CODE INSPECTION

POLITECNICO
MILANO 1863

Author: Riccardi Vincenzo 793241

Reference Professor: Mirandola Raffaela

Summary

1. Introduction

Purpose

Code inspection (e.g., code analysis, visual inspection, reverse engineering, etc.) is systematic examination (often known as peer review) of computer source code.

Scope

The scope of this document is to find mistakes overlooked in the initial development phase, improving both the overall quality of software and the developers' skills.

Document structure

1. Introduction

2. Classes that were assigned to the group

3. Functional role of assigned set of classes

4. List of issues found by applying the checklist

Sources and references

- Requirements Analysis and Specification Document (RASD), attached to this document.
- Document Design (DD), attached to this document.
- Code Inspection Checklist in “Assignment 3” document from : https://**beep**.metid.polimi.it

2. Classes and Method Assigned

The classes that were assigned to me are:

- **MQAddressList** from the MQAddressList.java file.

My aim is to analyze only three methods in this class:

- ***getHostNameFromDasProperties( )*** starts at line 200.
- ***getResolvedLocalJmsHostsInMyCluster(final boolean includeMe)*** starts at line 456.
- ***createUrl(JmsHost host , JmsService js , String overridedHostName)*** starts at line 570.

The scope of this assignment is a systematic examination of code to find mistakes overlooked in the initial development phase.

The analysis of code follows the checklist for JAVA code inspection reported in “Assignment 3” document.

3. Functional Role of Assigned Set of Classes

First Method

The first method to analyze is:

- ***getHostNameFromDasProperties( )***

This method is a getter which returns a string representation of HostName extracted from set of properties of Domain Admin Server (DAS).

The DAS controls the management of entire domain.

The Property class is a set (String *key* – String *value*) of properties.

This method creates a path:

"*System.getProperty(SystemPropertyCostant.INSTALL_ROOT_PROPERTY)/nodes/agent/config*", after using this path to create a file "dasPropsFile" with

the path as directory and containing the string "*das.property*" as child.

Afterwards, with the method "*Property.load()"* adds all DAS property to the set.

In the end extract the agent host name from the set and returns it.

Second Method

The second method to analyze is:

- ***getResolvedLocalJmsHostsInMyCluster(final boolean includeMe)***

The purpose of this method is to associate a specific Java Message Service host to a particular server.

The Java Message Service (JMS) API is a Java Message Oriented Middleware API for sending messages between two or more clients.

The method returns a Map (ServerName , JmsHost).

At first, it initializes the cluster with the cluster that contains the server "*myName*" using the method "*getClusterForServer (myName)*".

Afterwards, it creates a server list and puts them all the servers content in the cluster, so it tries and select the server "*myName*" in the list.

In the end, it adds the JMS host in the map.

Third Method

The third method to analyze is:

- ***createUrl(JmsHost host, JmsService js, String overridedHostName)***

This method is used to create a new URL setting all parameters that characterize it.

A Uniform Resource Locator (URL) is a reference to a web resource that specifies its location on a computer network and a mechanism for retrieving it.

At first, three parameters are set (name of URL, host name and port).

In the end, through the methods *"js.getMqScheme()"* and "js. getMqService()", the scheme and the specific service of URL are set.

4. List of Issues Found by Applying the Checklist

Naming conventions:

- Line 221 → For point 1 of checklist, I would use the "*dasPropertiesFilePath*" name of string instead of "*dasPropsFilePath*".

- Line 222 → For point 1 of checklist, I would use the "*dasPropertiesFilePath*" name of string instead of "dasPropsFilePath".

- Line 224 → For point 6 of checklist, the name of properties "*dasprops*" must be "*dasProps*".

- Line 225 → For point 1 of checklist, I would call differently the file input stream "*fis*". For example,
  I'd use a name more explanatory.

- Line 460 → For point 1 of checklist, "*buddies*" is not an appropriate name to represent an array of server. For example, I'd use "servers" name to represent it.

- Line 461 → For point 2 of checklist, I'd use only one character variable to the loop, because it is a "*throwaway*" variable.

- Line 597 → For point 2 of checklist, the exception could be called "*e*".

Indention:

- Line 234 → For the point 8 of checklist, the indentation is wrong because there are two spaces instead of three.

- Line 458 → For the point 8 of checklist, there is a space not necessary.

- Line 467 → For the point 8 of checklist, the indentation is wrong because there are eight spaces instead of four.

- From line 572 to line 596 → For the point 8 of checklist, the indentation is wrong because there are zero spaces instead of three.

Braces:

- Line 204 → For point 11 of checklist, all loops that have only one statement must have a different structure.

- Line 213 → For point 11 of checklist, all loops that have only one statement must have a different structure.

- Line 233 → For point 10 of checklist, there is a space not necessary.

- Line 462 → For point 11 of checklist, all loops that have only one statement must have a different structure.

- Line 466 → For point 10 of checklist, the style of "*try*" statement do not respect "*Allman*" and "*Kernighan and Ritchie*" rules.

- Line 468 → For point 10 of checklist, the style of "*catch*" statement do not respect "*Allman*" and "*Kernighan and Ritchie*" rules.

File Organization:

- Line 221 → For point 13 of checklist, the length of line is greater than 80 characters.

- Line 223 → For point 12 of checklist, there is a blank line useless.

- Line 456 → For point 13 of checklist, the length of line is greater than 80 characters.

- Line 472 → For point 12 of checklist, there is a blank line useless.

- Line 570 → For point 13 of checklist, the length of line is greater than 80 characters.

Wrapping Lines:

- Line 458 → For point 17 of checklist, incorrect alignment because the line is on the same level of the previous.

Comments:

- Line 219 → For point 19 of checklist, this comment should be removed because it is part of code not used.

- Line 240 → For point 18 of checklist, I'd change the comment because it not explains what the block of code are doing.

- Line 469 → For point 19 of checklist, this comment should be removed because it is part of code not used.

Java Source Files:

- The point 20 of checklist is fully respected.
- The point 21 of checklist is fully respected.
- The point 22 of checklist is fully respected.
- The point 23 of checklist is not respected because the Javadoc not cover methods assigned to me.

Package and Import Statements:

- The point 24 of checklist is fully respected because the import statements are placed at the beginning of the file.

Class and Interface Declarations:

- The class assigned to me respects the structure of the point 25 of checklist.

- The point 26 of checklist is not respected because the methods not are grouped by functionality. For example, the getters of the class.

- Line 200 → For the point 27 of checklist, the method "*getHostNameFromDasProperties()*" is too long, it can be divided in two different method. One method that creates the "*dasPropsFilePath*" and another method that manages the properties.

Initialization and Declarations:

- Line 456 → For the point 28 of checklist, the Boolean include me cannot be final.

- From line 570 to 600 → For the point 29 of the checklist, the attributes (name, hostname, port, scheme, service) should be declared at the beginning of the method.

- The point 30 of checklist is fully respected.

- Line 202 → For point 31 of checklist, the string "*dasPropertiesHostName*" is initialized out of the method and this could create problems.

- The point 32 of checklist is fully respected.

- The point 33 of checklist is fully respected.

Method Calls:

- The point 34 of checklist is fully respected.
- The point 35 of checklist is fully respected.
- The point 36 of checklist is fully respected because all the method returned values are used properly.

Arrays:

- Line 461 → Accessing array is securely.
- The point 37 of checklist is fully respected.
- The point 38 of checklist is fully respected.
- The point 39 of checklist is fully respected.

Object Comparison:

- All objects are compared with “equals”.

Output Format:

- The point 41 of checklist is fully respected.
- The point 42 of checklist is fully respected.
- The point 43 of checklist is fully respected.

Computation, Comparison and Assignments:

- The only operation is the string concatenation that respects all the point of checklist.
- The point 44 of checklist is fully respected.

- The point 45 of checklist is fully respected.
- The point 46 of checklist is fully respected.
- The point 47 of checklist is fully respected.
- The point 48 of checklist is fully respected.
- The point 49 of checklist is fully respected.
- Throw-catch expressions and the error condition is actually legitimate.
- Line 571 → For the point 50 of checklist, the "*try*" condition has too many statements.
- The point 51 of checklist is fully respected.

Exceptions:

- Line 571 → For the point 53 of checklist, the "*try*" condition has too many statements.

- Line 461 → For the point 52 of checklist, I'd also control the "*IndexOutOfBoundsExceptions*".

Flow of Control:

- The point 54 of checklist is fully respected.

- The point 55 of checklist is fully respected.

Files:

- The point 57 of checklist is fully respected.

- The point 58 of checklist is fully respected.

5. Appendix

The tools we used to create the Code Inspection Document are:

- Microsoft Office Word 2011.

I spent 35 hours to redacting and writing this document.