

**Code Inspection Document (CID)**

Computer Science and Engineering (CSE)

Software Engineering 2 Project

Year 2015/16

*Date: 05/01/2016 – Version: 1.0*



**STUDENTS:**

*Martino Andrea (****788701****)*

*Marchesani Francesco (****852444****)*

**PROFESSOR:**

*Mirandola Raffaela*

# Table of Contents

1. Table of Contents 2

2. Introduction 3

2.1. Purpose 3

3. Classes that were assigned to the group 4

3.1. ListSubComponentsCommand 4

4. Functional role of assigned set of classes 5

5. List of issues found by applying the checklist 6

5.1. Method 1 Checklist 6

5.2. Method 2 Checklist 9

5.3. Assigned Class in detail 9

6. Other highlighted problems 10

7. Hours of work 10

# Introduction

## Purpose

The **Code Inspection Document** is the third document of the **Software Engineering 2 Project** (*Politecnico di Milano – 2015/2016*). This document concerns the systematic examination of code about the release of the **Glassfish 4.1** Application Server.

Our main reference for the code inspection is the *Code Inspection Checklist* delivered with the *Assignment 3 pdf*.

Of course, we will try to follow the guidelines of the reference as much as possible.

Here is a resume of the steps of the project, with the related deadlines (in green documents already delivered, in yellow the current document):

# Classes that were assigned to the group

## ListSubComponentsCommand

****

**Class name:** *ListSubComponentsCommand*

**Full class path:**

*appserver/deployment/javaee-core/src/main/java/org/glassfish/javaee/core/deployment/ListSubComponentsCommand.java*

**Implemented Interface:** *AdminCommand (*org.glassfish.api.admin.AdminCommand*)*

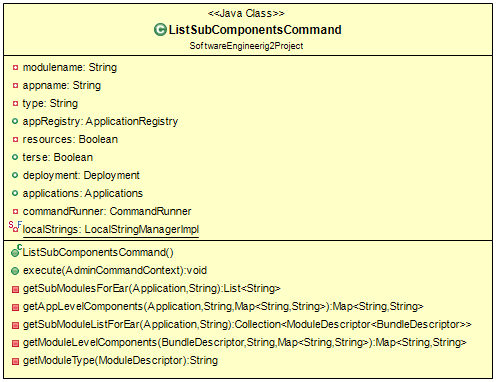
**Online Javadoc**: *http://glassfish.pompel.me/org/glassfish/javaee/core/deployment/ListSubComponentsCommand.html*

**Description:** We focus only on one class (*ListSubComponentsCommand*) and in particular on two methods (*execute [line 133]* and *getSubModulesListForEar [line 321]*). For the functional role of the class, see paragraph *4. Functional role of assigned set of classes*.

**Note:** Only one class was assigned to this group.

# Functional role of assigned set of classes

Qui diciamo che abbiamo usato sia la scarna JavaDoc sul sito della progetto, sia il reverse engineering tramite UML, sia ispezione nuda e cruda del codice.



# List of issues found by applying the checklist

## Method 1 Checklist

|  |  |
| --- | --- |
| **Method** | *private List<String> getSubModulesForEar(com.sun.enterprise.deployment.Application application, String type) {…}* |
| **Naming Conventions** | * The name of the method is self-explicative. In fact, it collect the list of the sub-modules in the application grouped in the Ear (Enterprise Archive). The other methods involved have meaningful and coherent names. The only suggestion is to change the returned variable name *moduleInfoList* with something like *subModuleEarInfoList* (more meaningful, but also longer in practice). * No one-character variables are used in this method. * The classes involved in the method are nouns in mixed case with the first letter capitalized. Only the *application* one has a path-structure:   *com.sun.enterprise.deployment.Application* (probably for path-readability purpose).   * The names of the involved methods follow the standard convention (verbs, with the first letter of each addition word capitalized). * The attributes are all in *CamelCase*. * There are not constants involved in this method. |
| **Indentions** | * The indentation is always a four spaces indentation (**note**: four spaces equals a tab, as a main standard in programming). |
| **Braces** | * The braces follow coherently the Kernighan and Ritchie style. * There are only two blocks in the method (one for, one if). Both of them have the suggested structure:   ***for/if***  *<condition> {*  *<< lines of code >>*  *}* |
| **Files Organization** | * No blank lines and optional comments are used to separate the different parts of the code.   For example, the insertion of blank lines between lines 278-279, 279-280, 282-283 may be a good idea (in order to separate lines of code different from a conceptual point of view).   * There is only a very long line (279) with 115 characters (over 80, but still less than 120). We can do better with a compression of the application class path -> from “*com.sun.enterprise.deployment.Application*” to just “*Application*”. |
| **Wrapping Lines** | * Line breaks always occur after a comma or an operator. * No high level breaks are used, because they are not requested in this specific part of code. * The statements are always aligned with respect to the nesting level. |
| **Comments** | * There is only a short comment line (278) at the top of the method. It says:   // list sub components for ear  The method in fact gets all the subcomponents for ear. Probably, the name of the method is self-explicative, so the comment is a little bit redundant.   * There are not out-date comments for this method. |
| **Java Source Files** | * There is not Javadoc in the method (and in the class). * There is only one class declared in *ListSubComponentsCommand.java*, that implements one interface (*AdminCommand*). * There are not inner classes. |
| **Package and Import Statements** | * There are many “*import*” commands, on the top of the class. |
| **Class and Interface Declarations** | * We are analyzing the *getSubModuleForEar* method in detail. By the way, we have analyzed the full class in paragraph 5.3. |
| **Initialization and Declarations** | * The method is private because it is not used in other classes. * There is only one variable in the method (the list of submodules information). It is clearly nested in the right way. * The list is initialized to empty by default. * The variable *moduleInfoList* is clearly declared at the beginning of the block. |
| **Method Calls** | * The parameters are presented in the correct order. * Every method called in the *getSubModulesForEar* is the right one. There not exist wrong method calls. |
| **Arrays** | * There are not arrays in this code.   Therefore, we will not have problem with indexes or array overflow.  Anyway, there is an ArrayList (*moduleInfoList*), without problems to highlight. |
| **Object Comparison** | * There is only one comparison (line 286) and it is done in the right way (with *equals()*). |
| **Output Format** | * The method is a getter. There is not a displayed output, but just a returned list of elements. * No error messages are necessary. |
| **Computation, Comparisons and Assignments** | * The implementation of the method is elegant. It avoids “brutish programming”. * The order of computation/evaluation, operator precedence and parenthesizing is correct. * No additional parentheses are added to clarify the operator precedence. By the way, we suggest to add parentheses after the ‘=’ in lines 284, 287 to clarify the precedence of the assignment over the string concatenation with ‘+’. * There are no divisions, so there are not denominators that can assume zero as value. * There are no divisions, so we cannot have problem with divisions between integer numbers. * There is only a comparison between objects (line 286). It is done in the right way, using *equals().* * Try-Catch blocks are absent in this part of the code. * There are not implicit type conversions in the analyzed part of code. |
| **Exceptions** | * No try-catch blocks are in the code. * There are not relevant exceptions to catch in the method. |
| **Flow of Control** | * There are not switch statements in the code. * After the execution of the *for statement*, the method always returns the list of sub-modules. Note that there is always convergence because the sub-modules are in a finite number. * The only one loop (for, lines 283~290) is well formed. |
| **Files** | * No Files are directly involved in the method. |

## Method 2 Checklist

## Assigned Class in detail

As we said in the checklists of the methods, in this chapter we will analyze the full class *ListSubComponentsCommand.java* following the specific points of the reference checklist.

We will focus on points 24~33 of the checklist. In fact, these points reflect the entire class and not just the two assigned methods.

* The package statement is the first of the non-comment statements.
* Then there are all the import statements, as suggested in the checklist.
* The class order follows the indications of the point 25 of the checklist.
* Methods are grouped by functionality as suggested.
* The code is free of duplicates and the class is not huge. Anyway, there is a very long methods: *execute(),* with 136 lines of code. We strongly suggest splitting the method in sub-methods in order to respect the “*Divide et Impera*” principle of software engineering.
* All the variables and class member are of the correct type.
* No ‘standard’ constructor of the class exists.
* All parameters are initialized before the use with a value, also with *null* in some cases, see:

*@Param(primary=true)*

*private String modulename = null;*

*@Param(optional=true)*

*private String appname = null;*

*@Param(optional=true)*

*private String type = null;*

[lines 104~111]

* Declarations of variables are always at the beginning of the blocks {…}.

# Other highlighted problems

We want to remark two problems about the *Code Documentation.*

* First, we do not have an appropriate documentation with **comments**. In fact, comments are rare in the code and they are probably too much concise. We know that it is not a good fact to write too many comments, but in this case there is a lack of comments.

Note that the methods are not always easy to understand from an external point of view. A good comprehension takes several hours of *reverse engineering*.

* Second, the **JavaDoc** of the class is very basilar. There are only the header of the class [lines 85 ~ 101] and the indications of Parameters (*@Param*) and Injections (*@Inject*). Our suggestion is to improve the *JavaDoc* (for example with pre-conditions/post-conditions) in order to give the software a better documentation (and to make it more professional and precise).

# Hours of work

* **Andrea Martino** ~ 25 Hours
* **Francesco Marchesani** ~25 Hours