POLITECNICO DI MILANO

SOFTWARE ENGINEERING 2 (2015 - 2016)

Software Engineering 2 Glassfish

Code Inspection v1.0

Authors:

Sun Chao

Daniel Naveda

Bakti Ariani Melinda Pertiwi

January 4th 2016

Contents

1. Assigned Class

We have been assigned to do code inspection on the class ApplicationArchivist. This class belong to the package com.sun.enterprise.deployment.archivist.

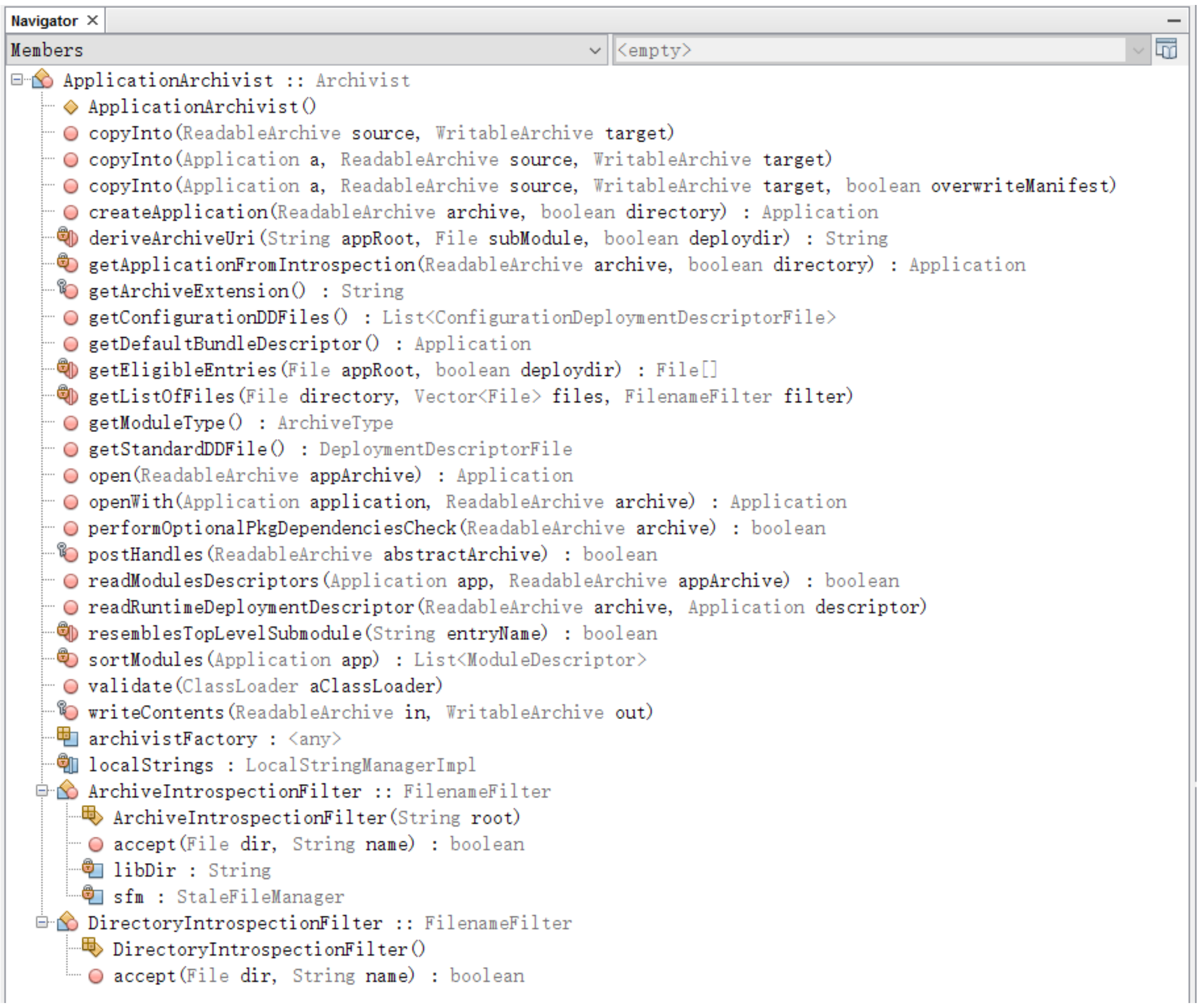
Methods to be inspected:

- getApplicationFromIntrospection(ReadableArchive archive , boolean directory)
- accept(File dir , String name)
- readModulesDescriptors(Application app , ReadableArchive appArchive)

2. Function Role

The purpose of the Class "ApplicationArchivist" is to handle the application archive files. This class is inherited from the class "Archivist", which is responsible for reading and writing correct J2EE Archives.

In order to have a better grasp of the Class "ApplicationArchivist", the following picture shows a summary of the member functions:



```
ApplicationArchivist :: Archivist
    ApplicationArchivist()
    copyInto(ReadableArchive source, WritableArchive target)
    copyInto(Application a, ReadableArchive source, WritableArchive target)
    copyInto(Application a, ReadableArchive source, WritableArchive target, boolean overwriteManifest)
    createApplication(ReadableArchive archive, boolean directory) : Application
    deriveArchiveUri(String appRoot, File subModule, boolean deploydir) : String
    getApplicationFromIntrospection(ReadableArchive archive, boolean directory) : Application
    getArchiveExtension() : String
    getConfigurationDDFiles() : List<ConfigurationDeploymentDescriptorFile>
    getDefaultBundleDescriptor() : Application
    getEligibleEntries(File appRoot, boolean deploydir) : File[]
    getListOfFiles(File directory, Vector<File> files, FilenameFilter filter)
    getModuleType() : ArchiveType
    getStandardDDFile() : DeploymentDescriptorFile
    open(ReadableArchive appArchive) : Application
    openWith(Application application, ReadableArchive archive) : Application
    performOptionalPkgDependenciesCheck(ReadableArchive archive) : boolean
    postHandles(ReadableArchive abstractArchive) : boolean
    readModulesDescriptors(Application app, ReadableArchive appArchive) : boolean
    readRuntimeDeploymentDescriptor(ReadableArchive archive, Application descriptor)
    resemblesTopLevelSubmodule(String entryName) : boolean
    sortModules(Application app) : List<ModuleDescriptor>
    validate(ClassLoader aClassLoader)
    writeContents(ReadableArchive in, WritableArchive out)
    archivistFactory : <any>
    localStrings : LocalStringManagerImpl
    ArchiveIntrospectionFilter :: FilenameFilter
        ArchiveIntrospectionFilter(String root)
        accept(File dir, String name) : boolean
        libDir : String
        sfm : StaleFileManager
    DirectoryIntrospectionFilter :: FilenameFilter
        DirectoryIntrospectionFilter()
        accept(File dir, String name) : boolean
```

getApplicationFromIntrospection(ReadableArchive archive, boolean directory)

```
/**
     * This method introspect an ear file and populate the Application
object.
     * We follow the Java EE platform specification, Section EE.8.4.2
     * to determine the type of the modules included in this application.
     *
     * @param archive   the archive representing the application root
     * @param directory whether this is a directory deployment
     */
```

accept(File dir , String name)

```
/**
     * This method is used to check all files in the archive, if it
satisfies the corresponding condition, then the archive accepts the file.
     * If the file ends with ".war" or ".rar", it accepts
     * If the file is not directory deployment, it refuses to accept
     * If the file is directory deployment and ends with "jar", it accepts
     *
     * @param dir   the archive representing the application root
     * @param name  the name of the file in the archive
     */
```

readModulesDescriptors (Application app, ReadableArchive appArchive)

```
    /**
     * read the modules deployment descriptor from this application object
using
     * the passed archive
     * @param app application containing the list of modules.
     * @param appArchive containing the sub modules files.
     * @return true if everything went fine
     */
```

3. Issues Found

The checklist given in class was applied to the functions specified below. The functions can be found in : "appserver/deployment/dol/src/main/java/com/sun/enterprise/deployment/archivist/ApplicationArchivist.java".

3.1. Function : getApplicationFromIntrospection

Name: getApplicationFromIntrospection(ReadableArchive archive , boolean directory)

Start Line: 276

Location:appserver/deployment/dol/src/main/java/com/sun/enterprise/deployment/archivist/ApplicationArchivist.java

- (Line 292) Violation Checklist 1 : All variable names should have meaningful names and do what the name suggests. “Unknows” is not suitable for understanding the purpose.

```
List<ReadableArchive> unknowns = new ArrayList<ReadableArchive>();
```

- (Line 469) Violation Checklist 7 : We have to declare Constant using all uppercase with words separated by an underscore, but in the code, constant use lowercase alphabet.

```
   private final FileArchive.StaleFileManager sfm;
```

- (Line 278) Violation Checklist 13 : The line length exceed 80 characters. The comment should be written above the statement.

```
 String appRoot = archive.getURI().getSchemeSpecificPart(); //archive
is a directory
```

- (Line 328) Violation Checklist 14 : This line exceed 80 characters and needs a break. For the example after the equal.

```
ModuleDescriptor<BundleDescriptor> md = new
ModuleDescriptor<BundleDescriptor>();
```

- (Line 343) Violation Checklist 14 : This line is longer than 80 characters. It should be there is a break after equal.

```
ModuleDescriptor<BundleDescriptor> md = new
ModuleDescriptor<BundleDescriptor>();
```

- (Line 358) Violation Checklist 14 : This line exceed 80 characters and needs a break after equal.

```
ModuleDescriptor<BundleDescriptor> md = new
ModuleDescriptor<BundleDescriptor>();
```

- (Line 380) Violation Checklist 8 : There are 8 spaces for indention in this line. It should be 4.

```
• if (subArchive != null) {
•                     try {
•                         subArchive.close();
•                     } catch (IOException ioe) {
•                         logger.log(Level.WARNING,
localStrings.getLocalString("enterprise.deployment.errorClosingSubArc
h", "Error closing subarchive {0}", new
Object[]{subModule.getAbsolutePath()}), ioe);
•                     }
•                 }
```

- (Line 398) Violation Checklist 14 : This line exceed 80 characters and needs a break after equal.

```
ModuleDescriptor<BundleDescriptor> md = new
ModuleDescriptor<BundleDescriptor>();
```

- (Line 381) Violation Checklist 14 : The line statement is 180 characters. It needs a break. For the example the break after the comma. (Point 14)

```
logger.log(Level.WARNING,
localStrings.getLocalString("enterprise.deployment.errorClosingSubArc
h", "Error closing subarchive {0}", new
Object[]{subModule.getAbsolutePath()}), ioe);
```

- (Line 276) Violation Checklist 27 : The name of the method is consist of four words, the length should be cut down.

```
    private Application getApplicationFromIntrospection(

            ReadableArchive archive, boolean directory) {

        String appRoot = archive.getURI().getSchemeSpecificPart();
//archive is a directory

        if (appRoot.endsWith(File.separator)) {

            appRoot = appRoot.substring(0, appRoot.length() - 1);

        }
```

- (Line 343) Violation Checklist 12 : Before the first statement of if condition should be no blank line added.

```
if (acArchivist.hasStandardDeploymentDescriptor(subArchive)
                                ||
acArchivist.hasRuntimeDeploymentDescriptor(subArchive)
                                ||
acArchivist.getMainClassName(subArchive.getManifest()) != null) {

                            ModuleDescriptor<BundleDescriptor> md = new
ModuleDescriptor<BundleDescriptor>();
                            md.setArchiveUri(uri);
                            md.setModuleType(DOLUtils.carType());
                            md.setManifest(subArchive.getManifest());
                            app.addModule(md);
                            continue;
                        }
```

- (Line 277) Violation Checklist 8: Line has 4 spaces more than it should have,
- (Line 277) Violation Checklist 1: The directory parameter may cause ambiguous, it's better to change to “IsDirectory”
- (Line 289) Violation Checklist 8: Line has 4 spaces more than it should have,

- (Line 311 - 321) Violation Checklist 17: Wrong warpping line break.

```
if ((!directory && name.endsWith(".war"))
                        || (directory &&
                        (name.endsWith("_war") ||
                                name.endsWith(".war")))) {
                    ModuleDescriptor<BundleDescriptor> md = new
ModuleDescriptor<BundleDescriptor>();
                    md.setArchiveUri(uri);
                    md.setModuleType(DOLUtils.warType());
                    // the context root will be set later after
                    // we process the sub modules
                    app.addModule(md);
                }
```

- (Line 276 - 413) Violation Checklist 25: Wrong order of instance variables. Instance variables are declared in wrong order. Order should be: public, protected, package level and private at last. This is not the case in this class.
- (Line 336,350) Violation Checklist 18: Wrong comments.

3.2. Function: accept

Name: accept(File dir , String name)

Start Line: 483

Location:appserver/deployment/dol/src/main/java/com/sun/enterprise/deployment/archivist/ApplicationArchivist.java

- (Line 469) Violation Checklist 3: Class names are nouns, in mixed case, with the first letter of each word in capitalized. So “FilenameFilter” should be written as “FileNameFilter”

```
   private static class ArchiveIntrospectionFilter implements
FilenameFilter
```

- (Line 504) Violation Checklist 3: Class names are nouns, in mixed case, with the first letter of each word in capitalized. So “FilenameFilter” should be written as “FileNameFilter”

```
   private static class ArchiveIntrospectionFilter implements
FilenameFilter
```

- (Line 471) Violation Checklist 7: Constants should be declared using all uppercase. Here “sfm” is a constant.

```
       private final FileArchive.StaleFileManager sfm;
```

- (Line 483 - 501) Violation Checklist 7: “true” and “false” is Boolean value, better to be written as “TRUE” “FALSE”.
- (Line 452) Violation Checklist 8: There are 8 spaces for indention, it should be reduced to 4.

```
        new ArchiveIntrospectionFilter(appRoot.getAbsolutePath()));
```

- (Line 457) Violation Checklist 8: There are 8 spaces for indention, it should be reduced to 4.

```
    File directory, Vector<File> files, FilenameFilter filter) {
```

3.3. Function: readModulesDescriptors

Name: readModulesDescriptors (Application app, ReadableArchive appArchive)

Start Line: 534

Location:appserver/deployment/dol/src/main/java/com/sun/enterprise/deployment/archivist/ApplicationArchivist.java

- (Line 544) Violation Checklist 14: The line statement contains 193 characters (Point 14 Constraint). This should be reduced by line breaks.

```
throw new
IllegalArgumentException(localStrings.getLocalString("enterprise.d
eployment.unsupporturi", "Unsupported module URI {0}, it contains
space(s)", new Object[]{aModule.getArchiveUri()}));
```

- (Line 550) Violation Checklist 13 : A line statement of 86 characters (Point 13 Constraint). A simple line break after the equal operator would solve it.

```
Archivist newArchivist =
archivistFactory.get().getArchivist(aModule.getModuleType());
```

- (Line 560) Violation Checklist 14: A line statement of 206 characters (Point 14 Constraint). Line breaks after the commas in the parameters' area would improve it.

```
• if (embeddedArchive == null) {
•                throw new
IllegalArgumentException(localStrings.getLocalString("enterprise.d
eployment.nosuchmodule", "Could not find sub module [{0}] as
defined in application.xml", new
Object[]{aModule.getArchiveUri()}));
•             }
```

- (Line 587) Violation Checklist 8 : The indentation of this block is 8 characters (Point 8 Constraint), it should be 4.

```
•          Object rdd = extension.open(newArchivist, embeddedArchive,
   descriptor);
```

- (Line 599) Violation Checklist 14: A line statement of 131 characters (Point 14 Constraint). Line breaks are needed.

```
•      DOLUtils.readAlternativeRuntimeDescriptor(appArchive,
   embeddedArchive, newArchivist, descriptor,
   aModule.getAlternateDescriptor());
```

- (Line 605) Violation Checklist 13: A line statement of 104 characters. Line breaks are needed.

```
•    extension.getKey().readRuntimeDeploymentDescriptor(newArchivist,
   embeddedArchive, extension.getValue());
```

- (Line 580) Violation Checklist 19: This is a TODO comment line; it should have a date, so it helps to understand when this requirement was established and if it is still necessary.

4. Appendix

4.1 Reference Materials

- Assignment of the functions: http://assignment.pompel.me/.
- Code reference: http://glassfish.pompel.me
- Document “Assignment 3.pdf” given by the professor.

4.2 Tools

- Notepad++: used to read the java file
- Microsoft Office: used to redact the document
- Subversion: used to download the source project from SVN site

4.3 Checklist

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;

Indention

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
a. first public class variables

b. next protected class variables

c. next package level (no access modifier)

d. last private class variables
E. instance variables
a. first public instance variables

e. next protected instance variables

f. next package level (no access modifier)

g. 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