Code Inspection

Riccardo Cattaneo 873647
Fabio Chiusano 874294

Version: 1.0.0
Release date: 20-01-2017

1 Table of Contents

2 Classes assigned

We have been assigned the PartyContentWrapper class, located at the following path:

- .../apache-ofbiz-6.11.01/applications/party/src/main/java/org/apache/ /ofbiz/party/content/PartyContentWrapper.java

3 Functional role of assigned set of classes

3.1 Introduction

OFBiz (Open For Business) is an Enterprise Resource Planning (ERP) System written in Java and houses a large set of libraries, entities, services and features to run all aspects of a business. It is composed of multiple applications, that share same core components and entities.
The OFBiz documentation can be found at the following link:
http://ofbiz.apache.org/documentation.html

The class we have to analyse is a part of the Party Management Application.
From the name of the class, PartyContentWrapper, we can deduce that is a wrapper for the content of a Party. The first answer we need to provide is about what a Party in OFBiz is. From the documentation, specifically from https://ofbiz.apache.org/apache-ofbiz-project-overview.html, we have learned that:

> A Party can be either a Person, or a group of Parties. A Party Group could be a company, an organization within the company, a supplier, a customer, and so forth. Information that describes Parties or is directly related to Parties is contained in these entities.
>
> One type of related data is Contact Mechanisms such as postal addresses, phone numbers, email addresses, internet URLs. Another is Roles that the Party acts in such as Customer, Supplier, Employee, Manager, Merchandiser, etc. Generally, a single party will interact with different parts of the system in many different roles.
>
> Another type of data that fits into the Party category is information about communication and agreements between Parties. This gets into the area of relationship management and also includes information about issues or trouble tickets that a Party may have. These entities are used along with the Work Effort entities to plan and track the research and resolution of such issues.

Thus, we can summarize by saying that a Party:

- can be either a Person or a group of Parties;
- contains the following data:
  - Contact Mechanisms such as postal addresses, phone numbers, email addresses, internet URLs;
  - Roles that the Party acts, such as Customer, Supplier, Employee, Manager;

- Information about communication and agreements between Parties; these entities are used along with the Work Effort entities to plan and track the research and resolution of issues such as trouble tickets.

The second answer is about what functionalities usually a Wrapper class provides. We can learn from Wikipedia.org that:

> A wrapper function is a subroutine in a software library or a computer program whose main purpose is to call a second subroutine or a system call with little or no additional computation.

We have searched documentation about this PartyContentWrapper class, but there is not almost any JavaDoc comment in the whole file and there are few comments, therefore it won't be probably easy to understand what the methods do.

3.2 Information from other classes

The comment before the class declaration is:

```
/**
 * WorkEffortContentWrapper; gets work effort content for display
 */
```

This is confusing, since there is no WorkEffortContentWrapper in our PartyContentWrapper class. However, by checking the WorkEffortContentWrapper.java file, we found that the same comment is present in that class too, along with a lot of copied code. Therefore, we can conclude that our class was not written from zero, but starting with a copy and paste of the WorkEffortContent class and then replacing each occurrence of *WorkEffort* with *Party*. This suggests that the functionalities may be the same.

In the WorkEffortContentWrapper.java file there are more comments than in its copied class, therefore it can be useful to see how many methods they have in common, so that we can read the JavaDoc comments in WorkEffortContentWrapper in order to understand them. It turns out that **all** methods in PartyContentWrapper.java have a corresponding method in WorkEffortContentWrapper.java that shares a huge part of code. This is a sign of bad code reuse and software design. However, these problems should have been addressed previously in the development and now we are interested in code issues only. Moreover, the ContentWrapper interface, that is the interface implemented by both PartyContentWrapper and WorkEffortContentWrapper, has only one of those methods.

3.3 Class Functionalities

In the previous analysis we have concluded that PartyContentWrapper is a wrapper for a Party, adding some functionalities to that entity.

Analysing the methods of the class, and therefore its functionalities, we have understood that PartyContentWrapper class has convenient methods to get its content, related to a Party entity, and display it.

There are multiple methods with the same name in the class, and one of them is the implementation of the interface ContentWrapper. Thus, functionalities provided by various methods should be the same, with differences in parameters and type of the returned value.

4 Issues found by applying the checklist

4.1 Naming Conventions

- The "module" variable is constant, since it is *static* and *final*, but it is not declared using all upper case characters:

```
57    public static final String module = PartyContentWrapper.class.getName();
```

- This method seems to be getter, bet the return type is void. We think that it is not a meaningful name:

```
187    public static void getPartyContentAsText(String contentId, String
partyId, GenericValue party, String partyContentTypeId, Locale locale, String
mimeTypeId, Delegator delegator, LocalDispatcher dispatcher, Writer outWriter)
throws GeneralException, IOException {    //…    }
```

  Another method with the same issue can be found at line 191.

4.2 Indentation

- The indentation is not consistent in line 134:

```
133    public static String getPartyContentAsText(GenericValue party, //…
134              Locale locale, String mimeTypeId, Delegator delegator, //…
135        return getPartyContentAsText(party, null, partyContentTypeId, //…
```

  Furthermore, also lines 139, 149, 152, 261-265, 298-301 have the same problem.

4.3 Braces

The brace style used is the Kernighan and Ritchie one and it is consistent in the whole class. Every if, while, do-while, try-catch and for statements that have only one statement to execute are surrounded by curly braces.
Thus, we have not identified any brace problem in the analysed class.

4.4 File organization

- There is almost any comment. We have analysed in section 3.2 the only comment present before the class declaration.
  We think that the lackness or the imprecision of comments is a primary problem, and should be resolved in the next releases of the software.

- There are many line that exceed 120 characters. For example, the line 191 is the longest of the class and is made of 281 characters.

```
191     public static void getPartyContentAsText(String contentId, String
partyId, GenericValue party, String partyContentTypeId, Locale locale, String
mimeTypeId, Delegator delegator, LocalDispatcher dispatcher, Writer outWriter,
boolean cache) throws GeneralException, IOException {
```

  The complete list of lines that exceed 120 characters is the following:
  60, 78, 83, 115, 123, 125, 126, 129, 134, 135, 139, 164, 168, 178, 182, 187, 188, 191, 201, 211, 216, 220, 231, 246, 259, 274, 282;

4.5 Wrapping Lines

All the line break occurs after a comma or an operator, and gih-level breaks are used. Furthermore, all the statements are aligned with the beginning of the expression at the same level as the previous line. Thus, we have not identified any wrapping line problem.

4.6 Comments

- As already stated in section 3.1 and 4.4, there is not almost any comment in the class. Moreover, the few line of comment present are inaccurate, and reveal the presence of cloned code from other classes.
  Instead of adapt pasted comments, developers have deleted almost completely them.

The absence of comments implies that there isn't any section of commented out code.

4.7 Java Source Files

- As already mentioned in section 4.6, the absence of comments includes also the absence of proper JavaDoc documentation.

Anyway, the Java Source File contains only one public class, and the implementation of the interface is done consistently.

4.8 Package and import statements

There are no problems in this section, since the package statements are the first non-comment statements, followed by import statements.

4.9 Class and Interface Declarations

After having underlined again that class documentation is missing, we can see that the proposed order of declarations is followed carefully.
Furthermore, there are some other problems within the class:

- The code of the class is cloned from another class similar to it, instead of using powerfull tools that Java provides, such as hierarchy. Anyway there isn't the evidence of cloned code between components and sections of the same class;
- Methods are sorted by accessibility instead of functionality, and this is explicitly mentioned by a comment. Indeed, Public methods are listed first, followed by public static ones.

- Methods that starts at line 138 and 191 are too long. Hence, it would be better to take advantage of some private methods that would have reduced complexity and increased readability of the code.
- Code present at lines 168, 178 and 182 present the problem of encapsulation, as in the following example:

```
168  outString ==party.getModelEntity().isField(candidateFieldName) ?
party.getString(candidateFieldName): "";
```

- The class itself, with its 317 lines of code, seems to be too big and heavy. The size would have reduced by sharing piece of code with other similar classes, such as WorkContentWrapper, using for example an abstract class.

4.10 Initialization and Declaration

- The following object is not created using a constructor. Furthermore, the class “GenericValue” seams to be not very expressive and meaningful:

```
91  GenericValue partyContent = getFirstPartyContentByType(null, party,
contentTypeId, party.getDelegator());
```

- Variables are not always declared at the beginning of a block:

```
155        try {
156            if (useCache) {
157                String cachedValue = partyContentCache.get(cacheKey);
158                if (cachedValue != null) {
159                    return cachedValue;
160                }
161            }
162
163        Writer outWriter = new StringWriter();
```

  Other declaration problems of this kind are present at line 166, 178, 182 and 209.

- We have found two visibility problems:

```
57 public static final String module= PartyContentWrapper.class.getName();
58 public static final String CACHE_KEY_SEPARATOR= "::";
```

Those are variables only used inside this class, and aren’t useful outside it. Thus, they should be declared as private, instead of public.

4.11 Method Calls

- There are six methods called “*getPartyContentAsText(…)*”, from line 123 to line 257. They have different parameters, but also different return type: four of them have *String* return type, two of them have *void*. We think that this is confusing, hence the returned values are not used properly.
- Even if the parameters are presented in a sufficiently correct order, some methods has a list of parameters that is too large.**************

4.12 Arrays

Arrays, but also other type of collections, are not used excessively. The usage of Lists is correct and there aren’t problems with indexes, and runtime exceptions are managed.

4.13 Object Comparison

All the comparisons in the class use “==” instead of using the method *equals()*.

However, there are only comparison with null pointers. Even if there are other way for checking if an object exists or the pointer is null, the comparison using "*== null*" still effective. Moreover, using the *equals()* method for checking a null pointer would rise a *NullPointerException*. Thus, we haven't considered these comparisons as problems.

4.14 Output Format

The output provided by the class is, in general, correct and comprehensive.

- However, exception error messages are not always explained:

```
302         } catch (GeneralException e) {
303             Debug.logError(e, module);
```

The other messages not explained are present at lines 103, 106 and 109.

4.15 Computations, Comparisons, Assignments

Since the class is about converting and creating a String representation of a Party entity, the logic involved is not too complex. Therefore, there aren't many problems concerning computation. However, some improvements can be done:

- There are multiple return statements. This can be identified as a "Brutish Programming" bad technique:

```
100      try {
101          return getPartyContentTextList(party, contentTypeId, locale,
mimeTypeId, party.getDelegator(), dispatcher);
102      } catch (GeneralException ge) {
103          Debug.logError(ge, module);
104          return null;
105      } catch (IOException ioe) {
106          Debug.logError(ioe, module);
107          return null;
108      } catch (Exception e) {
109          Debug.logError(e, module);
110          return null;
111      }
```

- There is a repeated section of code that should be avoided:

```
176   } catch (GeneralException e) {
177       Debug.logError(e, "Error rendering PartyContent, inserting empty
String", module);
178       String candidateOut =
party.getModelEntity().isField(candidateFieldName) ?
party.getString(candidateFieldName): "";
179        return candidateOut == null? "" : encoder.sanitize(candidateOut);
180   } catch (IOException e) {
181        Debug.logError(e, "Error rendering PartyContent, inserting empty
String", module);
182        String candidateOut =
party.getModelEntity().isField(candidateFieldName) ?
party.getString(candidateFieldName): "";
183        return candidateOut == null? "" : encoder.sanitize(candidateOut);
184    }
```

4.16 Exceptions

- As shown in the previous example of cloned code, from line 176 to line 181, the two different exception are managed in the same way. Moreover, the first

catched exception is the most general one. Indeed, GeneralException extends directly the Exception class. Hence, in this section of code a more effective and specialized management of exception should be adopted.
Another problem of the same type can be found at lines 102-108:

```
102     } catch (GeneralException ge) {
103         Debug.logError(ge, module);
104         return null;
105     } catch (IOException ioe) {
106         Debug.logError(ioe, module);
107         return null;
108     } catch (Exception e) {
109         Debug.logError(e, module);
110         return null;
111     }
```

4.17 Flow of Control

All loops that are present are correctly formed and have appropriate initialization, increment and termination expressions, because almost of them are iteration over a List. There aren't switch statement in the class. Thus, there aren't flow problems.

4.18 Files

In the analysed class there aren't files declared and opened. Thus, there aren't files problems.

5 Other problems

We have already discussed the almost complete absence of comments and Javadoc Documentation in the previous chapters of this document.
Moreover, we have found other issues, that we have summarised with the following list:

- As highlighted in section 3.2, PartyContentWrapper class was at first entirely copied from WorkEffortContentWrapper, as suggested from the comment before the class declaration that went unchanged in the copy and paste process. Copying code may be faster but may actually result in some bugs going unnoticed because, generally, modifying involves less concentration that writing from zero. Some issues found in this class are probably present in WorkEffortContentWrapper too;
- "*getPartyContentAsText(…)*" function is overloaded too many times, with a number of parametrs that growed too much. Moreover, as highlighted in section 4.11, the return type of these methods changes, leaving a section of code not totally meaningful and self-descriptive;
- The class implements ContentWrapper Interface, that contains only one method. However, a big piece of code is shared with the other ContentWrapper class. The hierarchy can be improved, for example by creating a ContentWrapper abstract class instead of a simple interface, or maybe by expanding the actual interface with more methods. The result would be a simpler and more efficient piece of code.

- There are a large number of check to find if an object is *null*. We think that this may suggest a bad management of null pointers. Almost every method begins with some checks, with consequent return statement if the pointer was null.
  A better design would avoid not only repetitive checking and simplify code readability, but also it would reduce the risk of a runtime Null Pointer Exception and improve general stability.

6 Hours of Work

We managed to distribute the workload fairly between days and team members in a way that allowed us to finish a week before the deadline and have time for an accurate check in the last days.

The total amount of time required to build this document is about 8 hours for both team members.