Politecnico di Milano
Software engineering 2

Glassfish

Code Inspection

Version 1.0
Release date 5th january 2016

790021 Antenucci Sebastiano
852461 Buonagurio Ilaria

Prof.ssa Di Nitto Elisabetta

Table of contents

1 Classes that were assigned to the group

As we can retrieve from the Assignment Search document http://assignment.pompel.me the classe that was assigned to our group (AntenucciBuonagurio) was the class Connector, whose namespace pattern is org.apache.catalina.connector.
The javadoc of this class can be found at http://glassfish.pompel.me/org/apache/catalina/connector/Connector

In particular we were assigned to analyze two methods: initialize() and translateAttributeName (String name).
The first method, initialize(), starts at line 1385. The second method, translateAttributeName (String name), starts at line 1544.
Both methods are located in appserver/web/web-core/src/main/java/org/apache/catalina/connector/Connector.java

2 Functional role of assigned set of classes

The class that was assigned to us, Connector, is a class that extends the class object and implements the interfaces Connector and Lifecycle. This class deals with the implementation of a Coyote connector for Tomcat 5.x. By reading the class's own methods we encounter the typical methods that are found in a class that manages the creation of a connection, especially when dealing with a socket connection. Moreover, the class deals with IP addresses, DNS and connections.
More in detail focusing on the methods that were assigned to us:

- initialize()
  This method initializes the connector and creates the ServerSocket.

- translateAttributeName(String name)
  This method translates the attribute name from the legacy factory names to their internal protocol names.

We have retrieved this information reading the class implementation, the comments and the javadoc of the class that can be found at http://glassfish.pompel.me/org/apache/catalina/connector/Connector .

3 List of issues found by applying the checklist

3.1 Connector.java and Connector

In the file Connector.java, the public class Connector is the first and only class contained, all class names, interface names, method names, class variables, method variables and constants used have meaningful names and do what the name suggests.
In particular class and interface names are nouns with the first letter of each word capitalized; method names are verbs with the first letter of each additional word capitalized; constants are declared using all uppercase words separated by an underscore.
In this file sections are usually separated by blank lines, but this don't always occur.
An example is shown in the following piece of code, where between the end of the comment and the declaration of the class there is no blank line.

```
/**
 * Implementation of a Coyote connector for Tomcat 5.x.
 *
 * @author Craig R. McClanahan
 * @author Remy Maucherat
 * @version $Revision: 1.23 $ $Date: 2007/07/09 20:46:45 $
 */
public class Connector
```

The javadoc related to this file is not completed, for example we can not find the class translateAttributeName(String name).
Package statements are correctly followed by import statements.
The declaration of the class static variables and of the instance variables does follow an order, but instead of following the public, protected, private order it follows the private, protected and public order. We can see an example in this piece of code.

```
private static final Logger log = StandardServer.log;
  private static final ResourceBundle rb = log.getResourceBundle();

  @LogMessageInfo(
        message = "The connector has already been initialized",
        level = "INFO"
  )
  public static final String CONNECTOR_BEEN_INIT =
"AS-WEB-CORE-00028";

  @LogMessageInfo(
        message = "Error registering connector ",
        level = "SEVERE",
        cause = "Could not register connector",
        action = "Verify domain name and type"
  )
  public static final String
ERROR_REGISTER_CONNECTOR_EXCEPTION = "AS-WEB-CORE-00029";
```

Methods are not grouped by functionality, scope or accessibility, even if a general pattern is followed: first getters and setters and then other methods. For example we can see that getter and setter are mixed with other methods, such as init() and destroy().

```
public String getDomain() {
        return domain;
    }

    /**
     * Set the domain of this object.
     */
    public void setDomain(String domain){
        this.domain = domain;
    }

    public void init() throws Exception {

        if( this.getService() != null ) {
            if (log.isLoggable(Level.FINE)) {
                log.log(Level.FINE, "Already configured");
            }
            return;
        }
    }

    public void destroy() throws Exception {
        if( oname!=null && controller==oname ) {
            if (log.isLoggable(Level.FINE)) {
                log.log(Level.FINE, "Unregister itself " + oname );
            }
        }
        if( getService() == null)
            return;
        getService().removeConnector(this);
    }

    // START SJSAS 6363251
    /**
     * Set the <code>Adapter</code> used by this connector.
     */
    @Override
    public void setHandler(HttpHandler handler){
        this.handler = handler;
    }
```

Concluding in this class coupling and cohesion are adequate and the code is free of duplicates, long methods or breaking encapsulation.

3.2 initialize()

In this method all variables start with a lower case and all the remaining words in the attribute name have their first letter capitalized.
All the method is indented with four spaces, without exceptions; no tabs are used for the indentation of the code. Moreover, the code of the method is written using the Khernighan and Ritchie style.
Line length never exceed 120 characters, but sometimes it exceed the 80 characters rules like in lines: 1404, 1429, 1432, 1443, 1448, 1467, 1543.
In the lines 1450 and 1468 like breaks occur after a comma or an operator.
However, the code is correctly commented and there is no use of "brutish programming"; there are no error in array indexing and all the objects are compared with .equals() instead of "==". All the displayed messages are error free and all the error messages are comprehensive. There are no precedence problems with the operator and code is free from any implicit type conversion.
Ending, all the relevant exceptions are caught and appropriate actions are taken for each catch block; all loops are correctly formed, with the appropriate initialization, increment and termination expressions.

3.3 translateAttributeName(String name)

In the implementation of this method, in order to indent text, the writer has used both spaces and tabulations. More in detail the first indentation (ifs, else ifs and last return) is done with tabulations, the second indentation (returns) is done by using 4 spaces. However the indentation is done consistently.
This method is written using the Kernighan and Ritchie style; line length does not exceed 80 characters and there are no line breaks.
In this method parameters are presented in the correct order and it returns a proper value, that is spelling errors free.
This method is private and this is the correct visibility that has to be assigned to it.
All objects are compared with equals and not with "==" .
By the using of multiple if statements "brutish" programming is avoided. The code is free from any implicit type conversions. All if clause have their return and there is a default return.

Summarizing we can say that the only issue encountered in this method is the use of tabulation in order to manage the first indentation. Here follows the code fragment that present the issue.

```
private String translateAttributeName(String name) {
        if ("clientAuth".equals(name)) {
            return "clientauth";
        } else if ("keystoreFile".equals(name)) {
            return "keystore";
        } else if ("randomFile".equals(name)) {
            return "randomfile";
        } else if ("rootFile".equals(name)) {
            return "rootfile";
        } else if ("keystorePass".equals(name)) {
            return "keypass";
        } else if ("keystoreType".equals(name)) {
            return "keytype";
        } else if ("sslProtocol".equals(name)) {
            return "protocol";
        } else if ("sslProtocols".equals(name)) {
            return "protocols";
        }
        return name;
}
```