Code Inspection

POLITECNICO
MILANO 1863

Version 1.0

Luca Santini 808710
Riccardo Remigio 874939

Index

1. High level class analysis

1.1 Assigned class

The class that we were assigned is ContentServicesComplex.java, it is located in the package org.apache.ofbiz.content.content of the Apache OFBiz Project.

1.2 Functional role of assigned class

ContentServicesComplex.java is a class of Apache OFBiz project.
Apache OFBiz is an open source enterprise resource planning (ERP) system. It provides a suite of enterprise applications that integrate and automate many of the business processes of an enterprise.
All of Apache OFBiz functionality is built on a common framework. The functionality can be divided into the following distinct layers:

Presentation Layer

Apache OFBiz uses the concept of "screens" to represent the Apache OFBiz pages. Each page is, normally, represented as a screen. A page in Apache OFBiz consists of components. A component can be a header, footer, etc. When the page is rendered all the components are combined together as specified in the screen definition. Components might be Java Server Pages ([JSP]s) <deprecated>, FTL pages built around FreeMarker template engine, Forms and Menus Widgets. Widgets are an OFBiz specific technology.

Business Layer

The business, or application layer defines services provided to the user. The services can be of several types: Java methods, SOAP, simple services, workflow, etc. A service engine is responsible for invocation, transactions and security.

Apache OFBiz uses a set of well established, open source technologies and standards such as Java, Java EE, XML and SOAP. Although Apache OFBiz is built around the concepts used by Java EE, many of its concepts are implemented in different ways; either because Apache OFBiz was designed prior to many recent improvements in Java EE or because Apache OFBiz authors didn't agree with those implementations.

Data Layer

The data layer is responsible for database access, storage and providing a common data interface to the Business layer. Data is accessed not in Object Oriented fashion but in a relational way. Each entity (represented as a row in the database) is provided to the business layer as a set of generic values. A generic value is not typed, so fields of an entity are accessed by the column name.

The class we are going to inspect belongs to the content component, here the description of that component.

Content

"The Content entities are used to track data resources and structure them into general content and knowledge. They include many concepts such as: a separation of information and organization allowing a data resource to be used in many content structures; flexible organization of content including free-from association in graphs or more constrained organization in trees, lists, named maps, templates, etc; the specification of meta-data for content and data resources that can be used to implicitly organize and explicitly describe the information. Individual pieces of content can be in various text and binary formats described by standard MIME types and character encodings.
Once general maintenance tools for this information are in place more advanced tools such as keyword based, meta-data based, and intelligent searching or text mining to automatically create additional structure or meta-data can be used to enable enterprise wide document and knowledge management.
The Content entities also include information about Web-based content such as pages and interaction with content including visits to a site (or application) and information about every request sent to the site. This is useful for tracking what users are doing with an application for security, marketing, usability and other reasons."

In particular, ContentServicesComplex.java returns a list of content data resources associated with the content Id passed.

2. Code Inspection

2.1 Checklist

This is the checklist we used to evaluate the quality of the code:

2.1 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.
2. If one-character variables are used, they are used only for temporary "throwaway" variables, such as those used in for loops.
3. Class names are nouns, in mixed case, with the first letter of each word in capitalized. Examples: `class Raster`; `class ImageSprite`;
4. Interface names should be capitalized like classes.
5. Method names should be verbs, with the first letter of each addition word capitalized. Examples: `getBackground()`; `computeTemperature()`.
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. Examples: `_windowHeight`, `timeSeriesData`.
7. Constants are declared using all uppercase with words separated by an underscore. Examples: `MIN_WIDTH`; `MAX_HEIGHT`.

2.2 Indention

8. Three or four spaces are used for indentation and done so consistently.
9. No tabs are used to indent.

2.3 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).

11. All `if`, `while`, `do-while`, `try-catch`, and `for` statements that have only one statement to execute are surrounded by curly braces. Example:
Avoid this:

```
if ( condition )
    doThis();
```

Instead do this:

```
if ( condition )
{
    doThis();
}
```

2.4 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).

13. Where practical, line length does not exceed 80 characters.

14. When line length must exceed 80 characters, it does NOT exceed 120 characters.

2.5 Wrapping Lines

15. Line break occurs after a comma or an operator.

16. Higher-level breaks are used.

17. A new statement is aligned with the beginning of the expression at the same level as the previous line.

2.6 Comments

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

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.

2.7 Java Source Files

20. Each Java source file contains a single public class or interface.

21. The public class is the first class or interface in the file.

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

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

2.8 Package and Import Statements

24. If any package statements are needed, they should be the first non-comment statements. Import statements follow.

2.9 Class and Interface Declarations

25. The class or interface declarations shall be in the following order:

    (a) class/interface documentation comment;

    (b) class or interface statement;

    (c) class/interface implementation comment, if necessary;

    (d) class (static) variables;

        i. first public class variables;

        ii. next protected class variables;

        iii. next package level (no access modifier);

        iv. last private class variables.

    (e) instance variables;

        i. first public instance variables;

        ii. next protected instance variables;

        iii. next package level (no access modifier);

iv. last private instance variables.

(f) constructors;

(g) methods.

26. Methods are grouped by functionality rather than by scope or accessibility.

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

2.10 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).

29. Check that variables are declared in the proper scope.

30. Check that constructors are called when a new object is desired.

31. Check that all object references are initialized before use.

32. Variables are initialized where they are declared, unless dependent upon a computation.

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.

2.11 Method Calls

34. Check that parameters are presented in the correct order.

35. Check that the correct method is being called, or should it be a different method with a similar name.

36. Check that method returned values are used properly.

2.12 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).

38. Check that all array (or other collection) indexes have been prevented from going out-of-bounds.

39. Check that constructors are called when a new array item is desired.

2.13 Object Comparison

40. Check that all objects (including Strings) are compared with `equals` and not with ==.

2.14 Output Format

41. Check that displayed output is free of spelling and grammatical errors.
42. Check that error messages are comprehensive and provide guidance as to how to correct the problem.
43. Check that the output is formatted correctly in terms of line stepping and spacing.

2.15 Computation, Comparisons and Assignments

44. Check that the implementation avoids "brutish programming": (see `http://users.csc.calpoly.edu/~jdalbey/SWE/CodeSmells/bonehead.html`).
45. Check order of computation/evaluation, operator precedence and parenthesizing.
46. Check the liberal use of parenthesis is used to avoid operator precedence problems.
47. Check that all denominators of a division are prevented from being zero.
48. Check that integer arithmetic, especially division, are used appropriately to avoid causing unexpected truncation/rounding.
49. Check that the comparison and Boolean operators are correct.
50. Check throw-catch expressions, and check that the error condition is actually legitimate.
51. Check that the code is free of any implicit type conversions.

2.16 Exceptions

52. Check that the relevant exceptions are caught.
53. Check that the appropriate action are taken for each catch block.

2.17 Flow of Control

54. In a `switch` statement, check that all cases are addressed by break or return.
55. Check that all switch statements have a default branch.
56. Check that all loops are correctly formed, with the appropriate initialization, increment and termination expressions.

2.18 Files

57. Check that all files are properly declared and opened.
58. Check that all files are closed properly, even in the case of an error.
59. Check that EOF conditions are detected and handled correctly.
60. Check that all file exceptions are caught and dealt with accordingly.

2.2 Issues found

We report all the issues we found, with the corresponding lines and the type of violation:

1. The class name, and the methods name are not very meaningful. All the methods have almost the same name.

7. The constant “module” at the line 55 isn’t in uppercase.

8. In the line 282 there is an indention that not respect the convention of four space.

10. Kernighan and Ritchie convention is used consistently.

11. In the line 268 there is an if construct of one statement without the curly braces.

12. There are some blank lines not used to separate sections, 226,230,234,237,244,279.

13. Many lines exceed the limit of 80 characters, lines: 66,77,86,90,93,98,101,112,118,120,122,127,129,131,134,137,143,196,197,203,206,220,221,228,252,253,266,271,282

14. Some of that lines exceed also the limit of 120 characters: 81,143,158,187,196,197,220,252,253,266,271

16. High-level breaks are not used in the line of code that exceed the limit of characters.

18. In the line 249 there is a TODO comment that should be removed.

22/23. Javadoc is almost absent in this class, there is only a simple and not exhaustive explanation of the main method.

27. All the methods of this class are a bit too long, but maybe is not possible to abbreviate them.

33. There are some variables that were not declared at the beginning of a block, in lines: 135,145,169,170,178:185,213,235,238,245,246,247,248,249,267,284.

40. Most of the object were not compared with the equal method, but with “==” or “!=” , especially in the lines: 201,208,223,231,270.

3. Appendix

3.1 Used Tools

The tools we used to produce this document are:

- Microsoft Word
- Github
- A TextEditor

3.2 Hours of Work

Luca Santini: 8 hours
Riccardo Remigio: 4 hours