POLITECNICO DI MILANO
Computer Science and Engineering
Project of Software Engineering 2

Inspection Document
Ver. 1.0
Release date: January 5, 2016

Authors: Simone Rosmini (853949)
Vincenzo Viscusi (858689)
Matteo Zambelli (776162)

Reference Professor: Mirandola Raffaela

TABLE OF CONTENT

Methods division:

Name: `processParameters()`
Start Line: 1515
Location:
appserver/web/webcore/src/main/java/org/apache/catalina/connector/Request.java
Inspector: Matteo Zambelli

Name: `getRemoteAddr()`
Start Line: 1677
Location:
appserver/web/web-core/src/main/java/org/apache/catalina/connector/Request.java
Inspector: Simone Rosmini

Name: `getRemoteHost()`
Start Line: 1706
Location:
appserver/web/web-core/src/main/java/org/apache/catalina/connector/Request.java
Inspector: Simone Rosmini

Name: `getRequestDispatcher(String path)`
Start Line: 1813
Location:
appserver/web/web-core/src/main/java/org/apache/catalina/connector/Request.java
Inspector: Vincenzo Viscusi

Name: `getScheme()`
Start Line: 1859
Location:
appserver/web/web-core/src/main/java/org/apache/catalina/connector/Request.java
Inspector: Simone Rosmini

Name: `removeAttribute(String name)`
Start Line: 1917
Location:
appserver/web/web-core/src/main/java/org/apache/catalina/connector/Request.java
Inspector: Matteo Zambelli

Name: `setAttribute(String name, Object value)`
Start Line: 1960
Location:
appserver/web/web-core/src/main/java/org/apache/catalina/connector/Request.java
Inspector: Vincenzo Viscusi

General Inspection of the class Request.java

Location:
appserver/web/web-core/src/main/java/org/apache/catalina/connector/Request.java
Inspector: Vincenzo Viscusi

Naming Conventions

1. Almost all of the names used in the class seems to be meaningful.
2. We have analysed this point only for the assigned methods.
3. The class name (**Request**) is a noun with the first letter in capitalized. Also all the imported classes seems to respect this rule ;
4. All the imported and implemented interfaces are correctly capitalized.
5. Almost all method names are verbs with the first letter of each addition word capitalized.
6. All the class variables seems to respect this rule.
7. There are some constant variables that are declared in lowercase: **`log`** (line 159), **`rb`** (line 160), **`info`** (line 329), **`defaultLocale`** (line 406).

Indention

8. All the block of the class are correctly indented with four spaces.
9. No tabs are used to indent.

Braces

10. For all the blocks of the class is used the Kernighan and Ritchie style. To be consistent with the other log declarations the round close brace on line 322 should be placed on a new line.
11. All the blocks with only one statement are surrounded by curly braces.

File Organization

12. Each section of the class is separated by blank lines and comments, but the number of blank lines before and after the separation comments is not the same for all the sections.
    In fact most of the separation comments have one blank line before and none after, while the comment that identifies the beginning of the instance variables section on line 396 has two blank lines after and none before. Furthermore the comment on line 683 that identifies the constructor of the class has one more blank line than the other separation comments.

13. Most of the declarations at the beginning of the class are more than 80 character long, this could be avoided by using shorter names for the constant variables and by breaking the Log message strings on more lines.

14. The Log message strings on line 205, 253 and 291 are long more than 120 character, so they should be break on more lines to have a better view of the code.

Wrapping Lines

15. Analysed only for the assigned methods.
16. Analysed only for the assigned methods.
17. Analysed only for the assigned methods.

Comments

18. Probably the number and the grade of detail of the comments used is not enough to really understand all the functionality of the class.
19. Almost all the commented out code in the class contains neither a reason for being commented out nor a date after which it can be removed from the source file. In particular we have found commented out code without an explanation at the following lines:
    1885 to 1887, 1799, 1782, 1764, 1746, 1729, 1695, 1584 to 1588, 1602 to 1606, 1540 to 1544, 1503 to 1512, 1482 to 1490, 1371 to 1386, 3446 to 3535, 3581 to 3588, 3593 to 3704, 3743 to 3790, 3812 to 3834, 3864, 3882, 3953 to 3954, 3969 to 3977, 4088, 4101, 4692 to 4769.

Java Source Files

No problems found relative to this aspect, all the interfaces and classes seem to be implemented in the right way, even the Javadoc appears quite detailed and complete to explain the general functions of the various classes.

Package and Import Statements

24. The unique package statement needed is placed before the import statements at the beginning of the file.

Class and Interface Declarations

25. The correct order of the declarations is not respected:
    Static variables: (private declarations among public or protected declarations)
    The declarations on lines 159, 160 and 580 should be moved after the line 331.
    The declarations on lines 406 and 474 should be moved up before the line 332.
    The public declaration on line 661 should be moved among the public static declarations at the beginning of the class.
    Instance variables:
    The declarations on lines: 414, 502, 524, 565 and from line 576 to 591 should be moved down among the private variables after the line 627.
    The declarations from line 646 to 649 should be moved up among the protected variables to the line 398.
26. Methods are divided in sections depending on the type of request that the specific method manages. For example: `ServletRequest Methods, HttpRequest Methods.` There is also a division in public and protected methods which is not the best way of grouping methods, so at the end the division of methods could be done better.

27. There are some methods which contain only the comment `“NOT USED”` (see lines from 1244 to 1277), so they could be deleted.

Initialization and Declarations

29. See point 7.

32. There are some class attributes that are not initialized where declared, in particular at the following lines: 675, 680, 654, 555, 656, 649, 642, 636, 637, 615, 583, 505, 377, 385.

Inspection of the method setAttribute():

Start Line: 1960
Location:
appserver/web/web-core/src/main/java/org/apache/catalina/connector/Request.java
Inspector: Vincenzo Viscusi

Naming Conventions:

All the names used in the method seem to be meaningful. Also the case of the variables used in the method seems to be correct. There is the one character variable t which is used only for a catch block so it is throwaway when the catch block ends.

Indention:

The entire method is indented with four spaces and without tabs.

Braces:

In the method is used the Kernighan and Ritchie style in a way which is consistent, even for the blocks with only one statement.

File Organization

Line 1992 exceeds 80 characters, to avoid this problem the parameter passed to the method format could be moved on a new line, but in terms of readability maybe the solution used is not so bad, because after all the line doesn't exceed the 120 characters.

Wrapping Lines:

The parameters passed to the method `ServletRequestAttributeEvent` on lines 2023 and 2027 are not correctly indented, they could be indented in a better way by moving them under the variable `servletContext` and by moving one line up the third parameter passed to the method for both the lines.

Comments:

The comments used seem to be enough detailed to explain the functionality of the method and its parts.

Java Source Files:

Refers to the structure of the class, see general part.

Package and Import Statements

Refers to the structure of the class, see general part.

Class and Interface Declarations

Refers to the structure of the class, see general part.

Initialization and Declarations

Line 1988, the variable `canonicalPath` isn't initialized when declared.
The declarations on lines 1981, 2020, 2031 and 2037 don't appear at the beginning of a block.

Method Calls

All the methods called are correctly used in terms of both passed parameters and returned value.

Arrays

The method uses a List called listeners which is explored using an iterator. The iterator methods `hasNext()` and `next()` are used in a correct way so it's avoided any kind of collection error.

Object Comparison

All the comparison present in the method are correct, the comparisons with the null value (line 1963) has been done with the "==" operator and not with "equals" because otherwise it will cause a compilation error.

Output Format

There are some output messages used for log or for exceptions:
On line 2046 the methods writes a log with the message `"Exception thrown by attributes event listener"` which says from where the exception comes but not how to resolve the problem.
On line 1993 the method throws a new security exception with the message on line 199 which explains the problem but not how to resolve it.

Computation, Comparisons and Assignments:

All the computations done in the method are quite simple so there are no problems of brutish programming or other computational errors like type conversions errors or operator precedence errors.

Exceptions:

All the necessary exceptions are catch and thrown in the method.

Flow of Control:

The method uses only a while loop with an iterator to scan a list, the usage of the methods `hasNext()` and `next()` assures the correct exit from the loop.

Files:

The method creates a new file but it checks only the security permissions on the file using the method `checkRead()`,after that it passes the file to other methods, so it doesn't open the file and so it's not needed to close the file in this method.

INSPECTION OF THE METHOD getRequestDispatcher():

Start Line: 1813
Location:
appserver/web/web-core/src/main/java/org/apache/catalina/connector/Request.java
Inspector: Vincenzo Viscusi

Naming Conventions:
All the names used in the method seem to be meaningful. Also the case of the variables used in the method seems to be correct.

Indention:
The entire method is indented with four spaces and without tabs.

Braces:
In the method is used the Kernighan and Ritchie style in a way which is consistent, even for the blocks with only one statement.

File Organization:
Refers to the structure of the class, see general part.

Wrapping Lines:
16. Line 1828, the parameter of the method **`getAttribute`** could be shifted on the previous line without exceed the 80 character length.

Comments:
The comments used seem to be enough detailed to explain the functionality of the method and its parts.

Java Source Files:
Refers to the structure of the class, see general part.

Package and Import Statements:
Refers to the structure of the class, see general part.

Class and Interface Declarations:
Refers to the structure of the class, see general part.

Initialization and Declarations:
33. The declarations of the variables **`servPath`**, **`pInfo`**, **`requestPath`** and **`pos`** aren't at the beginning of the block.

Method Calls:
All the methods called are correctly used in terms of both passed parameters and returned value.
Arrays:
No collections are used in the method

Object Comparison:

All the comparison present in the method are correct, the comparisons with the null value (line 1820) has been done with the "==" operator and not with "equals" because otherwise it will cause a compilation error.

Output Format:
The method does not display any kind of output.

Computation, Comparisons and Assignments:
All the computations done in the method are quite simple so there are no problems of brutish programming or other computational errors like type conversions errors or operator precedence errors.

Exceptions:
The method doesn't use exceptions, it's not an error because of the simplicity of the operations executed in this method.

Flow of Control:
The method uses neither switches nor loops.

Files:
The method doesn't use files.

INSPECTION OF THE METHOD processParameters():

Start Line: 1515
End Line: 1528
Location: appserver/web/web-core/src/main/java/org/apache/catalina/connector/Request.java

If the parameters have already been processed the method does nothing, otherwise processes them.

Checklist

Naming Conventions

No issues.

Indention

No issues.

Braces

No issues.

File Organization

No issues.

Wrapping Lines

According to the Sun's Java coding conventions, paragraph 4.1 "Wrapping Lines": When an expression will not fit on a single line, break it according to these general principles:

- Break after a comma.
- Break before an operator.
- Prefer higher-level breaks to lower-level breaks.
- Align the new line with the beginning of the expression at the same level on the previous line.

- If the above rules lead to confusing code or to code that's squished up against the right margin, just indent 8 spaces instead.

Line 1523 is aligned by 12 spaces. 8 spaces would be sufficient.

Comments

There are no comments for this method, although it would take to better explain what the block of code does.

Java Source Files

There is not a javadoc description for this method.

Package and Import Statements

No issues.

Class and Interface Declarations

There are not problems with the points A, B, C, F, G.
there is no order as according to the points D and E but the variables are mixed.

Initialization and Declarations

No issues.

Method Calls

The variable name "parametersProcessed" and the method name "processParameters" are easy to confuse.

Arrays

No issues.
There isn't any array in this method.

Object Comparison

Inside the line 1522 there is a comparison with "!=" and not "!equals".

Output Format

No issues.

Computation, Comparisons and Assignments

The implementation avoids brutish programming though in the line 1517 the method is forced to end with “return;”. It could be written better.

For a better understanding, all operators in the lines 1520 and 1522 should be enclosed in parentheses

Exceptions

No issues.
There are not exceptions to handle in this method.

Flow of Control

No issues.
This method doesn’t have switch statements and loops.

Files

There are no issues because this method doesn’t use files.

INSPECTION OF THE METHOD getRemoteAddr():

Name: getRemoteAddr( );
Start Line: 1677
End Line : 1700
Location: appserver/web/web-core/src/main/java/org/apache/catalina/connector/Request.java

This method initializes a string remoteaddr if it equals null, otherwise it returns the same without changes.

Checklist

Naming Conventions

No issues.

Indention

In the lines 1682-3 the wrong use of the newline makes the indention also wrong.
In the lines 1694-5 too many blank after the tag comment (//) makes the indention wrong.

Braces

No issues.

File Organization

No issues.

Wrapping Lines

According to the Sun's Java coding conventions, paragraph 4.1 "Wrapping Lines":
When an expression will not fit on a single line, break it according to these general principles:

- Break after a comma.
- Break before an operator.
- Prefer higher-level breaks to lower-level breaks.

- Align the new line with the beginning of the expression at the same level on the previous line.
- If the above rules lead to confusing code or to code that’s squished up against the right margin, just indent 8 spaces instead.

No issues.

Comments

No issues.

Java Source Files

No issues.

Package and Import Statements

No issues.

Class and Interface Declarations

There are not problems with the points A, B, C, F, G.
There is no order as according to the points D and E but the variables are mixed.

Initialization and Declarations

No issues.

Method Calls

No issues.

Arrays

No issues.
There isn't any array in this method.

Object Comparison

No issues.

Output Format

In the line 1685 there is a constant UNABLE_DETERMINE_CLIENT_ADDRESS that explains the error with message code but doesn’t explain how you can resolve it.

Computation, Comparisons and Assignments

In the lines 1681 and 1684 there is a composed expression with two operators. For more understandability each operator could be in a parenthesis.

Exceptions

No issues.
There are not exceptions to handle in this method.

Flow of Control

No issues.
This method doesn’t have switch statements or loops.

Files

There are no issues because this method doesn’t use files.

INSPECTION OF THE METHOD getRemoteHost():

Name: getRemoteHost( )
Start Line: 1706
End line : 1734
Location: appserver/web/web-core/src/main/java/org/apache/catalina/connector/Request.java

This method initializes a string if the remote host is equal to null, otherwise it returns the same without changes.

Checklist

Naming Conventions

No issues.

Indention

In the lines 1712-3 there are 8 spaces instead 4.
In the lines 1728-9 there is a comment with a wrong indentation.

Braces

No issues.

File Organization

In the lines 1711,1718 the length is over 80 characters (not counting the space before the first character) but less than 120.
We could join the lines 1712-3 because the length is less than 120 characters and the line would become more readable.

Wrapping Lines

According to the Sun's Java coding conventions, paragraph 4.1 "Wrapping Lines":
When an expression will not fit on a single line, break it according to these general principles:

- Break after a comma.
- Break before an operator.
- Prefer higher-level breaks to lower-level breaks.

- Align the new line with the beginning of the expression at the same level on the previous line.
- If the above rules lead to confusing code or to code that’s squished up against the right margin, just indent 8 spaces instead.

No issues.

Comments

No issues.

Java Source Files

No issues.

Package and Import Statements

No issues.

Class and Interface Declarations

There are not problems with the points A, B, C, F, G.
There is no order as according to the points D and E but the variables are mixed.

Initialization and Declarations

No issues.

Method Calls

No issues.

Arrays

No issues.
There isn't any array in this method.

Object Comparison

No issues.

Output Format

In the line 1718 there is a constant UNABLE_RESOLVE_IP_EXCEPTION that explains the error with message code but doesn’t explain how you can resolve it.

Computation, Comparisons and Assignments

In the line 1711 there is a composed expression with two operators. For more understandability each operator could be in a parenthesis.

Exceptions

No issues.
The UnknownHostException is the only exception of the method and and appropriate actions are taken in the right way.

Flow of Control

No issues.
This method doesn’t have switch statements or loops.

Files

There are no issues because this method doesn’t use files.

INSPECTION OF THE METHOD getScheme():

Name: getScheme( )
Start Line: 1859
End Line : 1870
Location: appserver/web/web-core/src/main/java/org/apache/catalina/connector/Request.java

First, this method tries to initialize a ProxyHandler if a condition of a connector is true. Then if another condition on proxyHandle is true this method returns a string with "https". But if at least one of two conditions is false returns a string with the schema of a coyoteRequest.

Checklist

Naming Conventions

No issues.

Indention

No issues

Braces

No issues.

File Organization

No issues

Wrapping Lines

According to the Sun's Java coding conventions, paragraph 4.1 "Wrapping Lines": When an expression will not fit on a single line, break it according to these general principles:

- Break after a comma.
- Break before an operator.
- Prefer higher-level breaks to lower-level breaks.
- Align the new line with the beginning of the expression at the same level on the previous line.

- If the above rules lead to confusing code or to code that's squished up against the right margin, just indent 8 spaces instead.

No issues.

Comments

No issues.

Java Source Files

No issues.

Package and Import Statements

No issues.

Class and Interface Declarations

There are not problems with the points A, B, C, F, G.
There is no order as according to the points D and E but the variables are mixed.

Initialization and Declarations

No issues.

Method Calls

No issues.

Arrays

No issues.
There isn't any array in this method.

Object Comparison

No issues.
There isn't any comparison in this method.

Output Format

No issues.
There isn’t any type of output in this method.

Computation, Comparisons and Assignments

In the lines 1861 and 1863 there is a composed expression with two operators. For more understandability each operator could be in a parenthesis.

Exceptions

No issues.
There isn’t any exception in this method.

Flow of Control

No issues.
This method doesn’t have switch statements or loops.

Files

There are no issues because this method doesn’t use files.

INSPECTION OF THE METHOD removeAttribute():

Name: removeAttribute(String name)
Start Line: 1917
End Line: 1951
Location: appserver/web/web-core/src/main/java/org/apache/catalina/connector/Request.java

Remove the specified request attribute if it exists.

Checklist

Naming Conventions

In line 1944 the listener method name “attributeRemoved()” is not a verb.

Indention

No issues.

Braces

No issues.

File Organization

No issues.

Wrapping Lines

No issues.

Comments

There are not parts of code commented in this method.

Java Source Files

No issues.

Package and Import Statements

No issues.

Class and Interface Declarations

There are not problems with the points A, B, C, F, G.
there is no order as according to the points D and E but the variables are mixed.

Initialization and Declarations

No issues.

Method Calls

No real issues here, but the name of this method "removeAttribute" and the listener method name "attributeRemoved" in line 1944 are easy to confuse.

Arrays

No issues.

Object Comparison

There is no any comparison in this method.

Output Format

At the line 1948 there is a constant that explains the error displaying it to user but doesn't explain how you can resolve it.

Computation, Comparisons and Assignments

The implementation avoids brutish programming though in the line 1930 the method is forced to end with "return;". This part of code could be written better.
At the line 1939 the loop is forced to end with "continue;". Also this part of the code could be written in another way.

Exceptions

No issues.
The ATTRIBUTE_EVENT_LISTENER_EXCEPTION is caught and appropriate actions are taken in the right way.

Flow of Control

No issues.

Files

No issues.
This method doesn’t use files.

APPENDIX

Software and tools used

- Microsoft Office Word 2013: to write this document.
- Oracle VM Virtual Box
- Eclipse
- Github

References

- "Assignment 3: Code Inspection" pdf document.
- Sun's Java coding conventions.
- Software Engineering: Principles and Practice (Hans Van Vliet).

Hours of works

- Rosmini Simone: ~ 18 hours.
- Viscusi Vincenzo: ~ 18 hours.
- Zambelli Matteo: ~ 18 hours.