POLITECNICO
MILANO 1863

Code Inspection Document

Joan Ficapal Vila (876805), Nicolò Vendramin (879113)

February 5, 2017
v1.0

Revision History

Revision	Date	Author(s)	Description
0.1	22.01.17	N and J	Division of the roles
0.1	24.01.17	N	Issue tracking
0.1	25.01.17	J	Issue tracking
0.1	29.01.17	J	Issue tracking
0.1	01.02.17	N	Issue tracking
0.2	02.02.17	N and J	Document Editing
0.3	04.02.17	N and J	Writing the introduction
0.4	05.02.17	N and J	Conclusion of the document
1.0	05.02.17	N and J	First release

Hours of work

- Joan Ficapal Vila : 10 hours
- Nicolò Vendramin : 10 hours

Table of Content

1 Description of the Class

The purpose of this document is to perform an analysis of the code, also known as “Code Inspection”, of the assigned class. The goal of this process is to spot errors in the source code, possible non-compliance with “best practices” or other mistakes, following mainly the points of the checklist that has been provided to us.
The class that has been inspected is part of the Apache OFBiz project (ver- sion 16.11.01), an open source product for the automation of enterprise processes that includes framework components and business applications for ERP (Enterprise Resource Planning), CRM (Customer Relationship Management) and other business-oriented functionalities.

1.1 Identification of the Class

In this document the SSLUtil class of the Apache OFBiz project [1] will be inspected. The class is located in the package "org.apache.ofbiz.base.util".

1.2 Functional Role of the Assigned Class

The functional role of our assigned class is to provide utilities for setting up ssl connections with specific client certificates. We have come to this conclusion for mainly two reasons, the first one are the comments that the code contains, even though they are few, they clearly state that functionality. The second main reason is the name functions that our code contains, they are really intuitive once given the context (e.g: isClientTrusted(),checkClientTrusted() , getHostCertNormalCheck()). Checking the contained code we have verified our thoughts.

2 List of issues found by applying the checklist

2.1 Naming Conventions

Line 54: module should be put on capital letters.

```
public static final String module = SSLUtil.class.getName();
```

In lines 56 57 58, HOSTCERT should be changed in HOST_CERT since they are two worlds. In addition a comment would be needed to clarify the names.

```
  private static final int HOSTCERT_NO_CHECK = 0;
  private static final int HOSTCERT_MIN_CHECK = 1;
  private static final int HOSTCERT_NORMAL_CHECK = 2;
```

Line 60: loadedProps is not clear enough as a variable name and since the variable has to be used only two times is suggested to have it longer and clearer.

```
private static boolean loadedProps = false;
```

Line 104: mgrs is not meaningful, managers would be better and preferable.

```
TrustManager[] mgrs = new TrustManager[0];
```

Line 116: mgr is not meaningful, manager would be better and preferable.

```
for (TrustManager mgr: mgrs) {  //manager in managers?
```

Line 132: KeyMgrs is not meaningful, KeyManagers would be better and preferable.

```
List<KeyManager> keyMgrs = new LinkedList<KeyManager>();
```

Lines 133, 160: ksi is not meaningful, keyStoreInfo would be better and preferable.

```
for (ComponentConfig.KeystoreInfo ksi: ComponentConfig.getAllKeystoreInfos()) {…}
```

Lines 135, 162: ks is not meaningful, keyStore would be better and preferable.

```
KeyStore ks = ksi.getKeyStore();
KeyStore ks = ksi.getKeyStore();
```

Lines 154, 202, 220: tm is not meaningful, trustManager would be better and preferable.

```
MultiTrustManager tm = new MultiTrustManager();
TrustManager[] tm;
```

```
TrustManager[] tm;
```

Lines 201, 219: km is not meaningful, keyManager would be better and preferable.

```
KeyManager[] km = SSLUtil.getKeyManagers(ks, password, alias);
 KeyManager[] km = SSLUtil.getKeyManagers(alias);
```

2.2 Indention

The indention is correctly performed using four spaces as specified in the java best practices. Tabs are never used in the document.

2.3 Braces

All the braces follow the Kernighan and Ritchie" style apart from these two cases:

Line 173

```
    public static TrustManager[] getTrustAnyManagers() {

        return new TrustManager[] { new TrustAnyManager() };

    }
```

Line 191

```
    public static TrustManager[] getTrustManagers(KeyStore ks) throws
GeneralSecurityException {

        return new TrustManager[] { new MultiTrustManager(ks) };

    }
```

11.

We have detected four statements that have only one statement to execute surrounded by more than just braces:

Line 73, Line 80 (Same element repeated)

```
            for (X509Certificate cert: certs) {
                Debug.logImportant("---- " + cert.getSubjectX500Principal().getName() + " valid: " +
cert.getNotAfter(), module);

            }
```

Line 154

```
        if (tm.getNumberOfKeyStores() < 1) {
            Debug.logWarning("System truststore not found!", module);
```

```
      }
```

Line 183

```
            if (keyManagers[i] instanceof X509KeyManager) {
               keyManagers[i] = new AliasKeyManager((X509KeyManager)keyManagers[i], alias);
            }
```

Line 299, consists on five “if” in a row, here there’s one of them.

```

if (protocol != null && !protocol.equals("NONE")) {
            System.setProperty("java.protocol.handler.pkgs", protocol);
         }
```

Apart from the ones mentioned above, we’ve found three more that we could consider correct because of the implication of else statements in lines 220, 202, 162.

2.4 File Organization

Blank lines are placed in a proper way making the file structure clear, but there’s a lack of explanatory comments to separate and explain sections.

Concerning the line length appeal, it follows more or less the required structure. On lines 73, 81, 130, 138, 152, 177, 195, 199, 213, 217, 231, 239 and 243 we’ve found lines that require more than 80 characters but are still surpassing the 120 character boundary.

2.5 Wrapping Lines

Line 139, the line should definitely be broken

```
if (Debug.verboseOn()) Debug.logVerbose("Loaded another cert store, adding [" +
(newKeyManagers == null ? "0" : newKeyManagers.size()) + "] KeyManagers for alias [" + alias
+ "] and keystore: " + ksi.createResourceHandler().getFullLocation(), module);
```

In general the breaks are not well used. The java standard suggests to break lines after no more than 120 character while the code that has been inspected overcomes that limit many times.

2.6 Comments

There are not many comments in this file, making it harder to understand it. The ones that appear are used only to clarify small operations of the code, but they are still way fewer than necessary.

2.7 Java Source Files

The javadoc is completely absent so it is not possible to check the consistency of the methods with the described interface. The lack of documentation can cause ambiguity in the interpretation of the external method interfaces that can lead to errors in their usage. In addition it will be required for the programmer
To check the class every time one or more of his element will have to be called in order to check exceptions, method interfaces and to check the behaviour.

2.8 Package and Import Statements

Package and import statements are done in the correct way.

2.9 Class and Interface Declarations

The class documentation comment is present but is too concise and should be expanded. The class statement is present after the comment. On the countrary the class implementation is not present and it would be necessary since no javadoc is present and the class is implementing an important functionality.
The order of the class variables is respected. They are correctly in the order suggested by the checklist. Instance variables are not present. The only private constructor of the class is correctly collocated before the other methods of the class. The private inner class present in the document is composed just by three methods and respects all the rules exception made for the fact that there is no comment neither to identify the role of the class nor to describe it and to describe the implementation.

2.10 Initialization and Declarations

The source is compliant with the points of the checlist.

2.11 Method Calls

All the methods are called in the correct way. The parameters are present in the correct order, the method called is always the right one, as far as what we can understand from the implementation. The expected return type is always the correct one.

2.12 Arrays

We have checked the few arrays in the file and all of them were treated following the requirements.

2.13 Object Comparison

In the line 139 (and other cases 115 136 163 182 300 303 306 309), an object is compared with null using the "==" operator. This is not a violation because, reading the code, we realised that the actual intention of the programmer is to check the value of the reference and not the referenced value.

2.14 Output Format

Line 271, Small grammar mistake, the article “The” should be added at the beginning of the string.

```
Debug.logWarning("Certificate is not valid!", module);
```

All the error messages are comprehensible, but they don’t provide an explanation or guide on how to solve the problem.

Line 138, the output is doesn’t contain grammatical mistakes but the output displayed can be too long, it should be better organized and stepped. Although it is using a function from a file of another class called Debug, we have checked it and it doesn’t manage that kind of problem.

```
            if (Debug.verboseOn()) Debug.logVerbose("Loaded another cert store, adding [" +
(newKeyManagers == null ? "0" : newKeyManagers.size()) + "] KeyManagers for alias [" + alias
+ "] and keystore: " + ksi.createResourceHandler().getFullLocation(), module);
```

2.15 Computation, Comparisons and Assignments

Line 70, although these two functions have different names that associate them to operate either in client or server domain, and also contain different log messages, they perform the same operation on the “for” loop. To reduce redundancy and improve the readability, a new function containing the loop could be called from both, for example checkTrustedAgents(), where agent can be clients or servers.

```
    public void checkClientTrusted(X509Certificate[] certs, String string) throws
CertificateException {
        Debug.logImportant("Trusting (un-trusted) client certificate chain:", module);
        for (X509Certificate cert: certs) {
            Debug.logImportant("---- " + cert.getSubjectX500Principal().getName() + " valid: " +
cert.getNotAfter(), module);

        }
    }

    public void checkServerTrusted(X509Certificate[] certs, String string) throws
CertificateException {
        Debug.logImportant("Trusting (un-trusted) server certificate chain:", module);
        for (X509Certificate cert: certs) {
            Debug.logImportant("---- " + cert.getSubjectX500Principal().getName() + " valid: " +
cert.getNotAfter(), module);
        }
    }
```

Line 105, this problem is placed here because it is using an excessive technique when more refined and effective alternatives are available. The GenericConfigException is already caught by the other two, hence it should be placed above them or be deleted.

```
    try {
        mgrs = SSLUtil.getTrustManagers();
    } catch (IOException e) {
        Debug.logError(e, module);
    } catch (GeneralSecurityException e) {
        Debug.logError(e, module);
    } catch (GenericConfigException e) {
        Debug.logError(e, module);
    }
```

Apart from the stated problems, the rest of the code is free of mistakes with regard to this section.

2.16 Exceptions

OK, checked exceptions are always caught in a meaningful way except for line 270 in which the catch block catches generic exception and writes on the debug without any information about the exception that has been caught. In addition at line 121 there is an empty catch block but is needed because in case an exception is raised the method should just go on looping. An option could be to write it on the debug log, or in another exception log.

Line 270

```
catch (Exception e) {
                    // certificate not valid
                    Debug.logWarning("Certificate is not valid!", module);
                    return false;
                }
```

Line 121

```
catch (CertificateException e) {
            // do nothing; just loop

        }
```

2.17 Flow Control

There’s just one switch statement and it is correct in terms of flow control.

Loops are properly done as well.

2.18 Files

There is no usage of Files in the code to be analysed.