Version 1.4

Software Engineering Project

Code Inspection Document

POLITECNICO
MILANO 1863

Vianney Payelle - Rémi Rigal - Noëlie Ramuzat

Revision Notice

Versions	Date	Description	Modifications
V1.1	29/01/2017	Creation of the document	_
V1.2	31/01/2017	Creation of: - The Introduction part - The List of issues part	Addition of : - The Introduction part - The List of issues part
V1.3	04/02/2017	Creation of the functional role part	Modification of the List of issues part
V1.4	05/02/2017	Validation of the document	Modification of : - The functional role part - The List of issues part

Table 1 : Code Inspection versions

Table of Contents

List of tables

1. Introduction

1.1. Purpose

This document presents the examination of the source code of an Apache OFBiz project script. Its aim is to find the mistakes in the program and evaluate the quality of the code by applying a review checklist. Therefore, it allows to improve the quality of the delivered software.

1.2. Classes assigned to the group

The class assigned to our group is the PdfSurveyServices that provides a set of methods related to surveys. It can build PDF files from surveys, or read those files to create surveys.
There is no namespace in our class.

2. Functional role of the PdfSurveyServices class

The PdfSurveyServices class is exclusively used by the survey service and provides a set of methods to build surveys from PDF files or build PDF files from survey responses:

- buildSurveyFromPdf

This method return a survey build from the acro fields of a PDF file. The survey is of type *Map<String,Object>*, and represents a list of questions that can be answered by text, radio buttons, combo box etc…

- buildSurveyResponseFromPdf

This method is quite similar to the previous one, but instead of only building a survey from questions, it also takes into account the answers.

- getAcroFieldsFromPdf

This method fetch all the acro fields from a PDF file and put them into a dictionary.

- setAcroFields

This method set specific acro fields to a PDF file.

- buildPdfFromSurveyResponse

This method creates a PDF file from a survey response by creating a paragraph for each question and answer, then adding all the paragraphs to the document.

- buildSurveyQuestionsAndAnswers

This method creates a list that contains the questions and the answers to a specific survey response.

- setAcroFieldsFromSurveyResponse

This method sets acro fields and create a file with them from a survey response.

- getInputByteBuffer

This method returns a ByteBuffer instance that allows to fetch the content of a file.

3. List of issues found by applying the checklist

In this part are reported the fragments of the code that don't fulfil some points of the code inspection checklist.

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.*

In the setAcroFieldsFormSurveyResponse method, line 546-548:
The variable name ctx may not be clear enough to explain what it represents (the valid context needed to set the acrobat fields for the pdf and then run it).

2. *If one-character variables are used, they are used only for temporary\throwaway" variables, such as those used in for loops.*

In the buildSurveyResponseFromPdf, getAcroFieldsFromPdf and setAcroFields methods, respectively line 251, 302 and 338:
The variable name s, representing the PdfSTamper has to be more explicit. Indeed, even if it is in a block with temporary variables needed by a for loop, it is used at least three times outside the loop.

3. *Class names are nouns, in mixed case, with the first letter of each word in capitalized.*

OK

4. *Interface names should be capitalized like classes.*

NO INTERFACE

5. *Method names should be verbs, with the first letter of each addition word capitalized.*

OK

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.*

OK

7. *Constants are declared using all uppercase with words separated by an underscore.*

Two constants are used in the code and are not declared in uppercase:

- Line 73: public static final String module = PdfSurveyServices.class.getName();
  It should be MODULE.
- Line 74: public static final String resource = "ContentUiLabels";
  It should be RESOURCE.

3.2. Indentation

8. *Three or four spaces are used for indentation and done so consistently.*

Four spaces are used for each indentation except for the catch exceptions, the indentation in these code blocks are of two spaces.

9. *No tabs are used to indent.*

OK

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).*

The style of bracing is consistent, using the "Kernighan and Ritchie" one.

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

All the if, for and try-catch statements are surrounded by curly braces.
There is only one while statement and it is not surrounded by curly braces but written on the same line:
Line 585: while ((c = fis.read()) != -1) baos.write(c);
There is no do-while statement in the code.

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).*

The blank lines are moreover used to separate the end of statements from new variable declarations or attributions (after the if, else, for, try-catch) (for instance lines 195-197 or 309-310).
Finally they are also used to manage blocks of similar code such as variable declarations or loops (for example lines 142-147 or 118-121).

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

OK

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

When it exceeds 80 characters it is to declare a method, so it is better to be written on the same line (example line 79 for the buildSurveyFromPdf method).
But it also occurs in some other parts of the code, because of:

- The Map, needed to create the SurveyQuestionCategory, buildSurveyResponseFromPdf, buildPdfFromSurveyResponse, buildSurveyQuestionsAndAnswers on lines 108-109, 227, 390 and 460 with the line lengths of 246, 124, 125 and 127 characters.
- The logWarning that has to be register for the debug lines 132, 148, 156, the line length are respectively of 161, 156, 189 characters.
- The generic values surveyQuestionAppl, surveyResponse, surveyResponseAnswer and surveyQuestion of lines 191, 236, 242, 401, 269, 423, 467, 470, 495, 513 with the line lengths from 135 to 172 characters.
- The list of generic values responses lines 419, 467 and 508 with 143 characters.
- The return errors that exceed 150 characters per line several times (lines 210, 213, 216, 279, 282, 285).

3.5. Wrapping Lines

There are two expressions that are break on several lines in the code:
Lines 258-261: The generic value surveyQuestionAndAppl is defined on 4 lines.
Lines 515-519: The generic value surveyQuestionAppl is defined on 5 lines.
Both of them have lines breaking that occur after comma and parenthesis (high-level break) and are well aligned according to the level of the previous line expression.

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

OK

16. *Higher-level breaks are used.*

OK

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

OK

3.6. Comments

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

There are not enough comments to describe the methods or the class. All of the ones before the methods are empty and the one before the class only give its name.
The useful comments are dispatched in the code, before some parts that need to be explained.
The comments are also used for the TODO, the functionalities that have to be added or changed in the future:
Line 129: //TODO: handle these specially with the acroFields.getListOptionDisplay (and getListOptionExport?)
Line 159: // TODO: need to find something better to put into these fields…

And for advice for the future:
Line 425:

// DEJ20060227 this isn't used, if needed in the future should get from SurveyQuestionAppl.externalFieldRef String fieldName = surveyQuestion.getString("description");

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.*

The only commented out code line in the script is not commented or dated:
Line 256: //AcroFields.Item item = fs.getFieldItem(fieldName);

3.7. Java Source Files

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

OK: public class PdfSurveyServices.

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

OK

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

OK

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

All methods from the class are covered. There is a lack of comments.

3.8. Package and Import Statements

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

OK. The package org.apache.ofbiz.content.survey is the first statement of the code.

3.9. Class and Interface Declarations

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

- ***a.*** *class/interface documentation comment;*
  OK
- ***b.*** *class or interface statement;*
  OK
- ***c.*** *class/interface implementation comment, if necessary;*
  OK
- ***d.*** *class (static) variables;*
  - *i. first public class variables;*
    OK
  - *ii. next protected class variables;*
    NONE
  - *iii. next package level (no access modifier);*

NONE

iv. *last private class variables.*
NONE

e. *instance variables;*
i. *first public instance variables;*
NONE

ii. *next protected instance variables;*
NONE

iii. *next package level (no access modifier);*
NONE

iv. *last private instance variables.*
NONE

f. *constructors;*
NONE

g. *methods.*
OK

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

Ok, it makes sense: Build survey from pdf, Get/Set AcroField, build Pdf, build survey, Set AcroField from survey, GetInputByteBuffer.

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

There are no duplicated methods; the longest method has a length of 144 lines and the class size is of 600 lines. So the class size is not really big, but these two following methods are too long:
The buildSurveyFromPdf and setAcroFieldsFromSurveyResponse ones (their size should not be greater than 60 lines but they are composed respectively of 144 and 90 lines).
Each methods and class variables are public.

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).*

The class variables are both public with the correct type.
The class methods are all public and with the correct type according to what they return.

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

The class variables are defined at the beginning of the definition of the class.
And methods variables are declared at the beginning of each method with the default visibility.

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

OK

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

OK. The default initialization is most of the time used.

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

OK

33. *Declarations appear at the beginning of blocks (A block is any code surrounded by curly braces `f' and `g'). The exception is a variable can be declared in a for loop.*

Line 191: The GenericValue surveyQuestionAppl is not declarated at the beginning of a block
Line 366: The ByteBuffer outByteBuffer is not declarated at the beginning of a block
Line 444: The ByteBuffer outByteBuffer is not declarated at the beginning of a block

3.11. Method Calls

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

OK

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

OK

36. *Check that method returned values are used properly.*

OK

3.12. Arrays

37. *Check that there are no one-by-one errors in array indexing (that is, all required array elements are correctly accessed through the index).*

Only on array is filled, in line 141 where 5 values are manually indexed from 0 to 4.

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

OK. All collections referred trough loop are using foreach model or for loop and size property to define the bound.

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

There is only one array declared line 140, used to get value from another object and constructors are not called, moreover there is no operations performed on it.

3.13. Object Comparison

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

OK

3.14. Output Format

41. *Check that displayed output is free of spelling and grammatical errors.*

OK

42. *Check that error messages are comprehensive and provide guidance as to how to correct the problem.*

The errors are throwing exceptions or are logged in an adequate way, forwarding the corresponding exception but rarely adding information on the origin of the error.

43. *Check that the output is formatted correctly in terms of line stepping and spacing.*

OK

3.15. Computation, Comparisons and Assignments

44. *Check that the implementation avoids "brutish programming".*

OK

45. *Check order of computation/evaluation, operator precedence and parenthesizing.*

OK

46. *Check the liberal use of parenthesis is used to avoid operator precedence problems.*

OK

47. *Check that all denominators of a division are prevented from being zero.*

OK

48. *Check that integer arithmetic, especially division, are used appropriately to avoid causing unexpected truncation/rounding.*

Line 155: The three floats fieldPage, fieldLly and fieldLlx are casted into longs, potentially causing unexpected rounding.

49. *Check that the comparison and Boolean operators are correct.*

OK

50. *Check throw-catch expressions, and check that the error condition is actually legitimate.*

OK

51. *Check that the code is free of any implicit type conversions.*

OK

3.16. Exceptions

52. *Check that the relevant exceptions are caught.*

OK

53. *Check that the appropriate actions are taken for each catch block.*

All the exceptions are handled in the same way when catched in separate catch blocks, no specific action is taken for any exception.

3.17. Flow of Control

54. *In a switch statement, check that all cases are addressed by break or return.*

No switch statements.

55. *Check that all switch statements have a default branch.*

No switch statements.

56. *Check that all loops are correctly formed, with the appropriate initialization, increment and termination expressions.*

OK

3.18. Files

57. *Check that all files are properly declared and opened.*

OK

58. *Check that all files are closed properly, even in the case of an error.*

Line 91: The PdfStamper instance pdfStamper is closed line 202 but not in case of errors.

Line 251: The PdfStamper instance s is closed line 276 but not in case of errors.

Line 302: The PdfStamper instance s is never closed.

Line 338: The PdfStamper instance s is closed line 364 but not in case of errors.

59. *Check that EOF conditions are detected and handled correctly.*

OK

60. *Check that all file exceptions are caught and dealt with accordingly.*

OK

4. Hours of work

4.1. Vianney Payelle

04/02: 1h
05/02: 2h

4.2. Rémi Rigal

04/02: 2h
05/02: 2h

4.3. Noëlie Ramuzat

29/01: 1h
31/01: 3h
05/02: 1h