Politecnico di Milano

A.A. 2015-2016

Software Engineering 2 Project: myTaxiService

Code Inspection Document

Damiano Binaghi, Giovanni Zuretti

Contents

1 Assigned Classes **3**

2 Functional Role of the Assigned Classes and Methods **3**
2.1 First Method: process(ProcessingContext ctx , Class c) 4
2.2 Second Method: processAnnotations(ProcessingContext ctx, AnnotatedElement element) 4
2.3 Third Method: process(ProcessingContext ctx, AnnotationInfo element, HandlerProcessingResultImpl result) 5

3 Code Inspection **6**
3.1 Overview of the class 6
3.2 Inspection of process(ProcessingContext ctx , Class c) 7
3.3 Inspection of processAnnotations(ProcessingContext ctx, AnnotatedElement element) 8
3.4 Inspection of process(ProcessingContext ctx, AnnotationInfo element, HandlerProcessingResultImpl result) 9
3.5 Other issues 10

1 Assigned Classes

Our assignment is to inspect three methods from the AnnotationProcessorImpl.java class of the source code of Glassfish 4.1. The namespace pattern through which the class can be found is the following: appserver/common/annotation-framework/src/main/java/org/glassfish/apf/impl/AnnotationProcessorImpl.java, and the assigned methods are: process(ProcessingContext ctx , Class c), processAnnotations(ProcessingContext ctx , AnnotatedElement element), process(ProcessingContext ctx , AnnotationInfo element , HandlerProcessingResultImpl result).

2 Functional Role of the Assigned Classes and Methods

Thanks to the javadoc present in our class and in the interface our class implements, and thanks to the file package-info.java found in our package, we arrived at the following comprehension of what the class we were assigned to does. The package in which our class is found provides the classes necessary to process J2SE 1.5 annotations in the context of the J2EE application server. An annotation, in the Java computer programming language, is a form of syntactic metadata that can be added to Java source code. Classes, methods, variables, parameters and packages may be annotated, thus annotations are defined by their annotation type. The AnnotationProcessorImpl class implements the AnnotationProcessor interface which is the core engine to process annotation. This class assumes that annotation handlers will be registered to it to process a particular annotation type. These annotation handlers have no particular knowledge outside of the annotation they process and the annotated element on which the annotation was defined. The AnnotationProcessorImpl class is responsible for maintaining a list of annotations handlers per annotation type. AnnotationHandlers are added to the class through the pushAnnotationHandler method and can be removed through the popAnnotationHandler method. Alternatively, the Factory singleton can be used to get an initialized AnnotationProcessorImpl object with all the default AnnotationHandler. The annotation processor uses the ProcessingContext to have access to Class instances.The ProcessingContext can be either created from the createContext method or through another mean. Once the ProcessingContext has been initialized, it is passed to the process(ProcessingContext ctx) method which triggers the annotation processing. Each class instance will be processed in order, and if annotations are present, the AnnotationProcessorImpl will also process Field, Constructor and Methods elements. Each time the annotation processor switches for one particular AnnotatedElement to another, it will send start and stop events to any AnnotatedElementHandler interface implementation registered within the ProcessingContext. This allows client code to keep context information about the AnnotatedElements being processed since AnnotationHandler only know about the AnnotatedElement the annotation was defined on.

We are now going to focus on the methods that were assigned to us.

2.1 First Method: process(ProcessingContext ctx , Class c)

The first method is process(ProcessingContext ctx , Class c), which is a private used to process the annotations of the class c in the context ctx. It returns a ProcessingResult type of data and it can be called from two different public method: process(ProcessingContext ctx) which process all classes in the given ctx context, and process(ProcessingContext ctx, Class[] classes) which will process a particula set of classes.
The first thing the method does is to check if the package of the class being processed has not already been processed. If that is the case, it will process the package type annotations, otherwise it will not. After that it checks if the class actually belongs to the context. If not, it reports an error to the ErrorHandler of the context ctx, howeve it does not abort the whole annotation process, returning as result the old result. On the contrary, if the class belongs to the context, the method will start by processing the annotations of the class calling the processAnnotations(ProcessingContext ctx, AnnotatedElement element), passing as annotated element the class itself. After that it will do the same with the FIELD type annotations, the CONSTRUCTOR type annotations (and their parameters annotations), the METHOD type annotations (with their parameters annotations) and finally the class annotations involving possible superclasses of the given class. It will return the new result, which is the old result plus the processed annotations added at each processing step.

2.2 Second Method: processAnnotations(ProcessingContext ctx, AnnotatedElement element)

The second method is processAnnotations(ProcessingContext ctx, AnnotatedElement element), which is also a private method, used to process the annotations of a particular given element in a given context. The method scans all the annotations for the element which is passed as a parameter. For each of the annotations, it initialize a new AnnotationInfo object called subElement. An object of class AnnotationInfo encapsulates all information necessary for an AnnotationHandler to process an annotation. In particular, an instace of this class orovides access to: the Annotation instance, the ProcessingContext and the AnnotatedElement which is a reference to the annotation element (Type,Method,...). If the annotation reffered as subElement has not been processed yet, this method calls the process(ProcessingContext ctx, AnnotationInfo element, HandlerProcessingResultImpl result) in order to actually process the annotation. Otherwise, it logs the fact that the annotation has already been processed. If there was an error in retrieving one of the annotations of the element passed as parameter (this is done using the getAnnotations() method), an ArrayStoreException is catched and logged. This does not prevent the an-

notation processign to proceeds further. The method returns a result which is the set of all the processed annotations of the element.

2.3 Third Method: process(ProcessingContext ctx, AnnotationInfo element, HandlerProcessingResultImpl result)

The third and last method is process(ProcessingContext ctx, AnnotationInfo element, HandlerProcessingResultImpl result), which is the private method use by the previous one to acctually process annotations and add them to the "result" set. As first thing the method logs the fact that the annotation is being processed and adds it to the result set as UNPROCESSED. Then it checks if the annotation starts with "java.lang". In such case the method returns because we are not interested in processing "java.*" annotations. Otherwise, i.e. if the annotation does not starts with "java.lang", the method calls the handler for this specific type of annotation. If the handler is found, we should check for possible precedences before actually start in handling the annotation. If dependencies are found and they have not been processed yet, then they are processed before processing our current annotation, by recursively calling process(ProcessingContext ctx, AnnotationInfo element, HandlerProcessingResultImpl result), passing as element the annotation which holds the precedence right. When all the dependencies are solved, the current annotation is actually processed by the handler. This is done by calling the processAnnotation(AnnotationInfo element) method of the handler, which could raise an AnnotationProcessorException which is immiediately caught. The exception is logged, and, if it is fatal, the method returns throwing an AnnotationProcessorException. Otherwise, the method increments the error-counter (errorCount) and checks if it is greater than 100. In such case the annotation processing is aborted beacuse of too many errors. If the error is not fatal and the error-counter is smaller than 100, then the method sets the ResultType as FAILED. Others exceptions may be catched and result in a termination of the method which launches an AnnotationProcessorException. If the annotation processing was carried out by the handler in a correct way or the catched exception resulted in a record of ResuType FAILED the processingResult are added to the result set passed as parameter.
If no handler was found for the given type of annotation the method asks to the delegate of AnnotationProcessorImpl if it has the right handler. If there is no delegate an error is reported in the ProcessingContext.

3 Code Inspection

We will proceed now with the inspection of the code assigned to us. We will do first a quick overview of what is wrong in general in the class with respect to the checklist that was given to us. Then we will focus on the errrors, again according to the checklist, of the three assigned methods.

3.1 Overview of the class

We found errors for what concerns the points 10, 23 and 28 of the checklist.

- Bracing style is not consistent throughout the class. Sometimes "Allman" style is used, sometimes the "Kernighan and Ritchie" style is used. As stated in point 10, the choice of bracing style should be consistent. We suggest to choose one of the styles, and use it throughout all the code.

- Point 23 is about the completeness of the javadoc in our class. The methods setDelegate() and getStack() are public but there is no javadoc for them. Though they are very simple methods, we think that a brief line of javadoc, which states what the method does, would not hurt. Furthermore, the method log(Level level, AnnotationInfo locator, String localizedMessage) has a really simple javadoc. We think a bit more could have been done, e.g. by adding javadoc on what is the meaning of the parameters required in the method and how they are used.

- Point 28 is about the visibility of the variables. In our class, all the variables are declared with the default visibility of java, which is "package" visibility. However, such variables are not used in other classes of the package outside our class and they are acessed only through the methods of our class. For such reasons we think it would be better to declare them as private, in order to avoid (in case of code maintenance) that future changes of the code could cause a wrong behavior of the code itself.

We will pass now to the inspection of the assigned methods.

3.2 Inspection of process(ProcessingContext ctx , Class c)

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.	We think the variables “ape” and “ctx” could be named in a more significative way, such as, for example, “annotationProcessingError” and “context”.
2. If one-character variables are used, they are used only for temporary “throwaway” variables, such as those used in for loops.	It should be asigned a more significative name to parameter c, which is not a “throaway” variable. One choice could be currentClass, or processedClass
8. Three or four spaces are used for indentation and done so consistently	Indentation of line 172 and lines from 181 to 185 is wrong. There should be 4 spaces less for each of them.
12. Blank lines and optional comments are used to separate sections	In line 211 the blank space is useless and inconsistent with what has been done in the other methods.
42. Check that error messages are comprehensive and provide guidance as to how to correct the problem	Line 182. This is not an error in the proper sense of the term, however instead of “classnotfounderror”, which is not that easy to read, the progammer could have mixed the case to divide the words, i.e. “classNotFoundError”

3.3 Inspection of processAnnotations(ProcessingContext ctx, AnnotatedElement element)

8. Three or four spaces are used for indentation and done so consistently	Indentation missing in line 285, with conseguent wrong indentation for all the lines from 285 to 301.
13. Where practical, line length does not exceed 80 characters. 14. When line length must exceed 80 characters, it does NOT exceed 120 characters.	All the following lines exceed 80 characters, but they do NOT exceed 120 characters. For all of them we thik the reason for which they are exceeding 80 characters is fair enough. 279: declaration of the method. 287: declaration and initialization of a new variable, which has a long class type and a constructor method with three parameeters. 292: the characters are 81 instead of 80. We think it's fine, however it would be also fine do divide the line on more than one line
15. Line break occurs after a comma or an operator.	In lines 298,299,300 the expression within the parenthesis was divided before the '+' and not after, as it is required in this point.
18. Comments are used to adequately explain what the class, interface, methods, and blocks of code are doing.	The comment in line 286 is in the wrong position. It should be in line 282, because, as the comment states, the result is initialized in line 283.
41. Check that displayed output is free of spelling and grammatical errors.	In line 298, speling error in the output: "annotaton" instead of "annotation"

3.4 Inspection of process(ProcessingContext ctx, AnnotationInfo element, HandlerProcessingResultImpl result)

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.	We think the variables “ape”, “ctx” and “ae” could be named in a more significative way, such as, for example, “annotationProcessingError”, “context” and “annotatedElement”.
8. Three or four spaces are used for indentation and done so consistently	Line 364 has wrong indentation (8 spaces instead of 4) causing wrong indentation also for lines 365 and 366. Indentation is missing for line 371. ine 384 has, again, wrong indentation (8 spaces instead of 4) causing wrong indentation also for line 385
11. All if, while, do-while, try-catch, and for statements that have only one statement to execute are surrounded by curly braces.	In line 319 curly braces are missing after an if statement.
12. Blank lines and optional comments are used to separate sections	Line 308 is a blank space in excess, and it is inconsistent with what has been done in the other methods (just one blank space after an opening curly brace, not two like in this case)
13. Where practical, line length does not exceed 80 characters. 14. When line length must exceed 80 characters, it does NOT exceed 120 characters.	All the following lines exceed 80 characters, but they do NOT exceed 120 characters. For all of them we thik the reason for which they are exceeding 80 characters is fair enough, except for line 312. 305: declaration of the method. 312: exceeding 80 characters can be easily avoid, in this case, by putting a line break after the first or the second ’+’ 321: declaration and initialization of a new variable, which has a long class type and is initialized by calling a series of other methods. 339: declaration and initialization of a new variable, which has a long class type and a constructor method with four parameeters. 373: declaration and initialization of a new variable which has a very long class type

15. Line break occurs after a comma or an operator.	The line break at line 384 should be after the '+' operator and not before.
18. Comments are used to adequately explain what the class, interface, methods, and blocks of code are doing.	The method is commented in a good way, which helps to understand how it works. However for the last part of the method, frome line 364 on, code is no longer commented. Comments for this last part could be helpful.
42. Check that error messages are comprehensive and provide guidance as to how to correct the problem	Line 365. This is not an error in the proper sense of the term, however instead of "toomanyerror", which is not that easy to read at a glance, the progammer could have mixed the case to divide the words, i.e. "tooManyErrors".

3.5 Other issues

We found a potential problem not in the code itself, but in how annotations may be used in other classes. Indeed, if two different annotations for the same class are declared to be dependent one from the other and viceversa (i.e. annotation 'A' depends on 'B' and annotation 'B' depends on 'A'), this would cause an infinite loop when trying to solve the dependencies and precedences in the process(ProcessingContext ctx, AnnotationInfo element, HandlerProcessingResultImpl result) method. Thus, we should make sure that, whenever annotations are used, such situation never appears, otherwise it would become a serious issue.