

Version: 1.0.0

Release date: 20-01-2017

Riccardo Cattaneo 873647

Fabio Chiusano 874294

Code Inspection

Table of Contents

2 Classes assigned 3

3 Functional role of assigned set of classes 4

4 Issues found by applying the checklist 6

5 Other problems 7

# 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

# Functional role of assigned set of classes

## 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.

## 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.

## 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.

# Issues found by applying the checklist

## 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.

## 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.

## 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.

## File organization

* There is almost any comment. We have analysed in section [3.2](#_Information_from_other) 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;

## 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 .

* Wrapping lines:
* Comments: ahahaha;
* Java source files: no javadoc
* Package and import statements;
* Class and interface declarations: manca class documentation document, ordered by scope (statica t the end), cloned class (WorkEffortContentWrappe), too long methods (138, 191), the class itself is too long; ok except for 168, 178, 182 (encapsulation);
* Initialization and declarations: (91), 166, 178, 182, 209, visibility: 57, 58;
* Method calls: …
* Arrays: ok;
* Object comparison: all the comparisons use ==, but it’s always a comparison with null;
* Output format: error messages are not explained: 103,106,109,303;
* Computations, comparisons and assignments: 102-110: multiple return statements, 177-181: repeated section of same code;
* Exceptions: : 177-181, 102-105-108,: different exceptions managed in the same way;
* Flow of control: ok;
* Files: there aren’t any file;

Some issues:

* Almost total absence of javadoc comments;
* This 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 time, with lots of parameters, the return type changes,…
* (uso scorrretto dell’interfaccia: la classe è tutta copiata da WorkEffort…, ma l’interfaccia comune ha un solo metodo… e il commento sull’implementazione dell’interfaccia è sopra al metodo sbagliato)
* (un sacco di null possibili, un sacco di controlli == null, quindi possibile propagazione dei null con possibili errori a runtime se manca una

# Other problems