Politecnico Di Milano
A.A. 2015/2016
Software Engineering 2: Inspection Document

Paramithiotti Andrea (Matr. 788794)
Rompani Andrea (Matr. 854052)
Zoia Lorenzo (Matr. 852392)

v1.0
5/1/2016

1 Contents

2 Assigned Methods

All the methods assigned to the group belong to the class **DOLUtils**, in the package **com.sun.enterprise.deployment.util.**

1. readAlternativeRuntimeDescriptor()
2. readRuntimeDeploymentDescriptor()
3. getSniffersFromModule()
4. getTypeFromModuleType()
5. getConfigurationDeploymentDescriptorFiles()
6. setElementValue()

3 Functional Role of the Methods

The Javadoc for the classes and the methods assigned was almost non-existent, thus the functional role of the methods was understood only by analysing the code line by line, trying to extrapolate the meaning of the actions performed also by reading the code of other classes as well.

3.1 readAlternativeRuntimeDescriptor

3.1.1 Parameters

- ReadableArchive appArchive
- ReadableArchive embeddedArchive
- Archivist archivist
- BundleDescriptor descriptor
- String altDDPath

3.1.2 Functional Role

The Archivist contains the info for the deployment of determined modules. If the deployment method exists, then it is the one that is used. Otherwise this method is called, which after some control on the parameters takes an alternative descriptor from the parameters at runtime and saves it in the Archivist.

3.2 readRuntimeDeploymentDescriptor

3.2.1 Parameters

- List<ConfigurationDeploymentDescriptorFile> confDDFiles,
- ReadableArchive archive
- RootDeploymentDescriptor descriptor
- Archivist main
- final boolean warnIfMultipleDDs

3.2.2 Functional Role

In the case that the parameter warnIfMultipleDDs is true, the method writes a warning on the depLogger for each element in the list confDDFiles. Then it takes the first ConfigurationDDFile in the list and sets its XML validation and its level of validation to true if confDD.isValidating() is true, otherwise it is set to false. The validation is taken from the Archivist that is passed as a parameter. In the end it calls the method Read() on the Descriptor passed as a parameter and on the InputStream that is taken from the Archivist passed as a parameter.

3.3 getSniffersFromModule

3.3.1 Parameters

- ServiceLocator habitat
- ReadableArchive archive,
- ModuleDescriptor md
- Application app

3.3.2 Functional Role

The method saves in the Archive passed as a parameter the list of the sniffers that are compatible with the module passed as a parameter, then returns it.

3.4 getTypeFromModuleType

3.4.1 Parameters

- ArchiveType moduleType

3.4.2 Functional Role

The method returns a String that defines the type of the document passed as a parameter. It makes the mapping between a string and a moduleType

3.5 getConfigurationDeploymentDescriptorFiles

3.5.1 Parameters

- ServiceLocator habitat
- String containerType

3.5.2 Functional Role

The method returns the configuration files useful for the deployment descriptors. In the list that is returned it saves only the elements for whom the indexedType of the descriptor of each handles of the habitat is equals to the type passed as a parameter.

3.6 setElementValue

3.6.1 Parameters

- XMLElement element
- String value
- Object o

3.6.2 Functional Role

Regardless of what the method does, it is never called by any function in all the project, so it is a useless dead part of the code. Sometimes another method similar to this, but with different parameters, is called in the project

4 Checklist

Here is the checklist that was followed to edit this document. Some of the possible issues referred to the entire class or even to the entire project, so it was impossible to check if those issues were present or not.

4.1 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; (ci sono errori)

4.2 Indention

8. Three or four spaces are used for indentation and done so consistently
9. No tabs are used to indent

4.3 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. Example: Avoid this:

```
if ( condition )
    doThis();
```

Instead do this:
```
if ( condition )
{
    doThis();
}
```

4.4 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.

4.5 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.

4.6 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.

4.7 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).

4.8 Package and Import Statements

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

4.9 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

        e. next protected instance variables

        f. next package level (no access modifier)

        g. 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.

4.10   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.

4.11　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

4.12　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

4.13　Object Comparison

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

4.14　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

4.15　Computation, Comparisons and Assignments

44. Check that the implementation avoids "brutish programming
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

4.16   Exceptions

52. Check that the relevant exceptions are caught
53. Check that the appropriate action are taken for each catch block

4.17   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

4.18   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

5 List of Issues

5.1 Class Issues

	Respected	Lines	Code	Possible solution
3	✔			
4	✔			
6	✔			
7	✖	132	Constant is written in lower case letters	
12	✖	127-130, 150-153, 155-157	these class attributes are divided by optional comments	
		313-315, 319-320, 324-38, 402-406, 423-425, 538-539, 616-618	these class methods are divided by optional comments	All the optional comments should be replaced by Javadoc comments
20	✔			
21	✔			
22	✔			
23	✖		It cover only the class declaration and some public methods	
24	✔			
25	✖		Subpoint D, subpoint a, lines 132,145 should be moved upper Subpoint D, subpoint d, NO	
26	✖		• setElementValue should go below • getTypeForModuleType should go with the other methods that handle "Type" • getConfigurationDeploymentDescriptorFile should go with the other methods that handle "DD"	
27	✔			
28	✔			
36	✔		we don't know because there is no javadoc in the methods that are called and our methods are not called in the code assigned to us	

5.2 readAlternativeRuntimeDescriptor()

	Respected	Lines	Code	Possible solution
1	✔			
2	✔			
3	✔			
5	✔			
7	✔			
8	✔			
9	✔			
10	✔		Kernighan and Ritchie style	
11	✔			
12	✔			
13	✖	425	Method declaration can be split on multiple lines	
		429	But cannot be split on multiple lines	
14	✖	425 430	These 4 lines exceed the 120 characters	
15	✖	451-452	No need to split the row	
		453-454	No need to split the row	
		441-442	The && operator must be on line 441	
16	✔			
17	✔			
18	✖	423-424	Presence of a comment that isn't in the Javadoc format but explains what the method does	
		434	Presence of a TODO	
19	✔			
29	✔			
30	✔			
31	✔			
32	✔			
33	✔			
34	✔		It respects the order of parameters used for the other methods	
35	✔			
37	✔			
38	✔			
39	✔			
40	✔			
41	❓		No output	
42	❓		No output	
43	❓		No output	
44	✔			
45	✔			
46	✔			
47	✔			
48	✔			
49	✔			
50	✔			
51	✔			

52	✖	450-461	`InputStream is = appArchive.getEntry(altRuntimeDDPath);`
			`confDD.setXMLValidation(`
			`    archivist.getRuntimeXMLValidation());`
			`confDD.setXMLValidationLevel(`
			`    archivist.getRuntimeXMLValidationLevel());`
			`if (appArchive.getURI()!=null) {`
			`    confDD.setErrorReportingString(`
			`        appArchive.getURI().getSchemeSpecificPart());`
			`}`
			
			`confDD.read(descriptor, is);`
			`is.close();`
			
			There is no try-catch block for IOException like a similar code that executes the same confDD.read() method on a different part of the class file
				Insert a try-catch-finally block

53	✔			
54	✔			
55	✔			
56	✔			
57	✔			
58	✔			
59	✔			
60	✔			

5.3 readRuntimeDeploymentDescriptor()

	Respected	Lines	Code	Possible solution
1	✖	480	Archivist main	Rename the parameter
2	✔			
3	✔			
5	✔			
7	✔			

8	✖	509-511
		512
			``if (confDD.isValidating()) {
			    confDD.setXMLValidation(main.getRu
			    confDD.setXMLValidationLevel(main.
			} else {
			    confDD.setXMLValidation(false);
			}```	

		502-504	```new Object[] {
			    confDDFiles.get(i).getDeploymentDescriptorPath(),
			    archive.getURI().getSchemeSpecificPart(),
			    confDD.getDeploymentDescriptorPath()});```	

9	✔			
10	✔		Kernighan and Ritchie style	
11	✔			
12	✔			
13	✖	480	Method declaration can be split on multiple lines	
		489	If statement condition can be split on multiple lines	
14	✔	480 489 502 507	These 4 lines not exceed the 120 characters	
15	✔			
16	✔			
17	✔			
18	✔			
19	✔			
29	✔			
30	✔			
31	✔			
32	✔			
33	✖	485	`InputStream is = null;`	This declaration must be at the beginning of the block
34	✔		It respects the order of parameters used for the other methods	
35	✔			
37	✔			
38	✔		If confDDFiles is an empty list this method will exit at the start Also if confDDFiles.size() returns a value of 1 or less the for cycle of row 497-506 won't be executed	
39	✔			
40	✔			
41	?		No output	

42	?		No output	
43	?		No output	
44	✔			
45	✔			
46	✔			
47	✔			
48	✔			
49	✔			
50	✔			
51	✔			
52	✔			

53	✖	515-522	```java
			} finally {
			    if (is != null) {
			        try {
			            is.close();
			        } catch (IOException ioe) {
			        }
			    }
			}
			``
				Log the exception instead of leaving the catch block empty

54	✔			
55	✔			
56	✔			
57	✔			
58	✔			
59	✔			
60	✔			

5.4 getSniffersFromModule()

	Respected	Lines	Code	Possible solution
1	✔			
2	✔			
3	✔			
5	✔			
7	✔			
8	✔			
9	✔			
10	✔		Kernighan and Ritchie style	
11	✔			
12	✔			Other blank lines could have been used
13	✖	540 545 546 547 553 572 593		Some are necessary due to the multiple parameters requested by the constructors
14	✖	545 547		Line 545 has 151 characters Line 547 has 153 characters

15	✔		`deplLogger.log(Level.WARNING,`
			`        INCOMPATIBLE_TYPE,`
			`        new Object[] { type,`
			`              md.getArchiveUri(),`
			`              incompatType });`
				Always respected, even if from line 578 to 582 it is unnecessary

16	✔			
17	✔			
18	✔			Always respected, although the comment in line 538 should have been a Javadoc
19	✔			No commented out code
29	✔			
30	✔			
31	✔			
32	✔			
33	✖	556 574	Sniffer mainSniffer = null; List<Sniffer> sniffersToRemove = new ArrayList<Sniffer>();	They should be declared at the beginning together with the others
34	✔			
35	✔			
37	✔			
38	✔			

39	✔			
40	✔			
41	?		No output	
42	?		No output	
43	?		No output	
44	✔			
45	✔			
46	✔			
47	✔			
48	✔			
49	✔			
50	✔		Not present	
51	✖	547	ExtendedDeploymentContext context = new DeploymentContextImpl(null, archive, parameters, habitat.<ServerEnvironment>getService(ServerEnvironment.class));	Remove the implicit type conversion
52	✔		Not present	
53	✔		Not present	
54	✔		Not present	
55	✔		Not present	
56	✔			
57	✔		Not present	
58	✔		Not present	
59	✔		Not present	
60	✔		Not present	

5.5 getTypeFromModuleType()

	Respected	Lines	Code or annotation	Possible solution
1	✔			
2	✔			
3	✔			
5	✔			
7	✔			
8	✔			
9	✔			
10	✔		Kernighan and Ritchie style	
11	✔			
12	✔		No blank lines or commented out lines	
13	✔			
14	✔			
15	✔			
16	✔			

17	✖		`if (moduleType.equals(DOLUtils.warType())) {`
			`    return "web";`
			`} else if (moduleType.equals(DOLUtils.ejbType())) {`
			`    return "ejb";`
				The "else if" statements should be on a new line, aligned with the "if". Align correctly

18	✔		No comments	
19	✔		No comments	
29	✔			
30	✔			
31	✔			
32	✔			
33	✔			
34	✔			
35	✔			
37	✔		No arrays	
38	✔		No arrays	
39	✔		No arrays	
40	✔			
41	?		No output	
42	?		No output	
43	?		No output	
44	✖		The method is made of an "if" statement followed by three "else if" statements	Change the implementation of ModuleType in order to avoid the chained "else if"
45	✔			
46	✔			
47	✔			
48	✔			
49	✔			
50	✔			
51	✔			
52	✔		No exceptions	
53	✔		No exceptions	
54	✔		No switch statements	

55	✔		No switch statements	
56	✔		No loops	
57	✔		No files	
58	✔		No files	
59	✔		No files	
60	✔		No files	

5.6 getConfigurationDeploymentDescriptorFiles()

	Respected	Lines	Code	Possible solution
1	✔			
2	✔			
3	✔			
5	✔			
7	✔			
8	✔			
9	✔			
10	✔		Kernighan and Ritchie style	
11	✔			
12	✔			
13	✖	632		Declaration can be split on multiple lines
		635		Statement condition can be split on multiple lines
14	✖	632 633 635 637	These lines exceed the 120 characters	
15	✔			
16	✔			
17	✔			
18	✔			
19	✔			
29	✔			
30	✔			
31	✔			
32	✔			
33	✔			
34	✔		It respects the order of parameters used for the other methods	
35	✔			
37	✔			
38	✔	635	No check for the get(0) call, the call can throw an out of bound exception	
39	✔			
40	✔			
41	❓		No output	
42	❓		No output	
43	❓		No output	
44	✔			
45	✔			
46	✔			
47	✔			
48	✔			
49	✔			
50	✔			
51	✔			

52	✔			
53	✔			
54	✔			
55	✔			
56	✔			
57	✔			
58	✔			
59	✔			
60	✔			

5.7 setElementValue()

	Respected	Lines	Code	Possible solution
1	✖	650-652 659	Object o, String Value String schema	Rename the variable
2	✖	650-652	Object o	Rename the variable
3	✔			
5	✔			
7	✔			
8	✔		There have been used always 4 spaces	
| 9 | ✖ | 659-665 | ```
String schema;
if (st.hasMoreElements()) {
    schema = (String) st.nextElement();
} else {
    schema = namespace;
    namespace = TagNames.JAVAEE_NAMESPACE;
}
``` | Remove TABS |
| 10 | ✔ | | Kernighan and Ritchie style | |
| 11 | ✖ | 667<br>669<br>671 | ```
if (namespace.equals(TagNames.J2EE_NAMESPACE))
    continue;
if (namespace.equals(TagNames.JAVAEE_NAMESPACE))
    continue;
if (namespace.equals(W3C_XML_SCHEMA))
    continue;
``` | Add missing braces |
12	✔			
13	✖	677 679	These 2 lines exceeds the 80 characters	
14	✔	677 679	These 2 lines not exceeds the 120 characters	
15	✔			
16	✔			
| 17 | ✖ | 659-665 | ```
String schema;
if (st.hasMoreElements()) {
    schema = (String) st.nextElement();
} else {
    schema = namespace;
    namespace = TagNames.JAVAEE_NAMESPACE;
}
``` | Align with the previous statement |
18	✔			
19	✔			
29	✔			
30	✔			
31	✔			
32	✔			
33	✖	676	`String clientSchemaLocation = sb.toString();`	This declaration must be at the beginning of the block
34	✔			
35	✔			
37	✔		No array	
38	✔		Usa hasmoreelemnts	
39	✔			
40	✔			

41	?		No output	
42	?		No output	
43	?		No output	
44	✖	667 669 671	`if (namespace.equals(TagNames.J2EE_NAMESPACE))` `    continue;` `if (namespace.equals(TagNames.JAVAEE_NAMESPACE))` `    continue;` `if (namespace.equals(W3C_XML_SCHEMA))` `    continue;`	This can be done with one if statement
45	✔			
46	✔			
47	✔			
48	✔			
49	✔			
50	✔			
51	✔			
52	✔			
53	✔			
54	✔			
55	✔			
56	✔			
57	✔			
58	✔			
59	✔			
60	✔			

6 Other Problems

- As said before, the method "setElementValue" inspected is a dead code, as it is never called in all the project. It may cause a misunderstanding for a developer with the task of improving the code, and it would be necessary at least to underline this issue with a proper Javadoc or commented out lines of code
- About the length of the lines (points 13 and 14 of the checklist), in order to remain in the maximum of 80 lines, it was decided to count only the blank spaces that occur after the first character and before the last one, so the blank spaces used at the beginning of the line to make the indentation are not considered