POLITECNICO
MILANO 1863

Politecnico di Milano
A.A. 2015/2016
Software Engineering 2

Assignment 3: Code Inspection

Version 1.0

Alessandro Baldassari (mat. 841561)
Alberto Bendin (mat. 841734)
Francesco Giarola (mat. 840554)

December 18, 2015

Contents

1 Classes that were assigned to the group

We were assigned the “WebPermissionUtil.java” class, located in the path:

`appserver/security/core-ee/src/main/java/com/sun/enterprise/security/web/integration/`

In particular we had to analyze the methods:

- handleNoAuth(Permissions collection , MapValue m , String name)
- handleConnections(Permissions collection , MapValue m , String name)
- processConstraints(WebBundleDescriptor wbd , PolicyConfiguration pc)
- createWebRoleRefPermission(WebBundleDescriptor wbd , PolicyConfiguration pc)

2 Functional role of assigned set of classes

This class is used for generating Web permissions based on the deployment descriptor.

3 List of issues found by applying the checklist

Here are reported only the issues found while analyzing the code with the provided Java code inspection checklist.

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.
2. **If one-character variables are used, they are used only for temporary “throw-away” variables, such as those used in for loops.**
3. Class names are nouns, in mixed case, with the first letter of each word in capitalized. Examples: `class Raster`; `class ImageSprite`;
4. Interface names should be capitalized like classes.
5. Method names should be verbs, with the first letter of each addition word capitalized. Examples: `getBackground()`; `computeTemperature()`.
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. Examples: `_windowHeight`, `timeSeriesData`.
7. Constants are declared using all uppercase with words separated by an underscore. Examples: `MIN_WIDTH`; `MAX_HEIGHT`.

Indention

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

   In the following example three and four spaces are mixed, in the first line a tab and 4 spaces are used, while in the second line there are 2 tabs and 7 spaces.

   abstract from method "processConstraints"

```
490 	␣␣␣␣logger.log(Level.FINE,"JACC:␣constraint␣translation:␣
        CODEBASE␣=␣"+
491 		␣␣␣␣␣␣␣pc.getContextID());
```

9. **No tabs are used to indent.**

   The following example shows how tabs are often used, sometimes mixed with spaces too.

   abstract from method "processConstraints"

```
488 	if␣(logger.isLoggable(Level.FINE)){
489 	␣␣␣␣logger.entering("WebPermissionUtil",␣"processConstraints");
```

   One should avoid using tabs to indent code also because the interpretation of tabs varies with different IDEs or text editors.

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).**

    In the following example line 487 opens the method using the "Allman" style, all the other blocks follow the "Kernighan and Ritchie" style.

    abstract from method "processConstraints"

```
484     public static void processConstraints(WebBundleDescriptor wbd,
485                        PolicyConfiguration pc)
486     throws javax.security.jacc.PolicyContextException
487     {
488     if (logger.isLoggable(Level.FINE)){
489         logger.entering("WebPermissionUtil", "processConstraints");
490         logger.log(Level.FINE,"JACC: constraint translation:
                CODEBASE = "+
491                pc.getContextID());
492     }
```

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

File Organization

12. Blank lines and optional comments are used to separate subsections (beginning comments, package/import statements, class/interface declarations which include class variable/attributes declarations, constructors, and methods).

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

    In the following example lines 503 and 504 could have been broken in three lines instead of two. The same can be applied to other lines facing the same problem.

abstract from method “processConstraints”

```
501     boolean deny = wbd.isDenyUncoveredHttpMethods();
502     if (logger.isLoggable(Level.FINE)){
503         logger.log(Level.FINE,"JACC: constraint capture: begin
                processing qualified url patterns"
504                 + " - uncovered http methods will be " + (deny ? "
                        denied" : "permitted"));
505     }
506
507     // for each urlPatternSpec in the map
508     Iterator it = qpMap.values().iterator();
```

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

    All the lines which (even if arguable) reasonably exceed 80 characters never violate the limit of 120 characters.

Wrapping Lines

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

    In the following example line 503 is written wrong because the line-break precedes the ”+” operator.

abstract from method “processConstraints”

```
503         logger.log(Level.FINE,"JACC: constraint capture: begin
                processing qualified url patterns"
504                 + " - uncovered http methods will be " + (deny ? "
                        denied" : "permitted"));
```

16. Higher-level breaks are used.

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

    The whole method “processConstraints” lacks a level of indentation (is at the same level of the external code); an example of this is the opening of the method itself at line 488 (it is evident when the number of spaces associated to a tab is 4).

abstract from method “processConstraints”

```
484     public static void processConstraints(WebBundleDescriptor wbd,
485                       PolicyConfiguration pc)
486     throws javax.security.jacc.PolicyContextException
487     {
488     if (logger.isLoggable(Level.FINE)){
489         logger.entering("WebPermissionUtil", "processConstraints");
490         logger.log(Level.FINE,"JACC: constraint translation: CODEBASE = "+
491                 pc.getContextID());
492     }
```

Other examples are `while` loops or `if` statements at the same level of the upper-level code, like at line 541, where one level of indention is missing.

abstract from method "processConstraints"

```
539         Enumeration e = excluded.elements();
540         while (e.hasMoreElements()) {
541         Permission p = (Permission) e.nextElement();
542         String ptype = (p instanceof WebResourcePermission) ? "WRP  " : "
                WUDP ";
543         logger.log(Level.FINE,"JACC: permission(excluded) type: "+ ptype
                + " name: "+ p.getName() + " actions: "+ p.getActions());
544         }
```

The same is for line 548 and 564.

Comments

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

Java Source Files

20. Each Java source file contains a single public class or interface.
21. The public class is the first class or interface in the file.
22. Check that the external program interfaces are implemented consistently with what is described in the javadoc.
23. Check that the javadoc is complete (i.e., it covers all classes and files part of the set of classes assigned to you).

Package and Import Statements

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

Class and Interface Declarations

25. The class or interface declarations shall be in the following order:
    (a) class/interface documentation comment;
    (b) class or interface statement;
    (c) class/interface implementation comment, if necessary;
    (d) class (static) variables;
        i. first public class variables;
        ii. next protected class variables;
        iii. next package level (no access modifier);
        iv. last private class variables.
    (e) instance variables;
        i. first public instance variables;

ii. next protected instance variables;
iii. next package level (no access modifier);
iv. last private instance variables.

(f) constructors;

(g) methods.

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

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

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).

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

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

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

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

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

    In the following example the `if` statement at line 488 must be postponed till after line 501 and line 508 must be put before the block of line 509.

abstract from method "processConstraints"

```
484     public static void processConstraints(WebBundleDescriptor wbd,
485                          PolicyConfiguration pc)
486     throws javax.security.jacc.PolicyContextException
487     {
488     if (logger.isLoggable(Level.FINE)){
489         logger.entering("WebPermissionUtil", "processConstraints");
490         logger.log(Level.FINE,"JACC: constraint translation:
                CODEBASE = "+
491                 pc.getContextID());
492     }
493
494     HashMap qpMap = parseConstraints(wbd);
495     HashMap<String, Permissions> roleMap =
496         new HashMap<String, Permissions>();
497
498     Permissions excluded = new Permissions();
499     Permissions unchecked = new Permissions();
500
501     boolean deny = wbd.isDenyUncoveredHttpMethods();
502     if (logger.isLoggable(Level.FINE)){
503         logger.log(Level.FINE,"JACC: constraint capture: begin
                processing qualified url patterns"
504                 + " - uncovered http methods will be " + (deny ? "
                        denied" : "permitted"));
505     }
```

```
506
507     // for each urlPatternSpec in the map
508     Iterator it = qpMap.values().iterator();
```

The same is for:

- line 539 must be before line 537
- line 564 must be before line 561

Method Calls

34. Check that parameters are presented in the correct order.
35. Check that the correct method is being called, or should it be a different method with a similar name.
36. Check that method returned values are used properly.

Arrays

37. Check that there are no off-by-one errors in array indexing (that is, all required array elements are correctly accessed through the index).
38. Check that all array (or other collection) indexes have been prevented from going out-of-bounds.
39. Check that constructors are called when a new array item is desired.

Object Comparison

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

Output Format

41. Check that displayed output is free of spelling and grammatical errors.
42. Check that error messages are comprehensive and provide guidance as to how to correct the problem.
43. Check that the output is formatted correctly in terms of line stepping and spacing.

Computation, Comparisons and Assignments

44. Check that the implementation avoids "brutish programming": (see `http://users.csc.calpoly.edu/~jdalbey/SWE/CodeSmells/bonehead.html`).
45. Check order of computation/evaluation, operator precedence and parenthesizing.
46. Check the liberal use of parenthesis is used to avoid operator precedence problems.
47. Check that all denominators of a division are prevented from being zero.
48. Check that integer arithmetic, especially division, are used appropriately to avoid causing unexpected truncation/rounding.
49. Check that the comparison and Boolean operators are correct.
50. Check throw-catch expressions, and check that the error condition is actually legitimate.
51. Check that the code is free of any implicit type conversions.

Exceptions

52. Check that the relevant exceptions are caught.
53. Check that the appropriate action are taken for each catch block.

Flow of Control

54. In a `switch` statement, check that all cases are addressed by break or return.
55. Check that all switch statements have a default branch.
56. Check that all loops are correctly formed, with the appropriate initialization, increment and termination expressions.

Files

57. Check that all files are properly declared and opened.
58. Check that all files are closed properly, even in the case of an error.
59. Check that EOF conditions are detected and handled correctly.
60. Check that all file exceptions are caught and dealt with accordingly.

3.1 Any other problem you have highlighted

3.2 Additional Material

Reference to the checklist provided and to the code version.

3.3 Appendix

3.3.1 Software and tools used

- TeXstudio 2.10.4 (http://www.texstudio.org/) to redact and format this document.
- NetBeans 8.1 (https://netbeans.org/) to download, inspect and edit the code.

3.3.2 Hours of work

The time spent to redact this document:

- Baldassari Alessandro: 30 hours.
- Bendin Alberto: 30 hours.
- Giarola Francesco: 30 hours.