Politecnico di Milano
A.A. 2015-2016
Software Engineering 2
Code **I**nspection

Roberto Clapis (841859), Erica Stella (854443)

January 5, 2016

Contents

1 Assigned Class

We were assigned the three methods

- private void registerBeanValidator(String rarName, ReadableArchive archive, ClassLoader classLoader)
- private List <String>getValidationMappingDescriptors(ReadableArchive archive)
- public void event(Event event)

from class ConnectorDeployer in package
com.sun.enterprise.connectors.module.

2 Functional Role of Class ConnectorDeployer

As far as the documentation is provided, the class methods are concerned with the deployment of previously created applications in their execution environment (a container), which is uniquely identified in order to communicate with the application, and the management of the application's life cycle.

3 Found Issues

3.1 Naming Conventions

3.1.1 1

Class name is meaningful.
No interfaces are in the file.
Method names are meaningful.
Class variables are meaningful but the "env" masks the field "env" of JavaEEDeployer and the "clh" variable has a meaningless name.
Method variables are meaningful, two or three letters names are often used but they are acronyms of what they do and in a ristrected scope they are not considered an issue.
Constants names are meaningful but the "EAR" constant doesn't have a meaningful name;

3.1.2 2

Some one-character variables were found, but they were all "e" for exceptions. We considered acceptable to have exceptions in catch blocks named "e" since they are throwaway variables and they have a very limited scope length.

3.1.3 3

The file only contains class ConnectorDeployer and it respects the naming convention.

3.1.4 4

No interface is declared in the assigned file.

3.1.5 5

All the methods respect the naming convention.

3.1.6 6

The convention is respected, but not for the variable "clh" which is a 3 letter lowercase word used as an acronym.

3.1.7 7

The constants respect the naming convention.

3.2 Indention

3.2.1 8

Indention is coherent, 4 or multiples of 4 spaces are used consistently.

3.2.2 9

line 518 uses tabs to indent.

```
513                         try {
514                             if (inputStream != null) {
515                                 inputStream.close();
516                             }
517                         } catch (Exception e) {
518                             // ignore ?
519                         }
```

3.3 Braces

3.3.1 10

Consistent bracing style is used.

3.3.2 11

Curly braces are always used for conditional statements.

3.4 File Organization

3.4.1 12

A blank line is missing before line 55 to divide imports of different domains. There are 2 meaningless blank lines (216 217) that break coherence in spacing. Blank lines 299, 342, 373 are meaningless.

3.4.2 13

There are 74 lines longer than 80 chars but splitting them would just reduce readability.

3.4.3 14

Only one line (278) exceeds 120 chars and it is 121 chars long.

3.5 Wrapping Lines

3.5.1 15

Line wrappings occur only after the following characters:

`>,=.+`

so the convention is respected.

3.5.2 16

In lines 640 and 649 lines are broken before a parentheses in order to have a higher-level break so the convention is respected.

```
if (resourcesUtil.filterConnectorResources
(resourceManager.getAllResources(), moduleName, true).size() > 0) {
```

3.5.3 17

Alignment of new statements is correct.

3.6 Comments

3.6.1 18

The assigned class lacks of documentation. Most of the methods are not documented, and even if some comments to explain what code does are present,

they are very few and not present in a constant way but only in some parts of the code.

3.6.2 19

Line 594 is the only one that has commented-out code and it has a justification to have the code, but not why it is commented out and when it should be put back in the code or removed.

```
        if (/*env.isDas() && */
            Deployment.UNDEPLOYMENT_VALIDATION.equals(event.type())) {
```

3.7 Java Source Files

3.7.1 20

The file ConnectorDeployer.java contains only the ConnectorDeployer public class.

3.7.2 21

The ConnectorDeployer public class is the first and only class in the file.

3.7.3 22

Since the following methods have no documentation this check is not doable.

- event (due to implementation of EventListener)
- preDestroy (due to implementation of PreDestroy)

The other methods are coherent with documentation.

3.7.4 23

As stated in the point above, two public methods are not documented. Also most of the private methods have no documentation.

3.8 Package and Import Statements

3.8.1 24

The package statement package com.sun.enterprise.connectors.module; is the first non-comment statement and is followed by the import statements.

3.9 Class and Interface Declarations

3.9.1 25

3.9.1.1 A The class has a documentation comment

3.9.1.2 B

The class statement follows his documentation

3.9.1.3 C

There is no implementation comment

3.9.1.4 D

Static variables are after the instance variables, so the convention is not respected.

3.9.1.5 E

There are only private variables

3.9.1.6 F

There is the empty constructor

3.9.1.7 G

The methods follow the constructor

3.9.2 26

Methods are grouped by functionality.

3.9.3 27

Code is free of duplicates longer than one line. There is a very long method that should be split in smaller, simpler methods which is the "event" method.

3.10 Initialization and Declarations

3.10.1 28

Variables and class members are of the correct type and they have the right visibility.

3.10.2 29

Variables are declared in the proper scope.

3.10.3 30

Constructors are always called unless it is intentional to assign "null" to the variable.

3.10.4 31

classLoader in the "load" method is probably intentionally left null, depending on the computation that follows its declaration. (line 176) All the other variables are initialized before use.

3.10.5 32

Every variable is initialized where it is declared, and if it requires further computation to be assigned with a value it is explicitly initialized with null, which sometimes is a valid value (see 31), and later computed and assigned.

3.10.6 33

Declarations appear at the beginning of blocks except for the "event" method, in which they are declared based on context in which they are used.

3.11 Method Calls

3.11.1 34

All the parameters are presented in the correct order.

3.11.2 35

No issues were found regarding this point.

3.11.3 36

All method returned variables are used properly.

3.12 Arrays

3.12.1 37

No issues were found regarding array indexing. All arrays and lists are accessed either with an enhanced for or in a while loop with an iterator that starts from the first element and scans all the other elements until there are no more.

3.12.2 38

As explained in the previous point, no issues were found.

3.12.3 39

Only one array is newly created in our method registerBeanValidator

```
Object params[] = new Object[]{rarName, e};
```

and the constructor is used to create it, so no issues were found.

3.13 Object Comparison

3.13.1 40

No issues were found regarding object comparisons as '==' is never used instead of equals.

3.14 Output Format

3.14.1 41

The only outputs of the methods we were assigned are the entries logged in the logger and they're all free of spelling and grammatical errors.

3.14.2 42

All the error messages logged in case of exceptions state which operation caused the arousal of the exception and eventually provide a stack trace, but no message provides guidance as to how to correct the problem.

3.14.3 43

All the logger's entries are formatted correctly.

3.15 Computation, Comparisons and Assignments

3.15.1 44

No examples of brutish programming have been found.

3.15.2 45

The order of computation and evaluation is correct.

3.15.3 46

There are no expressions in which there could be operator precedence problems.

3.15.4 47

There are no divisions in the methods we were assigned.

3.15.5 48

There is no integer arithmetic in the methods we were assigned.

3.15.6 49

All comparisons and boolean operators are correct.

3.15.7 50

No exceptions are explicitly thrown.

3.15.8 51

There are no implicit type conversions.

3.16 Exceptions

3.16.1 52

All relevant exceptions are caught.

3.16.2 53

The issues found are presented in the following lists divided by the method in which they were found:

- registerBeanValidator:
  - No actions taken at all:

```
513          try {
514             if (inputStream != null) {
515                inputStream.close();
516             }
517          } catch (Exception e) {
518             // ignore ?
519          }
```

  - The outer try block has no corresponding catch block.

```
484          try {
             ...
534          } finally {
             ...
536          }
```

- getValidationMappingDescriptors:
  - No actions taken at all:

```
569             try {
570                reader.close();
571             } catch (Exception e) {
572                //ignore ?
573             }
```

3.17 Flow of Control

3.17.1 54 and 55

There are no switch statements in the methods we were assigned.

3.17.2 56

All loops are correctly formed.

3.18 Files

3.18.1 57

All files are declared and opened properly.

3.18.2 58

All opened files are closed in a "finally" block.

3.18.3 59

Since files are read line by line with a buffered reader the condition is correctly checked with a null-check for the read line.

3.18.4 60

Exceptions are just caught and logged. If an exception is thrown while manipulating the file it is correctly closed.

4 Other Highlighted Problems

- The registerBeanValidator method does not close all streams but only the last one it tried to open and which threw an exception.

5 Appendix

Appendix for Roberto Clapis
Work hours: 10

Software Used:

Task	Software
Edit LaTeX Source	Vim
Convert LaTeX source to PDF	LaTeXMK
Spell Check	Aspell
LaTeX Check	LaCheck

Appendix for Erica Stella
Work hours: 10

Software Used:

Task	Software
Edit LaTeX Source	TexStudio
Convert LaTeX source to PDF	LaTeXMK