Politecnico di Milano

Software Engineering II project:
PowerEnjoy

Code Inspection

Gregori Giacomo and Ruaro Nicola

February 5, 2017
Version 1.0

Contents

Abstract

This document provides a detailed code inspection for the assigned classes in the Apache OFBiz project. It must properly describe the code issues and problems and provide fixing strategies, it should also be consultable as a code-fixing plan for the development team.

Introduction

1.1 Assigned Classes

The assigned class is **JavaMailContainer.java**. The class is composed by 392 lines of code and is located in the **org.apache.ofbiz.service.mail** package of the Apache OFBiz project.

1.2 Functional Role

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.
The **JavaMailContainer** class is part of the OFBiz eMail services, used to send/receive emails. Each JavaMailContainer is related to a single user email account.
The class exposes 4 public methods and 3 protected methods, respectively: **init**, **start**, **stop**, **getName**, **makeSession**, **getStore**, **updateUrlName**.

1.3 Issue Notation

All issue categories are labeled using numbers from the *Code Inspection checklist*[2] ([**n**] **Category Name**). When referring to specific lines of code, the notation **:LINE** is used.

Inspection

2.1 Issues

2.1.1 Naming Conventions

[1] Unmeaningful Names

:100 The **cfg** variable can be renamed to **config**.

:124 The meaning **prop** variable is not immediate. Considering that it is used as a throwaway variable, though, this is acceptable.

:208 The meaning of the **props** argument is not immediate. Since it is used in an extended context and not as a throwaway variable, it should be named **properties**.

:241 Similarly to the previous item, the **portProps** variable should be named **portProperties** instead.

[2] One-char variables

:114, :178, :188, :201, :246, :255, :305, :312, :385 The variable **e** is correctly used as a throwaway variable for Exceptions.

:166 The variable **p** is correctly used as a throwaway variable to loop through the **Property** attribute.

[3] Inconsistent Class Names

There are no inconsistent or improperly capitalized class names.

[4] Inconsistent Interface Names

There are no inconsistent or improperly capitalized interface names.

[5] Inconsistent Method Names

:208 The method **updateUrlName** is correctly capitalized according to the capitalization conventions. To integrate consistently with the rest of this project, though, this method should be named **updateURLName**.

[6] Inconsistent Attribute Names

There are no inconsistent or improperly capitalized attribute names.

[7] Inconsistent Constants Names

:61 The static variable **module** should be named **MODULE**, using uppercase letters.

2.1.2 Indentation

[8] Incorrect Indentation

Four spaces are used for indentation.

[9] Indentation Tabs

No tabs are used for indentation.

2.1.3 Braces

[10] Inconsistent Bracing Style

Consistent bracing style is used. In particular the "Kernighan and Ritchie" style is adopted (first brace is on the same line of the instruction that opens the new block).

[11] No Braces for One-Line Statements

:185, :265, :283, :363, :365, :369 **if (Debug.verboseOn())** is used as one-line statement, without braces.

2.1.4 File Organization

[12] Unseparated Sections

:18 The **package** statement has not been separated from the beginning comment. This is, though, a reasonable choice as it makes the code more readable.

:75 The **javadoc** for the **init** method has not been separated from the previous declarations.

:103 **this.deleteMail** should be grouped with the following assignments, beginning from line **:105**.

[13] **Exceeded 80char Line Lenght**

:70 The comment can eventually be rephased.

:84 The method declaration can be split after the **name** attribute.

:100 Wrapping this line is not practical and will compromise code readability.

:101 Wrapping this line is not practical and will compromise code readability.

:102 Wrapping this line is not practical and will compromise code readability.

:103 Wrapping this line is not practical and will compromise code readability.

:106 Wrapping this line is not practical and will compromise code readability.

:107 Wrapping this line is not practical and will compromise code readability.

:108 Wrapping this line is not practical and will compromise code readability.

:111 Wrapping this line is not practical and will compromise code readability.

:113 Wrapping this line is not practical and will compromise code readability.

:115 Wrapping this line is not practical and will compromise code readability.

:123 Wrapping this line is not practical and will compromise code readability.

:137 Wrapping this line is not practical and will compromise code readability.

:162 Wrapping this line is not practical and will compromise code readability.

:164 Wrapping this line is not practical and will compromise code readability.

:166 Wrapping this line is not practical and will compromise code readability.

:184 Wrapping this line is not practical and will compromise code readability.

:185 Wrapping this line is not practical and will compromise code readability.

:247 This line can be split after **protocol**.

:256 Wrapping this line is not practical and will compromise code readability.

:283 This line can be split after **typeString**.

:307 Wrapping this line is not practical and will compromise code readability.

:318 Wrapping this line is not practical and will compromise code readability.

:326 Wrapping this line is not practical and will compromise code readability.

:344 Wrapping this line is not practical and will compromise code readability.

:356 Wrapping this line is not practical and will compromise code readability.

:359 This comment can be easily split in two lines (eg. split after **continue**).

:382 Wrapping this line is not practical and will compromise code readability.

[14] Exceeded 120char Line Lenght

:135 This line can be split before **new PollerTask(dispatcher, userLogin)**.

:202 This line can be split after **"Unable to connect to mail store : "**.

:265 This line can be split after **protocol** and again before the following + operators where needed.

:357 This line can be split after **message.getFrom()[0]** and again before the following + operators where needed.

:363 This line can be split after **"Message from "** and again before the following + operators where needed.

:365 This line can be split after **"Message [**" and again before the following + operators where needed.

:369 This line can be split after **"Message [**" and again before the following + operators where needed.

2.1.5 Wrapping Lines

[15] Incorrect Line Breaks

Line break are correctly used.

[16] No Higher-level Breaks

No higher-level breaks are possible.

[17] Unaligned New Statements

All statements are aligned.

2.1.6 Comments

[18] Inadequate Comments

:59 Documentation for the **JavaMailContainer** class should be added.

:269 Documentation for the **LoggingStoreListener** class should be added.

:287 Documentation for the **PollerTask** class should be added.

:157 The **getName** method lacks documentation.

:162 The **makeSession** method lacks documentation.

:173 The **getStore** method lacks documentation.

:208 The **updateUrlName** method lacks documentation.

:272 The **notification** method lacks documentation.

:298 The **run** method lacks documentation.

:318 The **checkMessages** method lacks documentation.

:380 The **processMessage** method lacks documentation.

[19] Commented code

There isn't commented out code.

2.1.7 Java Source Files

[20] More Public Classes/Interfaces

The JavaMailContainer source file contains a single public class or interface.

[21] Non-Public Classes First

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

[22] Inconsistent External Interfaces Implementation

The external program interfaces are implemented consistently with what is described in the javadoc.

[23] Incomplete JavaDoc

The javadoc for the JavaMailContainer class is complete.

2.1.8 Package and Import Statements

[24] Unordered Package and Import Statements

The package statement is the first non-comment statement and is followed by import statements.

2.1.9 Class and Interface Declarations

[25] Unordered Class and Interface Declarations

The JavaMailContainer class lacks documentation and uses the default constructor. The variables are correctly listed (static, public, protected, package, private) and followed by the class methods and the inner classes.

[26] Incorrect Method Grouping

Methods are correctly grouped by functionality.

[27] Code Duplicates

No duplicates, long methods, big classes and breaking encapsulation have been detected. Moreover, coupling and cohesion are adequate.

2.1.10 Initialization and Declarations

[28] Incorrect Type or Visibility

:64 Variable **delegator** must be private and have accessor methods.

:65 Variable **dispatcher** must be private and have accessor methods.

:66 Variable **userLogin** must be private and have accessor methods.

:67 Variable **timerDelay** must be private and have accessor methods.

:68 Variable **maxSize** must be private and have accessor methods.

:69 Variable **pollTimer** must be private and have accessor methods.

:70 Variable **deleteMail** must be private and have accessor methods.

:72 Variable **configFile** must be private and have accessor methods.

:73 Variable **stores** must be private and have accessor methods.

:289 Variable **dispatcher** must be private and have accessor methods.

:290 Variable **userLogin** must be private and have accessor methods.

[29] Incorrect Scope

All the variables are declared in the proper scope.

[30] Not Called Contructors

:59 The JavaMailContainer class constructor is not explicitly declared. Using the default constructor is not recommended.

:269 The LoggingStoreListener class constructor is not explicitly declared. Using the default constructor is not recommended.

:64 The constructor for the **delegator** object should be called.

:65 The constructor for the **dispatcher** object should be called.

:66 The constructor for the **userLogin** object should be called.

:69 The constructor for the **pollTimer** object should be called.

:72 The constructor for the **configFile** object should be called.

:73 The constructor for the **stores** object should be called.

[31] Not Initialized Objects

:74 The **name** object should be initializated when declarated. Though, it is correctly initialized before use (assuming that the **init** method is correctly called).

:175 The Store **store** should be initializated when declarated. Though, it is correctly initialized before use.

:289 The LocalDispatcher **dispatcher** should be initializated when declarated. Though, it is correctly initialized before use (in the constructor).

:290 The GenericValue **userLogin** should be initializated when declarated. Though, it is correctly initialized before use (in the constructor).

[32] Incorrect Variable Initialization

All the variables inizialized, are inizialized with a correct value.

[33] Incorrect Variable Declaration

All the variables are declared at the beginning of blocks.

2.1.11 Method Calls

[34] Unordered Parameters

All the parameters are presented in the correct order.

[35] Incorrect Method Call

All methods calls are correct.

[36] Incorrect Use of Returned Values

All method's returned values are used properly.

2.1.12 Arrays

[37] Incorrect Array Access

All array structures are correctly managed (eg. :344: **messages**)

[38] Out-of-Bounds Array

The array's indexes have been prevented from going out-of-bounds.

[39] Not Called Constructors

:344 The **messages** array is not initialized as a new array, though it is correctly initialized using a method from the **javax.mail.Folder** class.

2.1.13 Object Comparison

[40] Incorrect Object Comparation

:193 **store** and **null** are compared with "==". This is correct, though, as reference are being compared and not objects.

:250 **portProps** and **0** are compared with "==". This is correct, though, as **portProps** is of primitive type **int**. Eventually the variable could be declared as **Integer**.

:338 **totalMessages** and **0** are compared with "==". This is correct, though, as **totalMessages** is of primitive type **int**. Eventually the variable could be declared as **Integer**.

2.1.14 Output Format

[41] Spelling/Grammatical Errors

All the output are free of spelling and grammatical errors.

[42] Non-Comprehensive Error Messages

All the error messages explain the problem in a comprehensive way.

[43] Incorrect Output Formatting

The outputs are formatted correctly in terms of line stepping and spacing.

2.1.15 Computation, Comparisons and Assignments

[44] "Brutish" Programming

The implementation avoids "brutish programming".

[45] Incorrect Operator Precedence and Parenthesizing

No computation/evaluation is present in the code.

[46] Incorrect use of Parenthesis

No computation/evaluation is present in the code.

[47] Division by Zero

No computation/evaluation is present in the code.

[48] Inappropriate Integer Arithmetic

No computation/evaluation is present in the code.

[49] Incorrect Boolean Comparison

All the comparisons and Boolean operators are correct.

[50] Inconsistent throw-catch

All the throw-catch expressions are well-formed.

[51] Implicit Type Comversion

No implicit type conversions in the code.

2.1.16 Exceptions

[52] Uncaught Exception

:164 A **NullPointerException** for the **client** object can be thrown. This should be monitored using opportune comparations, throw statements or catch blocks.

:177 A **NullPointerException** for the **session** object can be thrown. This should be monitored using opportune comparations, throw statements or catch blocks.

:208 A **NullPointerException** for the **urlName** and **proprs** objects can be thrown. This should be monitored using opportune comparations, throw statements or catch blocks.

:274 A **NullPointerException** for the **event** object can be thrown. This should be monitored using opportune comparations, throw statements or catch blocks.

:324 A **NullPointerException** for the **store** object can be thrown. This should be monitored using opportune comparations, throw statements or catch blocks.

[53] Inappropriate Catch Action

The try-catch blocks are well-defined.

2.1.17 Flow of Control

[54] Unaddressed Switch Cases

All switch cases are addressed properly (:274).

[55] Not Present Switch Default

:274 There isn't a default branch in the switch statement.

[56] Incorrect Loop

All loops are well-formed.

2.1.18 Files

[57] Improper File Declaration/Opening

No files are improperly declarated/opened.

[58] Improper File Closing

No files are improperly closed.

[59] Incorrect EOF Handling

No EOF is improperly handled.

[60] Improper File Exception Handling

No file exception is improperly handled.

2.1.19 Other Issues

The cyclomatic complexity of the **JavaMailContainer.updateUrlName** and **PollerTask.checkMessages** is, respectively, **14** and **13** and should be reduced to 10. Additional issues are:

:67 '**300000**' is a magic number. It should, instead, be declared as a constant(**static final**).

:68 '**1000000**' is a magic number. It should, instead, be declared as a constant(**static final**).

:107 '**300000**' is a magic number. It should, instead, be declared as a constant(**static final**).

:108 '**1000000**' is a magic number. It should, instead, be declared as a constant(**static final**).

Appendix A: Used Tools

A.1 LaTeX

Used to format and redact this document

A.2 *g*it

Used as version control system in order to lead development

Appendix B: Hours of work

These are the hours of work spent by each group member in order to redact this document:

- Ruaro Nicola: 7 hours
- Gregori Giacomo: 7 hours
- Total worktime: 14 hours

Appendix C: Revisions

These sections will be eventually redacted during future post-release updates in order to approach the ITPD modifiability providing a comfortable and higly effective way to trace changes:

Glossary

Brutish Programming see http://users.csc.calpoly.edu/ jdalbey/SWE/CodeSmells/bonehead.html.

Javadoc Javadoc is a documentation generator created by Sun Microsystems for the Java language for generating API documentation in HTML format from Java source code.

Bibliography

[1] Luca Mottola and Elisabetta Di Nitto, *Software Engineering 2: Project goal, schedule and rules*, 2016

[2] Luca Mottola and Elisabetta Di Nitto, *Assignment 3: Code Inspection*, 2016

[3] Apache OFBiz®, *org.apache.ofbiz.service.mail.JavaMailContainer.java*, 2016