

Code Inspection Document

Luca Nanni (850113) Giacomo Servadei (854819)

December 29, 2015

Contents

1	Classes and methods	1
1.1	Location	1
1.2	Namespace	2
1.3	Class name	2
1.4	Analyzed methods	2
2	Functional role of the classes	2
3	Issues found by applying the checklist	2
3.1	Naming Conventions	2
3.2	Indention	3
3.3	Braces	3
3.4	File organization	4
3.5	Wrapping Lines	5
3.6	Comments	6
3.7	Java Source Files	6
3.8	Package import statements	7
3.9	Class and Interface Declarations	7
3.10	Initialization and Declarations	7
3.11	Method Calls	8
3.12	Arrays	8
3.13	Object Comparisons	9
3.14	Output format	9
3.15	Computation, Comparisons and Assignments	9
3.16	Exceptions	10
3.17	Flow of control	10
3.18	Files	11
4	Other problems	11

1 Classes and methods

1.1 Location

appserver/ejb/ejb-container/src/main/java/
/org/glassfish/ejb/deployment/BeanMethodCalculatorImpl.java

1.2 Namespace

org.glassfish.ejb.deployment

1.3 Class name

BeanMethodCalculatorImpl

1.4 Analyzed methods

- **Method 1:** *getMethodsFor*(com.sun.enterprise.deployment.EjbDescriptor ejbDescriptor, ClassLoader classLoader)
- **Method 2:** *getTransactionalMethodsFor*(com.sun.enterprise.deployment.EjbDescriptor desc, ClassLoader loader)

2 Functional role of the classes

3 Issues found by applying the checklist

We use the following notation:

- ✓: the relative point in the checklist is satisfied by the method
- ✗: the relative point in the checklist is not satisfied and will follow the piece of code affected by the problem or a description of the problem

3.1 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:
 - Method 1: ✓
 - Method 2: ✓
2. If one-character variables are used, they are used only for temporary “throwaway” variables, such as those used in for loops.
 - Method 1: ✓
 - Method 2: ✓
3. Class names are nouns, in mixed case, with the first letter of each word in capitalized.
 - Method 1: ✓
 - Method 2: ✓
4. Interface names should be capitalized like classes
 - Method 1: ✓
 - Method 2: ✓
5. Method names should be verbs, with the first letter of each addition word capitalized.

- Method 1: ✓
 - Method 2: ✓
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
- Method 1: ✓
 - Method 2: ✓
7. Constants are declared using all uppercase with words separated by an underscore
- Method 1: ✓
 - Method 2: ✓

3.2 Indention

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

- Method 1: ✓
 - Method 2: ✗
Line 169: used TAB instead of spaces
- ```
169 // Session Beans
```

9. No tabs are used to indent:

- Method 1: ✓
  - Method 2: ✗  
Line 169: used TAB instead of spaces
- ```
169          // Session Beans
```

3.3 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) :

- Method 1: ✗
The bracing style used is not consistent: in the method declaration the first brace is underneath the opening block, whereas in the rest of the method is in the same line.

```
106    public Vector getMethodsFor (com.sun.enterprise.deployment.EjbDescriptor
107        ejbDescriptor, ClassLoader classLoader)
108        {
109        Vector methods = new Vector();
```

- Method 2: ✗
The bracing style used is not consistent: in the method declaration the first brace is underneath the opening block, whereas in the rest of the method is in the same line.

```

153     public Collection getTransactionalMethodsFor(com.sun.enterprise.deployment.
        EjbDescriptor desc, ClassLoader loader)
154     throws ClassNotFoundException, NoSuchMethodException
155     {
156         EjbDescriptor ejbDescriptor = (EjbDescriptor) desc;

```

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

- Method 1: ✓
- Method 2: ✓

3.4 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) :

- Method 1: ✓
- Method 2: ✓

13. Where practical, line length does not exceed 80 characters:

- Method 1: ✗
Often in the code, lines exceed 80 characters.

```

111     if (ejbDescriptor.isRemoteInterfacesSupported()) {
112         addAllInterfaceMethodsIn(methods, classLoader.loadClass(ejbDescriptor.
            getHomeClassName()));
113         addAllInterfaceMethodsIn(methods, classLoader.loadClass(ejbDescriptor.
            getRemoteClassName()));
114     }

```

- Method 2: ✗
Often in the code, lines exceed 80 characters.

```

171         Collection disallowedMethods = extractDisallowedMethodsFor(javax.ejb.
            EJBObject.class, sessionBeanMethodsDisallowed);
172         Collection potentials = getTransactionMethodsFor(loader,
            ejbDescriptor.getRemoteClassName() , disallowedMethods);

```

**

```

191         Collection disallowedMethods = extractDisallowedMethodsFor(javax.ejb.
            EJBLocalObject.class, sessionLocalBeanMethodsDisallowed);
192         Collection potentials = getTransactionMethodsFor(loader,
            ejbDescriptor.getLocalClassName() , disallowedMethods);

```

**

```

235         Set<LifecycleCallbackDescriptor> lcds = ejbDescriptor.
            getLifecycleCallbackDescriptors();

```

**

```

239         MethodDescriptor md = new MethodDescriptor(m, MethodDescriptor.
            LIFECYCLE_CALLBACK);

```

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

- Method 1: ✗

There is one line (ln.138) exceeding even 120 characters.

```
137     if (ejbDescriptor.hasWebServiceEndpointInterface()) {
138         addAllInterfaceMethodsIn(methods, classLoader.loadClass(ejbDescriptor.
            getWebServiceEndpointInterfaceName()));
139     }
140 }
```

- Method 2: ✗

See the point [13](#)

3.5 Wrapping Lines

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

- Method 1: ✗

This never happens. Not even in the method declaration.

```
106     public Vector getMethodsFor(com.sun.enterprise.deployment.EjbDescriptor
        ejbDescriptor, ClassLoader classLoader)
```

- Method 2: ✗

```
153     public Collection getTransactionalMethodsFor(com.sun.enterprise.deployment.
        EjbDescriptor desc, ClassLoader loader)
154     throws ClassNotFoundException, NoSuchMethodException
```

**

```
184         methods.add(new MethodDescriptor
185             (next, MethodDescriptor.EJB_REMOTE));
```

**

```
205         methods.add(new MethodDescriptor
206             (next, MethodDescriptor.EJB_LOCAL));
```

**

```
216         methods.add(new MethodDescriptor
217             (next, MethodDescriptor.EJB_LOCAL));
```

16. Higher-level breaks are used: ***** NOT SURE ABOUT THIS ONE *****

- Method 1:

- Method 2: ✗

```
227         methods.add(new MethodDescriptor(webMethods[i],
228             MethodDescriptor.EJB_WEB_SERVICE));
```

**

```

243         _logger.log(Level.FINE,
244             "Lifecycle_callback_processing_error", e);

```

17. A new statement is aligned with the beginning of the expression at the same level as the previous line:

- Method 1: ✓
- Method 2: ✓

3.6 Comments

18. Comments are used to adequately explain what the class, interface, methods, and blocks of code are doing.

- Class: ✗
Some methods do not present any comment above and it is necessary to make a reverse engineering of the code in order to understand what they do.

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.

- Method 1: ✓
- Method 2: ✓

3.7 Java Source Files

20. Each Java source file contains a single public class or interface.

- Method 1: ✓
- Method 2:

21. The public class is the first class or interface in the file.

- Method 1: ✓
- Method 2:

22. Check that the external program interfaces are implemented consistently with what is described in the javadoc.

- Method 1: ✓
- Method 2:

23. Check that the javadoc is complete

- Method 1: ✗
The Javadoc is not complete: it does not explain what this method is for and does not describe the kind and the role of the output of this method.
- Method 2:

3.8 Package import statements

24. If any package statements are needed, they should be the first noncomment statements. Import statements follow.
- Method 1: ✓
 - Method 2:

3.9 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
 - a. first public class variables
 - b. next protected class variables
 - c. next package level (no access modifier)
 - d. last private class variables
 - E. instance variables
 - a. first public instance variables
 - b. next protected instance variables
 - c. next package level (no access modifier)
 - d. last private instance variables
 - F. constructors
 - G. methods
- Class: ✓
26. Methods are grouped by functionality rather than by scope or accessibility:
- Class: ✓
27. Check that the code is free of duplicates, long methods, big classes, breaking encapsulation, as well as if coupling and cohesion are adequate:
- Class: ✓

3.10 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)
- Method 1: ✓
 - Method 2:
29. Check that variables are declared in the proper scope

- Method 1: ✓
 - Method 2:
30. Check that constructors are called when a new object is desired
- Method 1: ✓
 - Method 2:
31. Check that all object references are initialized before use
- Method 1: ✓
 - Method 2:
32. Variables are initialized where they are declared, unless dependent upon a computation
- Method 1: ✓
 - Method 2:
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
- Method 1: ✓
 - Method 2:

3.11 Method Calls

34. Check that parameters are presented in the correct order :
- Method 1: ✓
 - Method 2: ✓
35. Check that the correct method is being called, or should it be a different method with a similar name:
- Method 1: ✓
 - Method 2: ✓
36. Check that method returned values are used properly:
- Method 1: ✓
 - Method 2: ✓

3.12 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):
- Method 1: ✓
 - Method 2: ✓
38. Check that all array (or other collection) indexes have been prevented from going out-of-bounds:

- Method 1: ✓
- Method 2: ✓

39. Check that constructors are called when a new array item is desired:

- Method 1: ✓
- Method 2: ✓

3.13 Object Comparisons

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

- Method 1: ✓
- Method 2:

3.14 Output format

41. Check that displayed output is free of spelling and grammatical errors:

- Method 1: ✓
- Method 2:

42. Check that error messages are comprehensive and provide guidance as to how to correct the problem:

- Method 1: ✓
- Method 2:

43. Check that the output is formatted correctly in terms of line stepping and spacing:

- Method 1: ✓
- Method 2:

3.15 Computation, Comparisons and Assignments

44. Check that the implementation avoids 'brutish programming':

- Method 1: ✓
- Method 2:

45. Check order of computation/evaluation, operator precedence and parenthesizing:

- Method 1: ✓
- Method 2:

46. Check the liberal use of parenthesis is used to avoid operator precedence problems:

- Method 1: ✓
- Method 2:

47. Check that all denominators of a division are prevented from being zero:

- Method 1: ✓
 - Method 2:
48. Check that integer arithmetic, especially division, are used appropriately to avoid causing unexpected truncation/rounding:
- Method 1: ✓
 - Method 2:
49. Check that the comparison and Boolean operators are correct:
- Method 1: ✓
 - Method 2:
50. Check throw-catch expressions, and check that the error condition is actually legitimate:
- Method 1: ✓
 - Method 2:
51. Check that the code is free of any implicit type conversions: ***** NOT SURE

- Method 1:
 - Method 2:

3.16 Exceptions

52. Check that the relevant exceptions are caught
- Method 1: ✓
 - Method 2:
53. Check that the appropriate action are taken for each catch block
- Method 1: ✓
 - Method 2:

3.17 Flow of control

54. In a switch statement, check that all cases are addressed by break or return
- Method 1: ✓
 - Method 2:
55. Check that all switch statements have a default branch
- Method 1: ✓
 - Method 2:
56. Check that all loops are correctly formed, with the appropriate initialization, increment and termination expressions
- Method 1: ✓
 - Method 2:

3.18 Files

57. Check that all files are properly declared and opened
 - Method 1: ✓
 - Method 2:
58. Check that all files are closed properly, even in the case of an error
 - Method 1: ✓
 - Method 2:
59. Check that EOF conditions are detected and handled correctly
 - Method 1: ✓
 - Method 2:
60. Check that all file exceptions are caught and dealt with accordingly
 - Method 1: ✓
 - Method 2:

4 Other problems