GlassFish

Code Inspection

Monica Magoni 854091

Alberto Cibari 852689

January 21, 2016

Contents

1 Introduction

The scope of this document is to analyse some pieces of code extracted from a release of the *Glassfish 4.1* application server, in order to check if there are some issues by applying a code inspection checklist given by the assignment document[1].

1.1 Class Description

Our group has to inspect two methods of the class *AdminConsoleAuthModule*: the *initialize* and the *validateRequest* methods.
This class can be found in the class file under *appserver/admingui/common/src/main/java/org/glassfish/admingui/common/security/AdminConsoleAuthModule.java*
This class is responsible for providing the Authentication support needed by the admin console to both access the admin console pages as well as invoke REST requests.

1.1.1 *initialize* method

This method configures this authentication module and makes sure all the information needed to continue is present.
The function sets the callback handler of the class. It also checks if some options are passed. If there are options, it checks that these are not *null* and sets some options for the class, otherwise throws an exception.

1.1.2 *validateRequest* method

This method checks if a request is valid or not. If the request is not mandatory or the security check is not needed, the function returns always a 'SUCCESS' on the validation, otherwise it makes some checks, for example it checks if the Subject is marked as SAVED or if the username or password are null,

[1]see 'Bibliography' section

and returns a 'SUCCESS' or a 'FAILURE'. The last part of the method is used to make a REST request: then, if the response is successful, the RESt token, the Subject and the username are saved.

2 Issues

In this section, we list all the issues we have found in the code, referring to the assignment document. The order in which they are listed is the same as in the document we referred to. If a point is not present in our list is because we have not found problems concerning it.

2.1 General class problems

7. In the class, when the constant attributes are declared, there is the constant 'logger' (*private static final Logger logger = GuiUtil.getLogger();*) that is declared using all lowercase characters.

13. In the whole document there are lines of code that are more than 80 characters long.
    Here we list the lines that are in the class, but not in the methods we have to analyse:

    - Line *100* : declaration of *SUPPORTED_MESSAGE_TYPES* attribute
    - Line *326* : declaration of *secureResponse* method
    - Line *334* : declaration of *cleanSubject* method

14. We list the lines that are more than 120 characters long that are present in the previous point of the list: line *100*

18. Comments on code lines are adequate for describing the code. Javadoc comments, and class comments in general are poor written or absent. In general the code is very less commented.

19. Line *90*: variable declaration; this variable declaration is commented out but there is neither the reason why it has been commented out nor the date.

22. Since the class has no meaningful JavaDoc, it is very difficult to proceed with the analysis of the implementation of this class.

23. The JavaDoc is not complete. There is only a brief description of the class attributes and methods. Moreover, there are three methods where the Javadoc is completely absent: *secureResponse*, *cleanSubject* and *isMandatory*. Also, in all the methods of this class, the parameters and the return values are not described by the JavaDoc.

25. The static variables are declared after the instance variables. Also the static ones are declared in a non specific order. In this case *private* - *public* - *private*.

27. The class is formed of five methods. The method *cleanSubject* is empty and has a comment *//FIXME* inside. The relevant method in the class is *validateRequest* and is a very long method, which should be split into two methods.

2.2 *initialize* method

13. In the whole document there are lines of code that are more than 80 characters long.
    Here we list the lines that are in the method analyzed in this section:
    - Line *132*: the declaration of the method
    - Lines *138, 144, 154, 155*: strings concatenation
    - Lines *147, 149*: method concatenation

14. We list the lines that are more than 120 characters long that are also present in the previous point of the list: lines *132, 149, 155*

15. Lines *138, 139, 144, 145*: the '+' operator should be put in the previous line, in order to break the line after an operator.

18. There are few comments in the code. JavaDoc comments are poor written or absent. In general the code is very less commented.
    For the JavaDoc there are not the tags *'@param', '@throw'* to describe the parameters of the function.

23. View section 2.1, point '23' of the list.

33. In this method all the variable declarations and instantiations are done to create the value of a class variable at the end of the method. This behavior is still acceptable for the Java convention.

35. This function calls some deprecated methods, in particular the methods *getValue* and *putValue* of the class *StandardSession* and so these calls should be avoided.

2.3 *validateRequest* method

1. There are some variables such as '*rd*', '*ae*' and '*qs*' that have a name too short, so that it is not easy to understand their meaning. Although they are formed of two characters instead of one, they can be consider throwaway variables: indeed, they have only a temporary use. Their name is formed by the initial characters of their type. In particular '*rd*' stands for '*RequestDispatcher*', '*ae*' for '*AuthException*' and '*qs*' stands for '*QueryString*'.

5. On line *443* it is used a function called *basic(String, String)* and its name is not a verb. The syntax of each method is correct: the first letter of each addition word is capitalized.

13. In the whole document there are lines of code that are more than 80 characters long.
    Here we list the lines that are in the method analyzed in this section:
    - Line *171*: the declaration of the method;
    - Lines *191 to 193*: method concatenation;

  - Lines *241, 247, 248, 310*: variable declaration and instantiation;
  - Lines *214, 291*: 'if' statement;
  - Lines *289*: comment;

14. We list the lines that are more than 120 characters long that are also present in the previous point of the list: lines *141, 247*.

15. Line *178*: it should be better to put the AND operator (&&) in the previous line, in order to break the line after an operator.

17. Line *265*: there are four spaces that should be avoided.

18. Comments on code lines are adequate for describing the code. JavaDoc comments are poor written or absent. In general the code is very less commented.
    For the JavaDoc the tags *'@param', '@throw', '@return'* to describe the parameters of the function are absent.

19. Line from *233* to *237* is code commented out. There is the reason why it has been commented out, but it does not contain the date.

23. View section 2.1, point '23' of the list.

27. This is a very long method, more than 150 lines of code, and it should be divided into sub methods.

33. In this function the major part of the local variables are declared and instantiated just before a compound statement.

50. The code that can cause exceptions is handled by the *try-catch* statement. It is correct but can be improved. In fact the functions can throw more than one type of exception, but these exceptions are handled only by a generic catch statement (*catch (Exception ex)*). It would be better if there were multiple catch statements, one for each exception type.

Appendix

Tools used

- NetBeans: to navigate through the project of Glassfish and do some checks. [2]
- GitHub: to share our work. [3]
- ShareLaTeX: to write this document in LaTeX. [4]

Sites

- *http://yohanan.org/steve/projects/java-code-conventions/* for better specifications of the Java conventions.
- *http://users.csc.calpoly.edu/ jdalbey/SWE/CodeSmells/bonehead.html* for checking some brute force programming methods to avoid.

Hours of work

- Alberto Cibari: 10 hours
- Monica Magoni: 8 hours

[2]`https://netbeans.org/`
[3]`https://github.com`
[4]`https://it.sharelatex.com`

Bibliography

[1] Software Engineering 2 project, *AA 2015-2016 Software Engineering 2, Assignment 3: code inspection*

[2] Software Engineering 2 project , *AA 2015-2016 Software Engineering 2, Project goal, schedule and rules*

[3] Java code conventions, `http://yohanan.org/steve/projects/java-code-conventions/`

[4] Brutish programming, `http://users.csc.calpoly.edu/~jdalbey/SWE/CodeSmells/bonehead.html`