POLITECNICO
MILANO 1863

Inspection Document - myTaxiService

Ciceri, Filippo
filippo.ciceri@mail.polimi.it

Cesaro, Federico
federico.cesaro@mail.polimi.it

Capecchi, Luca
luca.capecchi@mail.polimi.it

January 5, 2016

Contents

1 Assigned Classes and Methods

Web Service Annotation: Both classes assigned to our group were related to javax.jws.WebService annotations. The annotation process and the WebService annotation style are described briefly below

1.1 Annotation

Annotations are files containing metadata about blocks of code or entire files. More information about annotations can be found on the official java site in the tutorial section at `https://docs.oracle.com/javase/tutorial/java/annotations/`.

1.2 javax.jws.WebService

The WebService annotation type is used when referring to Java classes and interfaced that implement a Web Service or a Web Service interface. The WebService annotation is built as follows:

Annotation Name	Description
name	The name of the Web Service. Used as the name of the wsdl:portType when mapped to WSDL 1.1.
targetNamespace	If the @WebService.targetNamespace annotation is on a service endpoint interface, the targetNamespace is used for the namespace for the wsdl:portType (and associated XML elements). If the @WebService.targetNamespace annotation is on a service implementation bean that does NOT reference a service endpoint interface (through the endpointInterface attribute), the targetNamespace is used for both the wsdl:portType and the wsdl:service (and associated XML elements). If the @WebService.targetNamespace annotation is on a service implementation bean that does reference a service endpoint interface (through the endpointInterface attribute), the targetNamespace is used for only the wsdl:service (and associated XML elements).
serviceName	The service name of the Web Service. Used as the name of the wsdl:service when mapped to WSDL 1.1.
portName	The port name of the Web Service. Used as the name of the wsdl:port when mapped to WSDL 1.1.
wsdlLocation	The location of a pre-defined WSDL describing the service. The wsdlLocation is a URL (relative or absolute) that refers to a pre-existing WSDL file. The presence of a wsdlLocation value indicates that the service implementation bean is implementing a pre-defined WSDL contract. The JSR-181 tool MUST provide feedback if the service implementation bean is inconsistent with the portType and bindings declared in this WSDL. Note that a single WSDL file might contain multiple portTypes and multiple bindings. The annotations on the service implementation bean determine the specific portType and bindings that correspond to the Web Service.
endpointInterface	The complete name of the service endpoint interface defining the service?s abstract Web Service contract. This annotation allows the developer to separate the interface contract from the implementation. If this annotation is present, the service endpoint interface is used to determine the abstract WSDL contract (portType and bindings). The service endpoint interface MAY include JSR-181 annotations to customize the mapping from Java to WSDL. The service implementation bean MAY implement the service endpoint interface, but is not REQUIRED to do so. If this member-value is not present, the Web Service contract is generated from annotations on the service implementation bean. If a service endpoint interface is required by the target environment, it will be generated into an implementation-defined package with an implementation- defined name.

All this information may be found on the Java EE site at `https://docs.oracle.com/javaee/5/api/javax/jws/WebService.html`.

1.3 Methods and their role

- ignoreWebserviceAnnotations(AnnotatedElement annElem,AnnotatedElementHandler annCtignoreWebserviceAnnotationsx):
  Verifies if the annotated element has a Stateless and a WebService and if the annotation context is a WebBundleContext or a WebComponentContext. In that case, returns true to ignore the annotation so it doesn't get added twice to the bundle descriptors.

- isJaxwsRIDeployment(AnnotationInfo annInfo):
  Verifies that WEB-INF/sun-jaxws.xml exists and it is not processed in EJB context (if the context is a WebBundleContext or a WebComponentContext).

- HandlerProcessingResult processAnnotation(AnnotationInfo annInfo):
  Processes annotations for javax.jws.WebService annotations and packages all relevant information into the annInfo instance.

2 Issue List

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`.

Indentation

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

9. No tabs are used to indent.

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();
}
```

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.

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.

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.

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).

Package and Import Statements

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

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.

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.

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.

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.

Object Comparison

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

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.

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.

Exceptions

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

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.

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.

3 Code Issues

From now on in this section, we will change the way how we call the methods in order to simplify the reading and the comprehension.
In particular we will refer to the method:

- *HandlerProcessingResult processAnnotation*

  (

  AnnotationInfo annInfo

)

  as Method 1

- *Boolean isJaxwsRIDeployment*

  (

  AnnotationInfo annInfo

)

  as Method 2

- *Boolean ignoreWebserviceAnnotations*

  (

  AnnotatedElement annElem,AnnotatedElementHandler annCtignoreWebserviceAnnotationsx

)

  as Method 3

Issue	Method 1	Method 2	Method 3
1	ok	ok	ok
2	ok	ok	ok
3	ok	ok	ok
4	ok	ok	ok
5	ok	ok	ok
6	ok	ok	ok
7	ok	ok	ok
8	There is an inconsistent indentation in line 281	ok	ok
9	ok	ok	ok
10	ok	ok	ok
11	ok	ok	ok
12	ok	ok	ok
13	Line lenght exceeds 80 characters	Line lenght exceeds 80 characters	Line lenght exceeds 80 characters
14	Line lenght does exceed 120 characters in multiple line (132-153-301-320)	ok	ok
15	See below	ok	ok
16	ok	ok	ok
17	ok	ok	ok
18	Comments could be more professional	Comments could be more professional	No comments
19	ok	ok	ok
20	ok	ok	ok
21	ok	ok	ok
22	Not implemented consistently	ok	ok
23	Incomplete javadoc	Incomplete javadoc	Incomplete javadoc
24	ok	ok	ok
25	ok	ok	ok
26	ok	ok	ok
27	The method could be broken down in multiple methods	Dupicated method in WebServiceProvider-Handler.java	Dupicated method in WebServiceProvider-Handler.java
28	ok	ok	ok
29	ok	ok	ok
30	ok	ok	ok

Issue	Method 1	Method 2	Method 3
31	ok	ok	ok
32	ok	ok	ok
33	See below	ok	ok
34	ok	ok	ok
35	ok	ok	ok
36	ok	ok	ok
37	ok	ok	ok
38	ok	ok	ok
39	ok	ok	ok
40	See below	See below	See below
41	ok	ok	ok
42	ok	ok	ok
43	ok	ok	ok
44	ok	ok	ok
45	ok	ok	ok
46	ok	ok	ok
47	ok	ok	ok
48	ok	ok	ok
49	ok	ok	ok
50	Used generic exception e	Used generic exception e	ok
51	ok	ok	ok
52	Used generic exception e	Used generic exception e	ok
53	ok	No action performed	ok
54	ok	ok	ok
55	ok	ok	ok
56	ok	ok	ok
57	ok	ok	ok
58	ok	ok	ok
59	ok	ok	ok
60	ok	ok	ok

3.1 Indentation

The indentation is generally well used except in line 281 of Method 1.

3.2 File Organization

Several issues arose analyzing the length of lines of code. In more than one instance, these not only exceeded the 80 character standard, but even the 120 character limit. Examples inlude lines 135-136, 154, 301, 320. In these cases, the length of the methods being called was too long and fragmenting the code would look very dirty. Curiously, at line 143 there seesm to be a typing mistake, where almost 40 empty spaces appear at the end of the line of code.
In other cases, like lines 218 and 297, the lines of code are longer than 80 characters but only marginally so, under the 120 character recommendation.

3.3 Wrapping lines

Sometimes there are more than one line breaks or comments after a set of operations is closed, like "if" and variable declarations/initializations. However, this is acceptable since it simplifies code reading.

3.4 Comments

Most comments are well placed and help clarify the code in the analyzed methods. However, in some instances like at line 155, some comments come across as unprofessional, with improper usage of exclamation marks and the like. Also, the comment in line 604(method 3) does not clearly explain what the program does when an exception is caught.

3.5 Java Source Files

During our analysis two problems arose with the java source files. The first problem was improper implementation of methods. While both classes extended an abstract class, only one of them properly overrides the methods. In the WebServiceProviderHandler class, method 1(line 107) and the inner class (line 101)

```
public Class<? extends Annotation>[]
getTypeDependencies()
```

are both missing the @Override flag.

Also, the javadoc is incomplete for a few private methods, method 2 and method 3. While not mandatory, it is incoherent. Firstly, it is only written for one of two methods. Furthermore, in the case of method 2, at lines 589-591, the javadoc only semantically explains the behavior of the code. The significance of the parameter and of the return value are not explicitly stated. The same thing happens in the WebServiceProviderHandler class, where the same methods exist and are documented in the same manner.
Another problem found with the document was the length of method 1. While it is mostly just calls to other methods, in some cases it is a long list checks being done within the code. Each one of these if-statement chains could have been moved to a separate method, for clarity and an easier comprehension of the code.

3.6 Class and Interface Declarations

Classes are correctly done and there are no issues with the declarations barring the notation errors described in the previous subsection. However, there seems to be some duplicate code in the WebServiceHandler class and the WebServiceProviderHandler class, where methods are duplicated. While method 1 is different in each file, methods 2 and 3 are exactly the same, and as a result should be placed in another class that may be accessed by both. Furthermore the order in which methods 2 and 3 are within the 2 analyzed classes is inverted, which seems to be an incoherent choice.

3.7 Initializations and declarations

Declarations and initializations are mostly done in the right way. The only problem noticed is that in Method 1 there are a lot of declarations that aren't at the beginning. This can still be the right choice in order to make the code more readable; since the method is very long, creating all the variables hundreds of lines before using them could be confusing.

3.8 Object Comparison

The comparisons are always done in the correct way. Comparsions with == are only used to check if an object is equal to *null*.

3.9 Exceptions

Exception are used in a very generic way, most of the time using a generic catch with the Exception *e* and without performing any action.
Only once a catch throws the more specific Exception *AnnotationProcessorException* in Method 1, at line 169.

4 Appendix

4.1 Software and Tools Used

- MiKTeX (http://www.miktex.org/) to format and create this document.
- NetBeans (https://netbeans.org) to read and analyze the code.

4.2 Work Hours

Time spent on the creation of this Code Inspection Document:

- Filippo Ciceri: 12 hours
- Federico Cesaro: 12 hours
- Luca Capecchi: 12 hours