

|  |
| --- |
| Politecnico di Milano A. A. 2015-2016 |
| Inspection Document |
| Software Engineering 2 |

|  |
| --- |
| 05/01/2015 Sartini Stefano Mat.(853697) |

Sommario

[2. Assigned Methods and Classes 3](#_Toc439760231)

[3. Functional role of assigned set of classes 3](#_Toc439760232)

[a. generateResourceObjects 3](#_Toc439760233)

[b. getResourcesOfType 3](#_Toc439760234)

[c. ResourcesXMLParser.java 4](#_Toc439760235)

[4. Code Inspection Checklist 5](#_Toc439760236)

[a. Naming Conventions 5](#_Toc439760237)

[d. Indention 5](#_Toc439760238)

[e. Braces 5](#_Toc439760239)

[f. File Organization 6](#_Toc439760240)

[g. Wrapping Lines 6](#_Toc439760241)

[h. Comments 7](#_Toc439760242)

[i. Java Source Files 7](#_Toc439760243)

[j. Package and Import Statements 7](#_Toc439760244)

[k. Class and Interface Declarations 7](#_Toc439760245)

[l. Initialization and Declarations 8](#_Toc439760246)

[m. Method Calls 8](#_Toc439760247)

[n. Arrays 8](#_Toc439760248)

[o. Object Comparison 9](#_Toc439760249)

[p. Output Format 9](#_Toc439760250)

[q. Computation, Comparisons and Assignments 9](#_Toc439760251)

[r. Exceptions 10](#_Toc439760252)

[s. Flow Control 10](#_Toc439760253)

[t. Files 10](#_Toc439760254)

[5. Other Problems 10](#_Toc439760255)

# Assigned Methods and Classes

As part of the inspection assignment I was given two methods of the same class to analyse. The previously mentioned class is the ResourcesXMLParser.java which is located in appserver/resources/resources-connector/src/main/java/org/glassfish/resources/admin/cli/ inside the glassfish application code. The two assigned methods are: private void generateResourceObjects(String scope) at the line 314 and private static List <org.glassfish.resources.api.Resource> getResourcesOfType (List<Resource> resources, int type, boolean isResourceCreation, boolean ignoreDuplicates) at line 392.

# Functional role of assigned set of classes

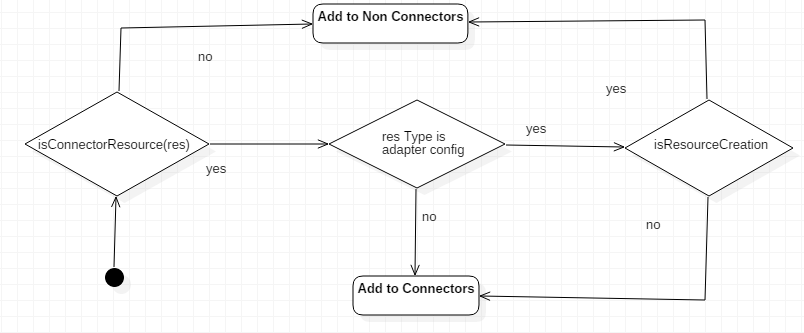
This chapter will try to describe what the two assigned methods do and also the role of the class in which they are. As such it will be divided in three parts.

## generateResourceObjects

This method is void, so it won’t return anything. It receives as an input a String named *scope* which will then be passed to the different methods that will create the resources. The class ResourcesXMLParser.java contains an object of type Document, which will be populated by another method of the class called initProperties at the line 222. InitProperties will parse the XML properties file and then populate the document object. GenerateResourceObjects will go through the nodes of the Document tree with a “for” cycle which initial condition is to create a Node object named nextKid which is the son of the first element in the document. The “for”exiting condition is that the nextKid is a null pointer and the incremental condition is that nextKid points to its sibling. Inside the “for”cycle the method will get the name of the node it is analysing at the moment and compare it with other strings situated in org. glassfish.resources.api.Resources. Once a match is found, it will create a Resource of the type found in the previous step by passing to a particular method of the class the node pointer nextKid and the string *scope* received as an input. There is a particular case for Persistence Manager Factory Resources which are no longer supported as stated in the documentation, so in this case the method will send a string to the logger with the description of the error. To sum it up, the functional role of this method is to scan every node in the Document object and generate the correct resource using other methods of the ResourceXMLParser.java class. Since the incremental condition of the “for” cycle is *nextKid = nextKid.getNextSibling()* this means that the document has a single root node, and then only children of the root. In this way this method would actually parse all the nodes in the document. GenerateResourceObjects is used by the class ResourcesXMLParser.java when one of the two constructor methods are called to initialize and populate the class objects. The two constructor method that use generateResourceObjects are: *public ResourcesXMLParser(File resourceFile)* and *public ResourcesXMLParser(File resourceFile, String scope).*

## getResourcesOfType

This second method will return a List<org.glassfish.resources.api.Resource>, which is a list of objects of type Resource. It receives as an input a list of Resource called *resources*, an integer *type* and two Booleans *isResourceCreation* and *ignoreDuplicates*. The method documentation specifies that the role of this part of the code will be to sort all the resources received as an input in two different lists, one containing all Connector Resources and one containing all Non Connector Resources. Once the different resources have been added to the correct list, two different methods will sort these two lists putting the resources in the order they have to be created. In the end the method will check *type*, and based on its value it will return either the connector list or the non-connector one. Every resource is allocated to the right list by a series of “if” statements which will check if the resource is a connector, if its type is adapter configuration and finally in which phase the method is called, by looking at the input *isResourceCreation*. This last check is necessary because, as specified in the method documentation and annotation, the Resource Adapter Configuration is considered as a Non Connector resource during the resource-creation phase, so depending on the phase this method is called, these particular resources must be treated differently. The phase is defined by the string received as an input: *isResourceCreation*. To better explain this part of the method, a state chart of this part will be shown. The control explained by the state chart is done for every Resource *res* inside the input list of resources.



GetResourcesOfType is a private method used by two public methods of the class, getConnectorResourceList and getNonConnectorResourcesList when they are called by other classes of the application.

## ResourcesXMLParser.java

As previously stated, generateResourceObjects is used in the initialization phase of the class, when the constructor method is invoked. There are two constructor methods, which differ because one receives as an input only a file object, while the other one also receives a string. GetResourcesOfType is a private method used to return a list of sorted Connectors, or a list of sorted Non-Connectors. In conclusion the functional role of this class is to create the Resources used by the application, to allocate them and eventually communicate them to other classes may they need this information. The other methods of the class are used to parse the XML file to populate the document object, initialize the variables and the class, to create the different resources, manage the document file and lists. There is one method for every different type of resource.

# Code Inspection Checklist

In this section of the document the given checklist will be used to verify the correctness of the two assigned methods.

## Naming Conventions

1. All the names chosen in these two methods are meaningful to what the correspondent class, method or attribute represents. Starting from the two methods name, they immediately communicate their role inside the class: generateResourceObjects, clearly explains its role as well as getResourcesOfType, which also suggest you have to give this method a type, in fact it receives as input an integer named *type*. Also attributes and variables have names that immediately recall their use. The only one which is a little more unintuitive is in getResourcesOfType when a new object of type Resource is created with the name *res*, but no better option was available since there already was a list named *resources*. This actually is a good choice that eliminates some misunderstanding that might have happened if the Resource was named *resource*. Another note can be made for the method named *equalsIgnoreCase* called by the generateResourceObjects method which actually isn’t clear on what it does.
2. No one-character variable is ever used in the two given methods, and the only few-character variable is the previously mentioned *res* which is only used as a temporary variable for a cycle.
3. All classes are nouns, in mixed case, such as the class that contains the two methods: ResourcesXMLParser.java. The only thing to notice here is that the name of this class in not in proper mixed case, which would be ResourcesXmlParser.java, but since XML is written all in upper case because it is an acronym, either one of the two would be correct.
4. No interfaces are used in the two methods given for the assignment.
5. All naming conventions for methods are respected in the assigned part of code. A similar argument to the one made in the third point could be made for methods names such as generateJNDIResource(), but since JNDI is an acronym for Java Naming and Directory Interface, the name is acceptable.
6. Class variables respect the naming convention and are in mixed case, such as the previously mentioned *res* at line 398 and some may also begin with a “­\_” such as *\_logger* at line 344.
7. Many constants are used in the two methods, and all of them respect the naming convention from constants. Some examples are EXTERNAL\_JNDI\_RESOURCE or RESOURCE\_ADAPTER\_CONFIG.

## Indention

1. Starting from the import statements and the name of the class that have no spacing before them, four spaces are used for indentation and this is consistent through all the two methods and the entire class.
2. No tabs are used as indentation in this class.

## Braces

1. The getResourcesOfType method is consistent with the “Kernighan and Ritchie” style in all of its lines: the first brace is on the same line of the instruction that opens the new block. The generateResourceObjects on the other hand has a mix of the “Kernighan and Ritchie” style and the “Allman” style. For example if we look at two “if” statements inside the “for” cycle, they use different styles.

The first one is “K&R”, line 321 322 and 323:

if (nodeName.equalsIgnoreCase(Resource.CUSTOM\_RESOURCE)){

generateCustomResource(nextKid, scope);

}

Another “if” statement, which actually is an “else if” at line 336 337 338, is in “Allman” style:

else if (nodeName.equalsIgnoreCase(Resource.MAIL\_RESOURCE))

{

generateMailResource(nextKid, scope);

}

1. In both methods, all if, while, do-while, try-catch and for with only one statements to execute are surrounded by curly braces, just like the two examples of the previous point show.

## File Organization

1. Comments are used to separate the two assigned method from the rest of the code. These comments describe its function and its parameters. Some other part of the class are separated by blank lines.
2. In some cases the line length exceeds 80 character, for example in the lines 324, 328, 332, 341, 344, 355, 359, 363 for the generateResourceObjects method. What makes the lines exceed 80 characters is the fact that each one of the conditions compare the name of the node with a constant from the class org.glassfish.resources.api.Resource.java. The checking is done through the use of a method called equalsIgnoreCase of the previously mentioned Resource.java class. The problem is that sometimes the checking is done between the node name and Resource.CONSTANT and other times between the node name and org.glassfish.resources.api.Resource.CONSTANT. Here are two examples taken from the code:
   1. if (nodeName.equalsIgnoreCase(Resource.CUSTOM\_RESOURCE)){ at line 321
   2. else if (nodeName.equalsIgnoreCase(org.glassfish.resources.api.Resource.EXTERNAL\_JNDI\_RESOURCE)) { at line 324.

The Resource.java class is imported in the first part of the code at line 46 so there should be no need to rewrite every time the path to that class. In any way these are two equivalent ways of using constants for the same class and as such using only the shortest would have been a better choice. Only line 344 exceeds 80 characters for another reason, which is that the string passed to the logger makes the line too long. For the getResourcesOfType method only two line exceed 80 characters, the line 394, 395, 416 and 417, where there is a similar problem since the same 37+ character string (org.glassfish.resources.api.Resource) is used to define the different lists that are created in those parts of the code.

1. The length of the lines never exceeds 120 characters.

## Wrapping Lines

1. The line break always occurs after a comma or an operator in the two methods.
2. There is only one line that is broken in two parts, and this happens at line 344 and 345 when the logger is used, and this is a high-level break.
3. Every statement in aligned with the beginning of the expression at the same level as the previous line.

## Comments

1. The comment for the method getResourcesOfType is very detailed and explains what the method does, all its parameters and functionalities. The comment for the generateResourceObjects is way less detailed and only gives a brief description of what the method does, without explaining which objects it uses or how it reaches its goals. It says that the method “Get(s) ALL the resources from the document object” without explaining that it also creates the resources based on what type they are. The comment for this method could have been more detailed. All the other comments on lines of the code are detailed enough and give a good explanation on some particular lines of the application and their functionalities (lines 340, 401, 402, 403).
2. There is no commented out code in the two assigned methods.

## Java Source Files

1. The RespurceXMLParser.java file contains only one public class ResourceXMLParser, and two more classes: AddresourcesErrorHandler that implements ErrorHandler, and MyLexicalHander which implements LexicalHandler, but these two are not public.
2. The public class ResourceXMLParser is the first class in the file.
3. No external programs interfaces are used in the two assigned method.
4. The two assigned methods are not present in the Javadoc of this class. This is because only public methods are reported in the Javadoc, and the two assigned methods are both private. GenerateResourceObjects in used in the two constructor methods which are reported in the Javadoc but they are not described with detail. GetResourcesOfType is used by the two public methods GetNonConnectorResourceList and GetConnectorResource list, and both these methods are well described in the Javadoc, with reference to the methods functionalities and parameters. As described at the point 18, comments for the first method are really scares, while they are complete for the second one.

## Package and Import Statements

1. Since the assigned part of the code are two methods, no package or import statements are present. Anyway the class that contains the two assigned methods contains correct package statements, followed by import statements.

## Class and Interface Declarations

1. The ResourceXMLParser class declarations are in the correct order except for the class variables: instance variables are declared before the class variables, while they should be the other way around. For example, “private Document *document*;”(line 76) should come after “private static final int NONCONNECTOR = 2;”(line 94). Other than that, class documentation comment, class statement, constructors, and methods are all in the correct order.
2. The two assigned methods are placed one after the other and this is because they operate necessary in this order: once the resources have been created with the constructor method (which uses the generateResourceObjects method), the class can give the list of resources sorted and divided by type to other classes or methods (using GetResourcesOfType). The two assigned methods are far from the public methods that use them, but this is simply because other private methods are written in between.
3. The two methods are free of duplicates. As for cohesion, it is high in the first method, also in getResourcesOfType it is high since this method creates the lists, but uses another method to sort them. Another factor that determines a high cohesion is the fact that for every different resource object, there is a specific method that will create that object rather than one method that creates different objects. As for coupling these two methods don’t have common variables, changing something in one won’t affect the other one, which makes for a low coupling.

## Initialization and Declarations

1. All classes and variables are of the correct type.The first method is private and void. Private because it is used by the constructor public method and void because it only reads the Document Object, but does not return anything. The second method is private since it is used by a public “get” method and returns a list of Resources Objects.
2. The two methods mainly use private variables of the class in which they are. This avoids external functions form messing with local variables, but they can still read them through the different public methods. All variables declared in the method are instance variables, and are created and then deleted once the code in executed.
3. In generateResourceObjects, the initial condition for the “for” cycle is to create a new node and make it point to the first son of the root of the document. This is done without using the constructor method for the Node Object. On the other hand in getResourcesOfType every list created if firstly created with the use of the constructor and then used by the method.
4. Same problem as before, some object are used without the proper initialization.
5. All variables are initialized where they are declared.
6. All declarations appear at the beginning of a block.

## Method Calls

1. For the first method only one parameter is passed so there is no ordering problem. For getResourcesOfType the parameters are always presented in the correct order: list of resources, type integer, Boolean isResourceCreation and Boolean IgnoreDuplicates.
2. The correct methods are always called by the two assigned methods.
3. The second method correctly returns the list of connectors or the list of non-connectors based on what it received as an input. The first method on the other hand does not return anything but it doesn’t have a return statement at all, when a simple return when the first if ended would have been enough. Furthermore as the code is written even if the resource type of the current node is equal to the first one it is compared to, all the other ones will be anyway checked. To avoid this a return after every method that generates a resource object could have been used. Since the resource type is a string, there can’t be a resource type that corresponds to two different strings, so a return after every method probably should have been used.

## Arrays

1. In generateResourceObjects there are no used arrays. In getResourcesOfType there are four list that are used: a list of connectors, a list of non-connector, a list of sorted connectors and a list of sorted non-connectors. All elements of this lists are added without overlap, and a further control is done checking the value of IgnoreDuplicates: if this Boolean is true, no duplicates will be added to the list.
2. The methods addToList and add used by the getResourcesOfType to add elements to the list verify and avoid going out of bounds.
3. The lists of nonConnectorResources and ConnectorResources are created with the use of the constructor method, while the finalSortedConnectorList and finalSortedNonConnectorList are initialized to point at the output of the two different methods sortConnectorResources and sortNonConnectorRisources.

## Object Comparison

1. All comparison are done with the use of .equals or similar methods: for example in generateResourceObjects al lint 336: “else if (nodeName.equalsIgnoreCase(Resource.MAIL\_RESOURCE))”. In getResourcesOfType we have a similar comparison at the line 399: “if (res.getType().equals(Resource.RESOURCE\_ADAPTER\_CONFIG))”.

## Output Format

1. The only output string is the one passed to the logger in case of persistence manager factory resources that are no more supported. The out is: “persistence-manager-factory-resource is no more supported, ignoring the resource description”. No error is present, a part from the capital letter at the start and a “.” at the end.
2. The above mentioned error is comprehensive and provides a good indication of the problem.
3. The line of output is formatted correctly in terms of line stepping and spacing.

## Computation, Comparisons and Assignments

1. No brutish programming was implemented in the two methods that were assigned. The first one contains eleven if and else if statements, but they are all necessary to determinate the type of resource described in the node and then create it. Since the type is a String the only way to determine which object must be crated is to compare the type of the current resource with every other type and then call the correct method. For the second method there are three conditions to be checked which create four choices, and also in this case three if conditions are required with the same amount of else if. An alternative solution could be sorting the list as soon as a new item is added, but this would only waste computational time. Probably the only thing that can be considered to be “brutish programming” in this second method is the fact that the method creates two different lists, it creates two pointers to the two sorted lists and then returns only one of them based on what type was created. A maybe better solution could have been to implement a check for the correct resources while scanning the input list for the first time, selecting only the correct resources so to create only one list, which would then be sorted and returned.
2. All computation order in the two methods are correct and also are evaluations. There are no mathematical operation executed in the two methods.
3. As said in the previous point, no mathematical operations are present in the two methods.
4. The two methods never divide by zero.
5. The two methods never use mathematical operations such as division.
6. All used Boolean comparisons are correct. They are always done through the use of .equals methods or similar, and the arguments are always Booleans. Sometimes methods that return Boolean are used as conditions and they are always well formulated checks.
7. The second method does not throw exceptions, while the first one does, but no catch is present at the end of the method. The logger is invoked in case of Persistent Factory Manager Resource which is no longer supported.
8. The code is free of implicit type conversions.

## Exceptions

1. As stated in the 50th point, the only exception in the two methods is in the first one and it is not caught, even the logger does not receive an exception as input, but only a constant Level.FINEST which represent the level of priority of the logging action. The Javadoc for this specific java method specifies that FINEST is an integer that can go up to 300.
2. There are no catch blocks in the two assigned methods.

## Flow Control

1. There are no switch statements.
2. There are no switch statements.
3. There is a loop in generateResourceObjects created by a “for” cycle. The initial condition is to create a new object of type Node named nextKid, and to set it to be the first child of the first element in the document. The incremental condition is to make nextKid point to its next sibling. The termination expression is that the nextKid Node is pointing to null. This is a well formed and organized loop, once all the sibling are parsed, the cycle will end. There is another loop in the second method, which is still made with a “for”. In this case initial condition, incremental condition and termination are implicit in the definition of the cycle: for(Resource res : resources) is the for statement and this means that the for will parse all the objects of type Resource inside the list of objects named resources. The current object that the cycle is analysing is called “res”. This is a cleaner and lighter way of using a “for” cycle to parse through a list or an array, and avoids brutish programming. This could not be done in the first loop described in this point because a particular method was needed to go to the next element of the list in that case.

## Files

1. – 60. No files are used in the two assigned methods.

# Other Problems

There are no other particular problems except for those already mentioned in the checklist. Some of these more important problems are: for the generateResourceObjects the comment ahead of the method is really inefficient in describing what the method does and its parameters. No return statement is present in this method and without it, the eleven if conditions inside the “for” cycle are always checked. This could be an implementation choice, but since the check is done by comparing the string obtained from the node with other constant strings, and since all these constant strings are different one from the other, there seems to be no need to verify all the conditions every time. As suggested above, a return statement after every method that generates a resource object could save some computational time. The last big problem with the first method is the different way these constant strings are referred to: sometimes org.glassfish.resources.api.Resource.constantname is used, while other times only Resource.constantname is used. This makes the code longer and harder to read. Since that directory is imported in the class the second way of referring to the constants should have always been used. For what regards the second method is has fewer problems: it has the same long way to refer to objects of type Resource, but not to the constants. The real problem with this method is that two lists are created and sorted but only one is returned, the other one is actually useless. As suggested before another condition checking which resource type added while creating the lists would allow to create only one list based on what was required. This list would then be sorted using one of the two methods based on what type of resources they are and finally returned. This would actually make for a more complicated method, but no time or resources would be lost creating a list that is never used. Probably the implemented way is more resource consuming but less time consuming, since another check for every resource could take a lot of time if the number of resource to check is very high. Everything else in the two methods is well written and accordant to the checklist.