             @Override
106             public void flush() throws IOException {
107                 out.flush();
108             }
109
110             @Override
111             public void close() throws IOException {
112                 try {
113                     if (Debug.verboseOn()) Debug.logVerbose("parms: " + amount + " " + format + " " + locale, module);
114                     Locale localeObj = null;
115                     if (locale.length() < 1) {
116                         // Load the locale from the session
117                         Environment env = Environment.getCurrentEnvironment();
118                         BeanModel req = (BeanModel) env.getVariable("request");
119                         if (req != null) {
120                             HttpServletRequest request = (HttpServletRequest) req.getWrappedObject();
121                             localeObj = UtilHttp.getLocale(request);
122                         } else {
123                             localeObj = env.getLocale();
124                         }
125                     } else {
126                         localeObj = new Locale(locale);
127                     }
128                     if (format.equals(OfbizAmountTransform.SPELLED_OUT_FORMAT)) {
129                         out.write(UtilFormatOut.formatSpelledOutAmount(amount.doubleValue(), localeObj));
130                     } else {
131                         out.write(UtilFormatOut.formatAmount(amount.doubleValue(), localeObj));
132                     }
133                 } catch (TemplateModelException e) {
134                     throw new IOException(e.getMessage());
135                 }
136             }
137         };
```

- line 114: useless assignment



```
114                     Locale localeObj = null;
```

5 Effort Spent

5.1 Hours of work

The time spent to redact this document:

- Bresich Matteo: 16 hours.

Days	Hours of work
01/02/17	4h
02/01/17	4h
04/01/17	4h
05/01/17	4h

6 References

- TeXstudio v2.11.2 (http://www.texstudio.org/) to produce this document.
- IntelliJ IDEA ULTIMATE 2016.3.2 (https://www.jetbrains.com/idea/) to inspect the assigned class.
- SonarLint v2.7.1.1640 (IntelliJ plug-in): plugin for code analysis.
- Sublime Text 3: to inspect the assigned class.