Politecnico di Milano
A.Y. 2015/2016

Glassfish 4.1

Code Inspection

Bernardis Cesare matr. 852509 Dagrada Mattia matr.852975

January 5, 2016

Contents

1 Checklist **3**
- 1.1 Naming Conventions . . . 3
- 1.2 Indention . . . 3
- 1.3 Braces . . . 3
- 1.4 File Organization . . . 4
- 1.5 Wrapping Lines . . . 4
- 1.6 Comments . . . 4
- 1.7 Java Source Files . . . 4
- 1.8 Package and Import Statements . . . 4
- 1.9 Class and Interface Declarations . . . 5
- 1.10 Initialization and Declarations . . . 5
- 1.11 Method Calls . . . 6
- 1.12 Arrays . . . 6
- 1.13 Object Comparison . . . 6
- 1.14 Output Format . . . 6
- 1.15 Computation, Comparisons and Assignments . . . 6
- 1.16 Exceptions . . . 7
- 1.17 Flow of Control . . . 7
- 1.18 Files . . . 7

2 Classes **8**
- 2.1 ApplicationHandlers . . . 8
  - 2.1.1 Details . . . 8
  - 2.1.2 Issues . . . 8

3 Assigned Class clusters **9**
- 3.1 Introduction . . . 9
- 3.2 getLaunchInfo . . . 9
  - 3.2.1 Description . . . 9
  - 3.2.2 Code . . . 10
  - 3.2.3 Issues . . . 10
- 3.3 if(appPropsMap != null) . . . 12
  - 3.3.1 Description . . . 12
  - 3.3.2 Code . . . 12
  - 3.3.3 Issues . . . 12
- 3.4 calContextRoot(String contextRoot) . . . 14
  - 3.4.1 Description . . . 14
  - 3.4.2 Code . . . 15
  - 3.4.3 Issues . . . 15
- 3.5 getURLs(List<String> vsList, String configName, Collection<String> hostNames, String target) . . . 16

3.5.1 Description 16
3.5.2 Code 17
3.5.3 Issues 18

4 Additional Material **19**

1 Checklist

1.1 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 throwaway 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;

1.2 Indention

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

9. No tabs are used to indent

1.3 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).

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

1.4 File Organization

12. Blank lines and optional comments are used to separate sections (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.

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

1.5 Wrapping Lines

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

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.

1.6 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.

1.7 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).

1.8 Package and Import Statements

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

1.9 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.

1.10 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.

1.11 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

1.12 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

1.13 Object Comparison

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

1.14 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

1.15 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

1.16 Exceptions

52. Check that the relevant exceptions are caught

53. Check that the appropriate action are taken for each catch block

1.17 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

1.18 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

2 Classes

2.1 ApplicationHandlers

2.1.1 Details

Name : ApplicationHandlers

Location : appserver/admingui/common/src/main/java/org/glassfish/admingui/common/handlers

Package : org.glassfish.admingui.common.handlers

Modifiers : public class

2.1.2 Issues

18 There are only a few comments in the code of the whole class, not enough, given the absence of a proper javadoc.

23 Javadoc is very poor, it is almost totally missing.

27 There is a method (getURLs at line 760) that is longer than 50 lines.

3 Assigned Class clusters

3.1 Introduction

The parts processed in this document belong to the same class ApplicationHandlers. In this class the javadoc is almost completely absent and the comments are rare and ineffective. The comprehension of the functional roles has been very difficult to deal with, because it has been forced to the pure code interpretation. Fortunately a good naming convention has been respected, making the understanding a little "less impossible", but yet very superficial.

3.2 getLaunchInfo

3.2.1 Description

Location : appserver/admingui/common/src/main/java/org/glassfish/admingui/common/handlers/ApplicationHandlers.java

Name : getLaunchInfo(String appName, String contextRoot, Map oneRow)

Modifiers : private static void

Start Line : 368

Functional Role : Retrieves some informations about an application (the name of which is passed as a parameter). It makes some rest requests and parses the responses to set, in particular, two values in the Map passed as parameter: 'contextRoot' (if not passed as parameter) and 'hasLaunch'.

3.2.2 Code

```
368     private static void getLaunchInfo(String appName, String contextRoot, Map oneRow) {
369         String endpoint = GuiUtil.getSessionValue("REST_URL") + "/applications/application/" + appName + ".json";
370         Map map = RestUtil.restRequest(endpoint, null, "GET", null, false);
371         Map data = (Map)map.get("data");
372         boolean enabled = false;
373         if (data != null) {
374             Map extraProperties = (Map)data.get("extraProperties");
375             if (extraProperties != null) {
376                 Map entity = (Map)extraProperties.get("entity");
377                 if (entity != null) {
378                     if (contextRoot == null)
379                         contextRoot = (String) entity.get("contextRoot");
380                     enabled = Boolean.parseBoolean((String) entity.get("enabled"));
381                 }
382             }
383         }
384         oneRow.put("contextRoot", (contextRoot==null)? "" : contextRoot);
385         oneRow.put("hasLaunch", false);
386 
387         if ( !enabled || GuiUtil.isEmpty(contextRoot)){
388             return ;
389         }
390 
391         List<String> targetList = DeployUtil.getApplicationTarget(appName, "application-ref");
392         for(String target : targetList) {
393             String virtualServers = getVirtualServers(target, appName);
394             String ep = TargetUtil.getTargetEndpoint(target) + "/application-ref/" + appName;
395             enabled = Boolean.parseBoolean((String)RestUtil.getAttributesMap(ep).get("enabled"));
396             if (!enabled)
397                 continue;
398             if (virtualServers != null && virtualServers.length() > 0) {
399                 oneRow.put("hasLaunch", true);
400             }
401         }
402     }
```

3.2.3 Issues

11 One statement statements are not surrounded by curly braces

- If at line 378
- If at line 396

13 Multiple lines exceed the 80 characters (spaces included). This is not a major issue, because none of them exceeds the limit of 120 characters on the same line and the readability is not compromised.

- Line 369 could be divided into segments after '+' operator.
- Line 380
- Line 391
- Line 394 could be divided into segments after '+' operator.
- Line 395

18 This is a private method, so it is comprehensible that no javadoc has been provided, but without comments it is quite hard to understand what the method does.

33 This will NOT be considered an issue: at line 396 there is a declaration and initialization that is not at the beginning of a "physical" block, but it can be considered an initialization for the loop at line 397 (in order to make it easier to understand).

52 There are no exceptions caught or thrown explicitly. There are extra controls to avoid method calls on null objects and some other measures to avoid execution errors, but they might not be enough.

3.3 if(appPropsMap != null)

3.3.1 Description

Location : appserver/admingui/common/src/main/java/org/glassfish/admingui/common/handlers/ApplicationHandlers.java

Name : if(appPropsMap != null)

Start Line : 617

Functional Role : For every value in appPropsMap (which is a Map containing, probably, the properties of an application) retrieves some informations and puts them into a Map that will be added to the result object.

3.3.2 Code

```
617     if (appPropsMap != null) {
618         for(Map.Entry<String,String> e : appPropsMap.entrySet()){
619             try{
620                 String engines = e.getValue();
621                 HashMap oneRow = new HashMap();
622                 oneRow.put("name", e.getKey());
623                 String encodedName = URLEncoder.encode(e.getKey(), "UTF-8");
624                 oneRow.put("targetName", target);
625                 oneRow.put("selected", false);
626                 String endpoint = prefix  + appRefEndpoint + encodedName;
627                 oneRow.put("endpoint", endpoint);
628                 Map appRefAttrsMap = RestUtil.getAttributesMap(endpoint);
629                 String image = (appRefAttrsMap.get("enabled").equals("true")) ?  "/resource/images/enabled.png" : "/resource/images/disabled.png";
630                 oneRow.put("enabled", image);
631                 image = (appRefAttrsMap.get("lbEnabled").equals("true")) ?  "/resource/images/enabled.png" : "/resource/images/disabled.png";
632                 oneRow.put("lbEnabled",  image);
633                 oneRow.put("sniffers", engines);
634                 List sniffersList = GuiUtil.parseStringList(engines, ",");
635                 oneRow.put("sniffersList", sniffersList);
636                 for(int ix=0; ix< sniffersList.size(); ix++)
637                     filters.add(sniffersList.get(ix));
638                 if (filterValue != null){
639                     if (! sniffersList.contains(filterValue))
640                         continue;
641                 }
642                 result.add(oneRow);
643             }catch(Exception ex){
644                 //skip this app.
645             }
646         }
647     }
```

3.3.3 Issues

9 A tab is used to indent line 617

11 One statement statements are not surrounded by curly braces

- For at line 636
- If at line 639

14 Two lines exceed the 120 characters (spaces included).

- Line 629 (151 chars) could be divided into '?:' operator segments.
- Line 631 (146 chars) could be divided into '?:' operator segments.

17 There is a mistake in the indentation: the wrapping If is not aligned with the upper statement.

18 In this piece of code (that is a particular part of a method) there are no sensible comments. Unfortunately also the javadoc is incomplete and is not clear the role of this public method.

23 The javadoc for this method is incomplete, practically it is totally missing.

33 In the For at line 618 there are several declarations and initializations mixed with "put" operations in the output row.

53 There is a single try-catch block for the whole piece of code. It is not very elegant, but it avoids errors. The problem is that the general exception is catched, but no actions are properly taken.

3.4 calContextRoot(String contextRoot)

3.4.1 Description

Location : appserver/admingui/common/src/main/java/org/glassfish/admingui/common/handlers/ApplicationHandlers.java

Name : calContextRoot(String contextRoot)

Start Line : 741

Functional Role : This method is used to correctly calculate the context root given one as paramenter that may not be complete or well formed. This is evident from the code and from the comment.

3.4.2 Code

```
741    private static String calContextRoot(String contextRoot) {
742        //If context root is not specified or if the context root is "/", ensure that we don't show two // at the end.
743        //refer to issue#2853
744        String ctxRoot = "";
745        if ((contextRoot == null) || contextRoot.equals("") || contextRoot.equals("/")) {
746            ctxRoot = "/";
747        } else if (contextRoot.startsWith("/")) {
748            ctxRoot = contextRoot;
749        } else {
750            ctxRoot = "/" + contextRoot;
751        }
752        return ctxRoot;
753    }
```

3.4.3 Issues

13 Line 745 exceeds the 80 characters.

23 The javadoc for this method is totally missing.

52 There are no exceptions caught or thrown explicitly but there is a control on null object. This should be enough since this is a simple method.

3.5 getURLs(List<String> vsList, String configName, Collection<String> hostNames, String target)

3.5.1 Description

Location : appserver/admingui/common/src/main/java/org/glassfish/admingui/common/handlers/ApplicationHandlers.java

Name : getURLs(List<String> vsList, String configName, Collection<String> hostNames, String target)

Start Line : 760

Functional Role : This method is used to get the list of URLs starting from a list of virtual servers. Starting from the server it creates a configuration string that is used to recover the configuration of virtual servers. It then recovers the local host name and for each virtual server it which is not null, it computes again the configuration string and from which it tries to recover a list of listeners. In case the list is not empty it computes every single different listener. Finally it calculate the type of protocol, security and port for every listener, computing the URL and adding it to the list to be returned.

3.5.2 Code

```
760     private static Set getURLs(List<String> vsList, String configName, Collection<String> hostNames, String target) {
761         Set URLs = new TreeSet();
762         if (vsList == null || vsList.size() == 0) {
763             return URLs;
764         }
765         //Just to ensure we look at "server" first.
766         if (vsList.contains("server")){
767             vsList.remove("server");
768             vsList.add(0, "server");
769         }
770         String ep = (String)GuiUtil.getSessionValue("REST_URL");
771         ep = ep + "/configs/config/" + configName + "/http-service/virtual-server";
772         Map vsInConfig = new HashMap();
773         try{
774             vsInConfig = RestUtil.getChildMap(ep);
775 
776         }catch (Exception ex){
777             GuiUtil.getLogger().info(GuiUtil.getCommonMessage("log.error.getURLs") + ex.getLocalizedMessage());
778                 if (GuiUtil.getLogger().isLoggable(Level.FINE)){
779                     ex.printStackTrace();
780                 }
781         }
782         String localHostName = null;
783         try {
784             localHostName = InetAddress.getLocalHost().getHostName();
785         } catch (Exception ex) {
786             // ignore exception
787         }
788 
789         for (String vsName : vsList) {
790             if (vsName.equals("__asadmin")) {
791                 continue;
792             }
793             Object vs = vsInConfig.get(vsName);
794             if (vs != null) {
795                 ep = (String)GuiUtil.getSessionValue("REST_URL") + "/configs/config/" +
796                         configName + "/http-service/virtual-server/" + vsName;
797                 String listener = (String)RestUtil.getAttributesMap(ep).get("networkListeners");
798 
799                 if (GuiUtil.isEmpty(listener)) {
800                     continue;
801                 } else {
802                     List<String> hpList = GuiUtil.parseStringList(listener, ",");
803                     for (String one : hpList) {
804                         ep = (String)GuiUtil.getSessionValue("REST_URL") +
805 "/configs/config/" + configName + "/network-config/network-listeners/network-listener/" + one;
806 
807                         Map nlAttributes = RestUtil.getAttributesMap(ep);
808                         if ("false".equals((String)nlAttributes.get("enabled"))) {
809                             continue;
810                         }
811 //                          String security = (String)oneListener.findProtocol().attributesMap().get("SecurityEnabled");
812                         ep = (String)GuiUtil.getSessionValue("REST_URL") + "/configs/config/" +
813                                 configName + "/network-config/protocols/protocol/" + (String)nlAttributes.get("protocol");
814                         String security = (String)RestUtil.getAttributesMap(ep).get("securityEnabled");
815 
816                         String protocol = "http";
817                         if ("true".equals(security))
818                             protocol = "https";
819 
820                         String port = (String)nlAttributes.get("port");
821                         if (port == null)
822                             port = "";
823                         String resolvedPort = RestUtil.resolveToken((String)GuiUtil.getSessionValue("REST_URL") +
824                                 "/servers/server/" + target, port);
825 
826                         for (String hostName : hostNames) {
827                             if (localHostName != null && hostName.equalsIgnoreCase("localhost"))
828                                 hostName = localHostName;
829 //                              URLs.add("[" + target + "]  - " + protocol + "://" + hostName + ":" + resolvedPort + "[ " + one + " " + configName
830 //                                      + " " + listener + " " + target + " ]");
831                             URLs.add(target + "@@@" + protocol + "://" + hostName + ":" + resolvedPort);
832                         }
833                     }
834                 }
835             }
836         }
837         return URLs;
838     }
839 }
```

3.5.3 Issues

2 There two variables named "ep" and "vs" which are used throughout the entire method but their names don't help at all understanding what they mean.

8 Line 805 is not intedented properly, actually it is not intented at all.

13 A lot of lines exceed the 80 characters. This doesn't compromise readability too much but the method looks really messy.

- Line 771 (84 chars)
- Line 777 (112 chars)
- Line 795 (88 chars)
- Line 797 (97 chars)
- Line 802 (82 chars)
- Line 808 (83 chars)
- Line 812 (96 chars)
- Line 805 (95 chars)
- Line 814 (104 chars)
- Line 823 (114 chars)
- Line 827 (97 chars)
- Line 831 (105 chars)

14 Line 813 exceeds 120 characters (123 character). This could be prevented using another high-level break.

19 There is commented code with no reasons to be so, nor a date on which it could be removed.

- Line 811
- Line 829 and 830

23 The javadoc for this method is totally missing.

52 There are two try-catch blocks but are used for generic exception, it works fine but a more specific exception could be better. A lot of code is instead lacking try-catch blocks but there are few controls in If statements. Even tho it is probably enough, other blocks could be useful.

53 There is an exception caught at line 785 which action is non existent, there is only a comment saying "ignore exception". This may not be totally wrong if the comment said also why the exception can be ignored. Anyway it is always better to do something in the catch block, even sending an error messagge.

4 Additional Material

In order to write this document we used:

- Javadoc available at http://glassfish.pompel.me
- Class clusters available at http://assignment.pompel.me
- PDF document with the assignment
- Glassfish code available at https://svn.java.net/svn/glassfish svn/tags/4.1.1