Politecnico di Milano

A.Y. 2015-2016

Software Engineering 2: “myTaxiService”

Code Inspection Document

version 1.0

Adrian Mihai Berbieru #854698

Attilio D’Onofrio #789614

05/01/2016

Table of contents

Classes that were assigned to the group

The portion of source code assigned to our group is inside the WebContainer class, whose path in the GlassFish project is appserver/web/web-glue/src/main/java/com/sun/enterprise/web/WebContainer.java
The methods contained in the WebContainer class we have inspected are the following ones:

1. *createJKConnector(NetworkListener listener , HttpService httpService)*
   Starts the AJP connector that will listen to call from Apache using mod_jk, mod_jk2 or mod_ajp.
2. *configureHttpServiceProperties(HttpService httpService , PECoyoteConnector connector)*
   Configure http-service properties.
3. *checkHostnameUniqueness(String listenerId , HttpService httpService)*
   Ensures that the host names of all virtual servers associated with the HTTP listener with the given listener id are unique.

Functional role of assigned set of classes

The WebContainer class, as the name itself states, is a container of utilities, i.e. methods, to manage operations related to connections. It's part of of a web server that interacts with Java servlets and has the responsibility of managing the lifecycle servlets, mapping a URL to a particular servlet and ensuring that the URL requester has the correct access rights. It handles requests for servlets, JavaServer Pages (JSP) files, and other types of files that include server-side code. The WebContainer creates servlet instances, loads and unloads servlets, creates and manages request and response objects, and performs other servlet management tasks. It also implements the web component contract of the Java EE architecture, specifying a runtime environment for web components that includes security, concurrency,lifecycle management, transaction, deployment, and other services.
The class uses the singleton pattern, which means that a single instance of WebContainer will be running on the server. There is a wide array of constants, declared as static final attributes - usually belonging to the string type.

List of issues found by applying the checklist

1. Constants “logger” and “rb” aren’t declared using uppercase letters with words separated by an underscore.
2. Lines: 1208, 1222, 1248, 1262, 1322, 1323, 1324, 1325, 1331, 1338, 1340, 1351, 1370, 1373, 1374, 1378, 1401 and 1402 exceed 80 characters in length.
3. A break within a parenthesized expression on line 1241.
4. The *“checkHostnameUniqueness”* is the only method not documented in JavaDoc, being a private method.
5. Both static and instance variables are declared in a random way regardless of the visibility of the variable (public, protected, private).
6. The two methods *“loadWebModule”* and *“updateHost”* are long methods which should be decomposed into smaller one for easier understanding.
7. In the two method “*configureHttpServiceProperties*” and “*checkHostnameUniqueness*” all comparisons are done through the use of *“==”* instead of *“.equals”*.

Code inspection checklist *(createJKConnector)*

```
protected WebConnector createJKConnector(NetworkListener listener,
                        HttpService httpService) {

                int port = 8009;
                boolean isSecure = false;
                String address = null;

                if (listener == null) {
                        String portString =

                System.getProperty("com.sun.enterprise.web.connector.enableJK");
                        if (portString == null) {
                                // do not create JK Connector if property is not set
                                return null;
                        } else {
                                try {
                                        port = Integer.parseInt(portString);
                                } catch (NumberFormatException ex) {
                                        // use default port 8009
                                        port = 8009;
                                }
                        }
                } else {
                        port = Integer.parseInt(listener.getPort());
                        isSecure =
        Boolean.valueOf(listener.findHttpProtocol().getSecurityEnabled());
                        address = listener.getAddress();
                }

                if (isSecure && defaultRedirectPort == -1) {
                        defaultRedirectPort = port;
                }

                if ("any".equals(address) || "ANY".equals(address)
                || "INADDR_ANY".equals(address)) {
                        address = null;
                        /*
                        * Setting 'address' to NULL will cause Tomcat to pass a
                        * NULL InetAddress argument to the java.net.ServerSocket
                        * constructor, meaning that the server socket will accept
                        * connections on any/all local addresses.
                        */
                }

                jkConnector = (WebConnector) _embedded.createConnector(address,
```

```
            port, "ajp");
            jkConnector.configureJKProperties(listener);

            String defaultHost = "server";
            String jkConnectorName = "jk-connector";
            if (listener != null) {
                    defaultHost =
listener.findHttpProtocol().getHttp().getDefaultVirtualServer();
                    jkConnectorName = listener.getName();
            }
            jkConnector.setDefaultHost(defaultHost);
            jkConnector.setName(jkConnectorName);
            jkConnector.setDomain(_serverContext.getDefaultDomainName());
            jkConnector.setInstanceName(instanceName);
            if (listener != null) {
                    jkConnector.configure(listener, isSecure, httpService);
                    connectorMap.put(listener.getName(), jkConnector);
            }

            if (logger.isLoggable(Level.INFO)) {
                    logger.log(Level.INFO, JK_LISTENER_CREATED,
                            new Object[]{listener.getName(), listener.getAddress(),
listener.getPort()});
            }

            for (Mapper m : habitat.<Mapper>getAllServices(Mapper.class)) {
                    if (m.getPort() == port && m instanceof ContextMapper) {
                            ContextMapper cm = (ContextMapper) m;
                            if (listener.getName().equals(cm.getId())) {
                                    jkConnector.setMapper(m);
                                    break;
                            }
                    }
            }

            _embedded.addConnector(jkConnector);


            return jkConnector;

    }
```

Naming Conventions

1- All the names used for variables, attributes and methods are self explanatory and do what their name says.
2- A one-character variable is used once within a generalized for loop.
3- The name of the class, WebContainer, is clearly a name.
4- The interfaces implemented by the WebContainer class all start with an uppercase letter.
5- The methods' name are properly capitalized and are verbs.
6- All class variables are properly written.
7- Not all constants are declared using only uppercase letters with words separated by an underscore: for example, the constants "*logger"*, "*rb"* (line 198) does not follow this guideline.

Indentation

8- Four spaces are consistently used for indentation, except when there are many parameters in the invoked methods: in that cases, a double indentation is used (eight spaces instead of four).
9- No tabs are used for indentation.

Braces

10- The variant 1TBS of "Kernighan and Ritchie" bracing style is consistently used in the class.
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 spaces and comments are properly used to separate different sections of the code.
13- Some lines exceed 80 characters of length, such as lines 1208, 1222, 1248 and 1262.
14- Lines exceeding 80 characters do not exceed 120 characters.

Wrapping Lines

15- Line breaks always occur after a comma or a logic operator.
16- The break in line 1241 occurs within a parenthesized expression, which is to avoid.
17- Statements of a level of indentation are aligned with the line of the same level.

Comments

18- Comments properly explain, sometime in a redundant way, what the code does.
19- In the methods inspected there is no commented out code.

Java Source Files

20- The java source file inspected contain a single class, which is the WebContainer.
21- The WebContainer is the first and only class in the source file.
22- There is no discrepancy between the implementation and the javadoc.
23- The *"checkHostnameUniqueness"* method is not included in the JavaDoc.

Package and Import Statements

24- Package statements and import statement are the first non-comment statement.

Class and Interface Declarations

25- The points D and E are not satisfied: in particular, public, protected and private variables, both static and instance are mixed.
26- Methods seem to be grouped by functionality rather than by accessibility.
27- The code contains methods, such as *"loadWebModule"* (line 1995) or *"updateHost"* (line 2973), that are sensibly larger (about 200 lines) than the average.

Initialization and Declarations

28- Variables and class members are of the correct type and they have the right visibility.
29- Variables are used within the methods in which they are declared
30- Constructors are called when a new object is needed.
31- Object references are initialized before use.
32- Variables are initialized where they are declared.
33- Declaration appear at the beginning of blocks or in a for loop.

Method Calls

34- Parameters are presented in the correct order.
35- Correct methods are called.
36- There are no problems with returned values of methods.

Arrays

37- The array elements are correctly accessed through the index.
38- Collection indexes have been prevented from going out-of-bounds.
39- Constructors are called when a new array item is desired.

Object Comparison

40- Objects are always compared with “equals”.

Output Format

41- Displayed output is error free.
42- Error messages provide guidance on how to fix the problem.
43- Output is formatted correctly.

Computation, Comparisons and Assignments

44- Implementation avoid brutish programming.
45- Order of computation and operator precedence are correct and parenthesizing avoids precedence errors.
46- The use of parenthesis avoids operator precedence problems.
47- All denominators of a division are prevented from being zero.
48- Integer arithmetic are not common in the inspected code but are used appropriately to avoid problems.
49- Comparisons and boolean operators are correct.

50- Throw-catch expressions have legitimate error condition.
51- Type conversion is always explicit.

Exceptions

52- Relevant exceptions are caught.
53- Catch block properly handle the relative errors.

Flow of Control

54- In switch statements all cases are addressed by break or return.
55- All switch statements have a default branch.
56- Loops have correct initialization, increment and termination expressions.

Files

57- Files are properly declared and opened.
58- Files are properly closed.
59- EOF conditions are detected and handled correctly.
60- File exceptions are caught and dealt with accordingly.

Code inspection checklist *(checkHostnameUniqueness)*

```
private void checkHostnameUniqueness(String listenerId,
                        HttpService httpService) {

    List<com.sun.enterprise.config.serverbeans.VirtualServer> listenerVses = null;

    // Determine all the virtual servers associated with the given listener
    for (com.sun.enterprise.config.serverbeans.VirtualServer vse :
httpService.getVirtualServer()) {
      List<String> vsListeners = StringUtils.parseStringList(vse.getNetworkListeners(), ",");
      for (int j = 0; vsListeners != null && j < vsListeners.size(); j++) {
        if (listenerId.equals(vsListeners.get(j))) {
          if (listenerVses == null) {
            listenerVses = new
ArrayList<com.sun.enterprise.config.serverbeans.VirtualServer>();
          }
          listenerVses.add(vse);
          break;
        }
      }
    }
    if (listenerVses == null) {
      return;
    }

    for (int i = 0; i < listenerVses.size(); i++) {
      com.sun.enterprise.config.serverbeans.VirtualServer vs
          = listenerVses.get(i);
      List hosts = StringUtils.parseStringList(vs.getHosts(), ",");
      for (int j = 0; hosts != null && j < hosts.size(); j++) {
        String host = (String) hosts.get(j);
        for (int k = 0; k < listenerVses.size(); k++) {
          if (k <= i) {
            continue;
          }
          com.sun.enterprise.config.serverbeans.VirtualServer otherVs
              = listenerVses.get(k);
          List otherHosts = StringUtils.parseStringList(otherVs.getHosts(), ",");
          for (int l = 0; otherHosts != null && l < otherHosts.size(); l++) {
            if (host.equals(otherHosts.get(l))) {
              logger.log(Level.SEVERE,
                    DUPLICATE_HOST_NAME,
                    new Object[]{host, vs.getId(),
                          otherVs.getId(),
                          listenerId});
            }
```

```
        }
      }
    }
  }
}
```

Naming conventions

1. All the names used for variables, attributes and methods are self explanatory and do what their name says.
2. All one-character variables are used as temporary variables.
3. The assigned class WebContainer has a name which is formed by nouns where the first letter of each noun is capitalized.
4. All interfaces are also capitalized like classes.
5. All methods' names are verbs and are correctly capitalized.
6. All attributes are correctly named and capitalized.
7. Most of the constants are declared using only uppercase letters with words separated by an underscore, the only exceptions are "*logger"* (at line 198) and "*rb"* (at line 201).

Indentation

8. Four spaces are used for indentation and done so consistently.
9. No tabs are used for indentation.

Braces

10. The *"Kernighan and Ritchie"* style is used throughout the class.
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. Comments and blank spaces are used to separate various sections of the class.
13. Some lines exceed 80 characters of length, such as lines 1370,1373, 1374, 1378, 1401 and 1402.
14. No line exceeds 120 characters of length.

Wrapping Lines

15. Line breaks always occur after a comma or a logic operator.
16. Only higher-level breaks are used.
17. All statements are aligned with the beginning of the expression at the same level as the previous line.

Comments

18. Comments explain, in various details, what each section of the class does.
19. There are no commented out code in the assigned methods.

Java Source File

20. The java source file inspected contains a single class: WebContainer.
21. The WebContainer is the first and only class in the source file.
22. The implementation and the Java Doc are coherent with each other.
23. The Java Doc cover all public classes but not the private ones like in WebContainer.

Package and Import Statements

24. Package statements and import statement are the first non-comment statements.

Class and Interface Declarations

25. The inspected class satisfies most of checks (A, B, C, F, G) while it does not satisfy D and E: the static and instance variables do not follow the normal declaration list (public, protected, package level, private).
26. Methods are group mainly based on functionality rather than accessibility.
27. The code contains methods, such as "*loadWebModule"* (line 1995) or "*updateHost"* (line 2973), that are sensibly larger (about 200 lines) than the average.

Initialization and Declarations

28. Variables and class members are of the correct type and they have the right visibility.
29. Variables are used within the methods in which they are declared
30. Constructors are called when a new object is needed.
31. Object references are initialized before use.
32. Variables are initialized where they are declared.
33. Declaration appear at the beginning of blocks or in a for loop.

Method calls

34. Parameters are presented in the correct order.
35. Correct methods are called.
36. There are no problems with returned values of methods.

Arrays

37. The array elements are correctly accessed through the index.
38. Collection indexes have been prevented from going out-of-bounds.
39. Constructors are called when a new array item is desired.

Object Comparison

40. None of the comparison in the assigned methods use the "equals", instead they rely on "==".

Output Format

41. There is no output to display.
42. Error messages provide guidance on how to fix the problem.
43. There is no output to display.

Computation, Comparisons and Assignments

44. The implementation avoids brutish programming.
45. Order of computation and operator precedence are correct and parenthesizing avoids precedence errors.
46. The use of parenthesis avoids operator precedence problems.
47. There are no divisions, therefore there is no problem of a division by zero.
48. There are no arithmetic operation in the assigned method.
49. Boolean operators and comparisons are used correctly.
50. There are no throw-catch statements in the assigned method.
51. Type conversion is always explicit.

Exceptions

52. There are no exception in the assigned method.
53. There are no catch statements.

Flow of control

54. There are no switch statements.
55. No switch statements present.
56. All loops are correctly formed.

Files

57. Files are properly declared and opened.
58. Files are properly closed.
59. EOF conditions are detected and handled correctly.
60. File exceptions are caught and dealt with accordingly.

Code inspection checklist *(checkHostnameUniqueness)*

```
 public void configureHttpServiceProperties(HttpService httpService,
                                PECoyoteConnector connector) {
     // Configure Connector with <http-service> properties
     List<Property> httpServiceProps = httpService.getProperty();

     // Set default ProxyHandler impl, may be overriden by
     // proxyHandler property
     connector.setProxyHandler(new ProxyHandlerImpl());

     globalSSOEnabled = ConfigBeansUtilities.toBoolean(httpService.getSsoEnabled());
     globalAccessLoggingEnabled =
ConfigBeansUtilities.toBoolean(httpService.getAccessLoggingEnabled());
     globalAccessLogWriteInterval = httpService.getAccessLog().getWriteIntervalSeconds();
     globalAccessLogBufferSize = httpService.getAccessLog().getBufferSizeBytes();
     if (httpServiceProps != null) {
       for (Property httpServiceProp : httpServiceProps) {
         String propName = httpServiceProp.getName();
         String propValue = httpServiceProp.getValue();

         if (connector.configureHttpListenerProperty(propName, propValue)) {
           continue;
         }
         if ("connectionTimeout".equals(propName)) {
           connector.setConnectionTimeout(Integer.parseInt(propValue));
         } else if ("tcpNoDelay".equals(propName)) {
           connector.setTcpNoDelay(ConfigBeansUtilities.toBoolean(propValue));
         } else if ("traceEnabled".equals(propName)) {
           connector.setAllowTrace(ConfigBeansUtilities.toBoolean(propValue));
         } else if ("ssl-session-timeout".equals(propName)) {
           connector.setSslSessionTimeout(propValue);
         } else if ("ssl3-session-timeout".equals(propName)) {
           connector.setSsl3SessionTimeout(propValue);
         } else if ("ssl-cache-entries".equals(propName)) {
           connector.setSslSessionCacheSize(propValue);
         } else if ("proxyHandler".equals(propName)) {
           connector.setProxyHandler(propValue);
         } else {
           String msg = rb.getString(INVALID_HTTP_SERVICE_PROPERTY);
           logger.log(Level.WARNING, MessageFormat.format(msg,
httpServiceProp.getName()));;
         }
       }
     }
   }
```

Naming conventions

1. All the names used for variables, attributes and methods are self explanatory and do what their name says.
2. All one-character variables are used as temporary variables.
3. The assigned class WebContainer has a name which is formed by nouns where the first letter of each noun is capitalized.
4. All interfaces are also capitalized like classes.
5. All methods' names are verbs and are correctly capitalized.
6. All attributes are correctly named and capitalized.
7. Most of the constants are declared using only uppercase letters with words separated by an underscore, the only exceptions are "*logger"* (at line 198) and "*rb"* (at line 201).

Indentation

8. Four spaces are used for indentation and done so consistently.
9. No tabs are used for indentation.

Braces

10. The *"Kernighan and Ritchie"* style is used throughout the class.
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. Comments and blank spaces are used to separate various sections of the class.
13. Some lines exceed 80 characters of length, such as lines 1322, 1323, 1324, 1325, 1331, 1338, 1340 and 1351.
14. No line exceeds 120 characters of length.

Wrapping Lines

15. Line breaks always occur after a comma or a logic operator.
16. Only higher-level breaks are used.
17. All statements are aligned with the beginning of the expression at the same level as the previous line.

Comments

18. Comments explain, in various details, what each section of the class does.
19. There are no commented out code in the assigned methods.

Java Source File

20. The java source file inspected contains a single class: WebContainer.
21. The WebContainer is the first and only class in the source file.
22. The implementation and the Java Doc are coherent with each other.
23. The Java Doc cover all public classes but not the private ones like WebContainer.

Package and Import Statements

24. Package statements and import statement are the first non-comment statements.

Class and Interface Declarations

25. The inspected class satisfies most of checks (A, B, C, F, G) while it does not satisfy D and E: the static and instance variables do not follow the normal declaration list (public, protected, package level, private).
26. Methods are group mainly based on functionality rather than accessibility.
27. The code contains methods, such as "*loadWebModule"* (line 1995) or "*updateHost"* (line 2973), that are sensibly larger (about 200 lines) than the average.

Initialization and Declarations

28. Variables and class members are of the correct type and they have the right visibility.
29. Variables are used within the methods in which they are declared
30. Constructors are called when a new object is needed.
31. Object references are initialized before use.
32. Variables are initialized where they are declared.
33. Declaration appear at the beginning of blocks or in a for loop.

Method calls

34. Parameters are presented in the correct order.
35. Correct methods are called.
36. There are no problems with returned values of methods.

Arrays

37. The array elements are correctly accessed through the index.
38. Collection indexes have been prevented from going out-of-bounds.
39. Constructors are called when a new array item is desired.

Object Comparison

40. None of the comparison in the assigned methods use the "equals", instead they rely on "==".

Output Format

41. There is no output to display.
42. Error messages provide guidance on how to fix the problem.
43. There is no output to display.

Computation, Comparisons and Assignments

44. The implementation avoids brutish programming.
45. Order of computation and operator precedence are correct and parenthesizing avoids precedence errors.
46. The use of parenthesis avoids operator precedence problems.
47. There are no divisions, therefore there is no problem of a division by zero.
48. There are no arithmetic operation in the assigned method.
49. Boolean operators and comparisons are used correctly.
50. There are no throw-catch statements in the assigned method.
51. Type conversion is always explicit.

Exceptions

52. There are no exception in the assigned method.
53. There are no catch statements.

Flow of control

54. There are no switch statements.
55. No switch statements present.
56. All loops are correctly formed.

Files

57. Files are properly declared and opened.
58. Files are properly closed.
59. EOF conditions are detected and handled correctly.
60. File exceptions are caught and dealt with accordingly.

Appendix

Hours spent

- Adrian Mihai Berbieru: ~4 hours
- Attilio D’Onofrio: ~4 hours