POLITECNICO DI MILANO
Computer Science and Engineering
Software Engineering 2 Project

Code Inspection Document

Danchenko Elena, 874840
Cilloni Stefano, 880924

February 5th, 2017

Document version: 1.1

Contents

1. Introduction

1.1. Purpose

The aim of this document is to define the usage and to find the problems related to a java class, part of the main project: Apache OFBiz. The code inspection activity has many goals. First of all, it is useful to improve the quality of the code by finding bugs, missing elements, unadopted conventions or just syntactic errors. Furthermore, developers who do code inspection can improve their skills, augmenting the ability to understand code written by other. Following the guideline to built this document [1], we also have to realize what the given class is used for, and what is the context in which it is located.
In the second section of this document we provide an high view analysis of the software with its context, package and class analysis.

1.2. Scope

The Apache OFBiz project is an open source enterprise resource planning system. The software is made-up by framework components and applications to automate business processes inside an enterprise. These applications are used for ERP (Enterprise Resource Planning), CRM (Customer Relationship Management), SCM (Supply Chain Management), E-Business / E-Commerce, MRP (Manufacturing Resource Planning), MMS/EAM (Maintenance Management System/Enterprise Asset Management). The Apache OFBiz project provides a starting base for reliable, secure and scalable enterprise solutions.

1.3. Definitions and abbreviations

- *Apache:* open source company, notorious for its web server.
- *Apache OFBiz.* 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.
- *QR Code.* QR code is the abbreviation for Quick Response Code and it is a type of matrix barcode (or two-dimensional barcode). So, it is an image consisting of a matrix with a checkered black and white texture that contains information and that can be read very quickly with a photocamera.
- *K&R style*: Indentation style named after Kernighan and Ritchie, who used this style in their book "The C Programming Language".

2. Description of the code

2.1. Assigned classes

The class assigned to our group is:

- QRCodeServices

This class is located in the org.apache.ofbiz.common.qrcode package of the Apache OFBiz project.
The full path to exactly locate the class in the source code is:
../apache-ofbiz-16.11.01/framework/common/src/main/java/org/apache/ofbiz/common/qrcode/QRCodeServices.java

2.2. Functional role of the class

As documentation for OFBiz states that QRCodeServices are services for QRCode.
Services are independent pieces of logic which when placed together process many different types of business requirements. OFBiz has been characterized as having an "event-driven, service-oriented architecture". Service-oriented design of OFBiz is implemented via a context-aware Service Engine available for use across the entire framework, multiple invocation methods, chaining of Services, a fully integrated job scheduler for recurring and single use asynchronous job scheduling, variable service creation and implementation tools.
Using a factory pattern with Service Engine factory, OFBiz provides an easily extendable Service management and invocation tool supporting any number of concurrent Services and any number of third-party execution engines including, but not limited to: Java, Groovy, Javascript, JPython, and the OFBiz "simple" Service (based on the OFBiz Mini-Language.)

2.2.1. Common package

The QRCodeServices class is placed in the framework section of the project and in particular it is part of the Common package. The Apache OFBiz Project Overview page do not describe precisely the framework sections (but it does for the application main packages) and what we can deduce by the name of the package and its content is that it is used by many other parts of the framework and applications to accomplish common services.
The class to review is contained in the subpackage called: qrcode.
The subpackages of the Common package are:

- authentication
- email
- geo
- image

- login
- period
- preferences
- qrcode
- scripting
- status
- test
- uom

What we can state by the name of our subpackage (qrcode), by its position (inside the Common package) and referring to the documentation of the QRCodeServices [3] is that the services provided by the subpackage are for QR codes management.

2.2.2. Qrcode subpackage

The qrcode package is contained in the Common package of OFBiz and it contains two classes (one of which is the one we have to inspect), that are:

- QRCodeEvents
- QRCodeServices

By the Javadoc of the package [6], what we can know about the first class is that it deals with the events related to QR codes like the streaming of an image as output.

2.2.3. QRCodeServices class

The QRCodeServices class is a service for QR code of OFBiz.
As any other service in OFBiz QRCodeServices has a public method that returns a Map<String, Object> object to return back the main object managed by the service. In this case a QR code image.
In fact, QRCodeServices's public method is called *generateQRCodeImage* and as it is seen from the name the service generates a QR code image.
As parameters it takes the instances of the class **DispatchContext** and a **context** passed by a Map<String, Object>.
DispatchContext object provides all the necessary environmental information required to invoke a Service.
The Map <String, Object> **context** contains all the information required for that particular invocation to generate QRCode according the requested parameters.
Apart from generic Java classes and OFBiz classes QRCodeServices also uses a java based barcode reader and generator library called ZXing. Finally, it uses also the class from Apache FreeMarker that is a Java library to generate text output.

2.2.4 Class usage

The class is used just one time in the xml file associated to the QR code services. The file has the name *services_qrcode.xml* and it is located in the folder *common/servicedef*.

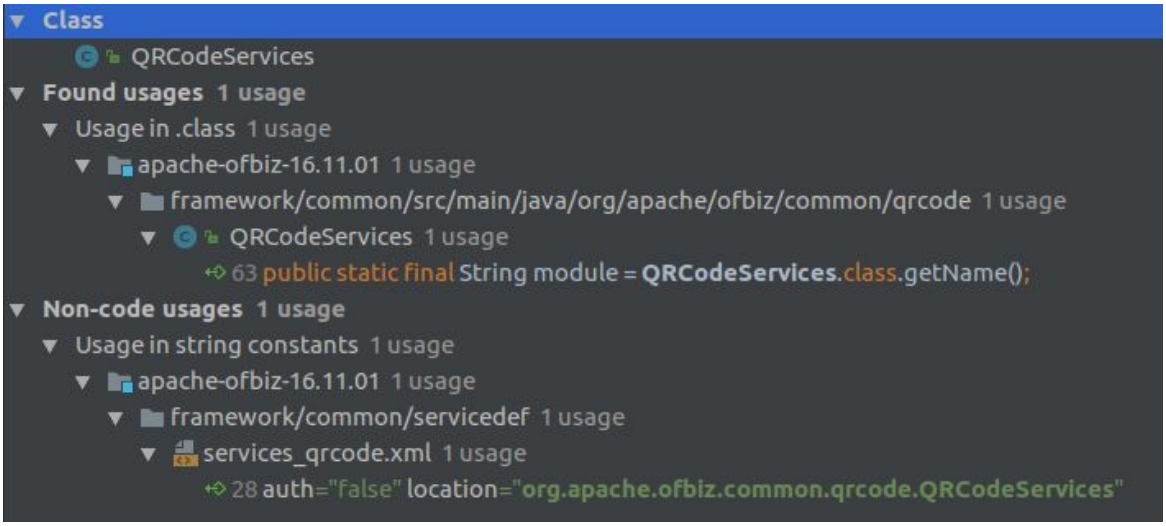

Figure 1. Usage of the QRCodeServices class

3. Results of inspection

This chapter contains the results of the code inspection that we did on the assigned classes and methods. All the points of the checklist [1] have been checked.

3.1. Notation

- The items of the code inspection checklist [1] will be referred as follows: C1, C2, etc.
- L.X indicates the X-th line of code of the class
- L.X-Y indicates the interval of lines of code between X and Y, bounds included

3.2. Inspection checklist

3.2.1. Naming conventions

All given names (to the class, to methods, class variables, etc.) are, correctly, sufficiently meaningful [C1]. No one-character variables is used [C2].
Mainly the methods names, as the convention wants, are verbs:

- *generateQRCodeImage* L.106
- *createMatrixFromImage* L.265

But, one method, *toBufferedImage* (L.248) is not precisely a verb even if it suggests a conversion and then an action [C5].
All class and methods variables respect the convention to start with a lowercase letter or with an underscore [C6].
The constant *module* (L.63) is declared using lower case [C7].

3.2.2. Indentation

The indentation is in the most cases correct, with only four spaces, except for the following lines that mix both tabs and spaces [C8][C9]:

- L.95
- L.168-172
- L.192
- L.196

3.2.3. Braches

In the class it is used, in a consistent way without any exception, the K&R style (Kernighan and Ritchie) that put the first curly brace on the same line as the instruction that opens a new code block [C10]. Moreover, every control statement (if, while, do-while, try-catch, for) with only one instruction, lot of try-catch and if, are surrounded by curly braces [C11].

3.2.4. File Organization

This is the overall file configuration:

- L.1-18 Apache license terms
- L.19 package definition
- L.21-32 java imports
- L.34-40 Apache OFBiz local modules
- L.42.54 Zxing library imports
- L.56 Freemarker import
- L.63-87 constants declaration. Note that L.25 is a declaration of an instance variable and it is mixed with other static variable declarations. It should be placed after the other declarations (L.87).
- L.106 *generateQRCodeImage* method
- L.107-116 initialization of context variables
- L.118-144 variables validation and initialization
- L.147-163 *bitMatrix*
- L.165-181 *newBufferedImage* creation
- L.183-222 *newBufferedImage* validation
- L.227-235 grouped catch statements
- L.248 *toBufferedImage* method
- L.249-253 variables initialization
- L.254-256 os type checking for image format correction
- L.257-261 for-loop to create the image
- L.265 *createMatrixFromImage* method
- L.266-269 variables initialization
- L.272-282 nested for-loop for matrix creation

Blank lines are used correctly to separate imports of different kind, methods and sections of code that act on different concepts [C12].
The line length convention is respected in most of the cases. The 14.33% of the lines (41 lines) exceeds by much or little the length of 80 characters [C13]. Moreover the lines L.65, L.67, L.69, L.71, L.73, L.92, L.95, L.119, L.125, L.132, L.138, L.147, L.153, L.167, L.173, L.179, L.208, L.217, L.228, L.230, L.232, L.234 exceed also 120 symbols [C14] and should be broken respecting the conventions.

3.2.5. Wrapping Lines

The present broken lines are overall well broken: after a comma or an operator [C15].
All statements are aligned with the beginning of the expression at the same level as the previous line [C17].

3.2.6. Comments

The class is poorly commented in general. The main class description that states "*Services for QRCode.*" is not much exhaustive and could be improved.
Comments that should contain relevant information to reading and understanding the class are almost absent.
The last method of the class, *createMatrixFromImage*, does not have any comment at all. Moreover all three methods bodies lack of comments [C18].

3.2.7. Java Source Files

The file contains correctly only one class, and its public [C20].
The class is the first and only class of the file [C21].
The file does not contain much documentation that can be used to generate the Javadoc. The only description for the class is "*Services for QRCode.*".
Since the last method *createMatrixFromImage* does not have any description it is not described also in the Javadoc.
The method *toBufferedImage* is well described in the comment before of its statement in the class. However, this description is not present in the public available Javadoc [3] and that means that the Javadoc is out of date and not updated with the comment [C23].

3.2.8. Package and Import Statements

As written before in the section 3.2.4. File organization, both the package definition and the imports (from java or from the project itself) are correctly included as first statements of the file before the class definition.

3.2.9. Class and Interface Declarations

The class declarations have the following order [C25]:

1. Software license terms (L.1-18)
2. Class description comment (L.58-60)
3. Class statement (L.61)

4. Class variables (L.63-87). All are static variables except for the instance variable *defaultLogoImage* that ought to be placed after the constants.
5. No constructor is defined for the class. It could be a good practice to add a private constructor to hide the implicit public one.
6. The three public methods are declared:
    - *generateQRCodeImage*
    - *toBufferedImage*
    - *createMatrixFromImage*

The code does not have duplicates and log methods. The only debug and logging statement is the one at the L.95 *"Debug.logError("Your logo image file(" + QRCODE_DEFAULT_LOGOIMAGE + ") cannot be read by javax.imageio.ImageIO. Please use png, jpeg formats instead of ico and etc.", module);*". Encapsulation of some functions could be improved breaking down into more methods the biggest method of the class: *generateQRCodeImage*. The last two ones are well structured and they're not too long [C27].

3.2.10. Initialization and Declarations

The class variables in lines L.63-83 are declared public without any need. It would be better to declare them as *private* [C28].
All the variables are declared in the right scope [C29].
All constructors are called when a new object is needed [C30].
All object references are initialized before use [C31].
Variables are initialized where they are declared, unless dependent upon a computation [C32].
The convention of declaring variables at the beginning of blocks is not respected for the variables [C33]:

- *newBufferedImage* (L.165)
- *dimensionMap* (L.171)
- *logoImageResult* (L.173)
- *newBitMatrix* (L.176)
- *graphics* (L.178)
- *result* (L.224)

3.2.11. Method Calls

All the methods are called in the correct way, using the right parameters and in the right order [C34][C45]. Moreover, the returned value is used properly in every section of the class [C36].

3.2.12. Arrays

Both the methods *toBufferedImage* and *createMatrixFromImage* use nested for-loops to iterate over a matrix. In both methods the indexes are well structured and they cannot generate any wrong access to the arrays [C37][C38].
All arrays and matrixes of the class are correctly instantiated with their constructor when they're needed [C39].

3.2.13. Object Comparison

All the objects are correctly compared with the .equals method. For the null comparison are used, correctly, the "==" and "!=" symbols [C40].

- *.equals* at L.204, L.207, L.216, L.254
- "==" at L.121, L.127
- "!=" not occur

3.2.14. Output Format

The only one displayed log message (L.95 "*Your logo image file(" + QRCODE_DEFAULT_LOGOIMAGE + ") cannot be read by javax.imageio.ImageIO. Please use png, jpeg formats instead of ico and etc.*") does not contain any spelling or grammatical error [C41].
This debug/error message is well understable and state clearly the error and how to correct it [C42]. It is also well formatted with correct reference included to let the user understand better which file caused the error [C43].

3.2.15. Computation, Comparisons and Assignments

No brute programming aspects are present in the class [C44] and no cases in which the operator precedentedness should be explicated [C45] are present. Consequently there is no need to use additional parenthesis to avoid operator precedence problems [C46].
All arithmetics calculations are done over integer values and no division by zero can occur since the only two divisions present are done in L.29 and divide an integer by 2 [C47]. On the same line, both the divisions are not managed properly because the final values passed to the object constructor are truncated implicitly [C48].
The code is free of any implicit type conversion [C51].

3.2.16. Exceptions

All the relevant exceptions in the class are caught [C52] and the appropriate actions (mostly error handling) are taken for each catch block [C53].

3.2.17. Flow of control

There are no control statement with the exception of conditional statement [C54][C55].
Each for loop is correctly formed, with the appropriate initialization, increment and termination expressions [C56].

3.2.18. Files

The class does not handle any file.

4. References

[1] Code Inspection Assignment Task Description.pdf

[2] Apache OFBiz Project Overview web page: https://ofbiz.apache.org/apache-ofbiz-project-overview.html

[3] https://ci.apache.org/projects/ofbiz/site/javadocs/org/apache/ofbiz/common/qrcode/QRCodeServices.html

[4] https://github.com/zxing/zxing

[5] Apache OfBiz Cookbook http://dl.finebook.ir/book/69/10826.pdf

[6] https://ci.apache.org/projects/ofbiz/site/javadocs/org/apache/ofbiz/common/qrcode/package-summary.html

5. Hours of work

5.1. Elena Denchenko

- 2017-01-28: 2h
- 2017-01-29: 2h 30min
- 2017-01-30: 1h
- 2017-02-02: 2h
- 2017-03-03: 1h
- 2017-03-04: 2h

5.2. Stefano Cilloni

- 2017-01-24: 2h
- 2017-01-29: 2h 30min
- 2017-01-30: 1h
- 2017-03-04: 3h
- 2017-03-04: 2h

6. Changelog

- v 1.1
    - Section 3.2.1. Naming conventions improved
    - Minor corrections