Politecnico di Milano

Software engineering 2

Code inspection v1.0

Alessandro Caprarelli	874206
Roberta Iero	873513
Giorgio De Luca	875598

February 3, 2017

Contents

Chapter 1

Introduction

1.1 Purpose

The purpose of this document is to recap all the results deriving from the analysis of the code. This analysis is performed taking into account the main inspection techniques that check if the code is 'well-written' or not. The term 'well-written' means that the code has to be written following a certain set of rules. These are summarized in the following checklist:

- Naming Conventions
- Indention
- Braces
- File Organization
- Wrapping Lines
- Comments
- Java Source Files
- Package and Import Statements
- Class and Interface Declarations
- Initialization and Declarations
- Method Calls
- Arrays

- Object Comparison
- Output Format
- Computation, Comparisons and Assignments
- Exceptions
- Flow of Control
- Files

1.2 Scope

The main scope of this document is to give developers a list of mistakes to repair in order to make the code more robust and of quality. In this way if the developers write the code following the same conventions, it will be also more readable.

1.3 Definition, acronyms, abbrevations

1.3.1 Definition

1.3.2 Acronyms

- **CI:** Code inspection

1.3.3 Abbrevetations

1.4 Reference documents

- Code inspection assignment document

1.5 Document overview

This document is composed of five sections:

- **Introduction:** this section contains the description of the document, of its purpose and some general information.
- **Assigned classes:** this section contains the list of the classes that will be inspected in section 4.
- **Functional role:** this part describes what the classes, that are going to be inspected in section 4, do.
- **List of Issues:** in this last section are listed all the issues found during the inspection of the code of the previously described classes. In particular for each class is specified which kind of rule is violated and what would be the solution.

Chapter 2

Assigned classes

The assigned classes that we are going to describe and inspect are:

- `EbayStoreCategoryFacade.java`
- `ControllerViewArtifactInfo.java`

2.1 EbayStoreCategoryFacade.java

The class is declared as follows:

```
57  public class EbayStoreCategoryFacade {
58      ....
345 }
```

Listing 2.1: EbayStoreCategoryFacade declaration

This class resides in a package declared at the beginning of the file:

```
19  package org.apache.ofbiz.ebaystore;
```

Listing 2.2: Package declaration

This package is inside a module called `EbayStore` and its complete pathname is:

```
/apache-ofbiz-16.11.01/specialpurpose/ebaystore/src/main/java/org/
    ↪ apache/ofbiz/ebaystore/EbayStoreCategoryFacade.java
```

2.2 ControllerViewArtifactInfo.java

The class is declared as follows:

```
35 public class ControllerViewArtifactInfo extends ArtifactInfoBase
   ↪ {
36     ....
129 }
```

Listing 2.3: ControllerViewArtifactInfo declaration

This class resides in a package declared at the beginning of the file:

```
19 package org.apache.ofbiz.webtools.artifactinfo;
```

Listing 2.4: Package declaration

This package is inside a module called `WebTools` and its complete pathname is:

```
/apache-ofbiz-16.11.01/framework/webtools/src/main/java/org/apache/
   ↪ ofbiz/webtools/artifactinfo/ControllerViewArtifactInfo.java
```

Chapter 3

Functional roles

3.1 EbayStoreCategoryFacade.java

This class is implemented by means of the Facade pattern. As the name suggests this pattern is used to create an architectural Facade. In fact `EbayStoreCategoryFacade` has the scope of providing a simple interface for a larger and more complex portion of code

This class is used by the following class:

- `EbayEvents`
- `EbayStoreHelper`
- `EbayStoreOptions`

3.2 ControllerViewArtifactInfo.java

This class is a subclass of `ArtifactInfoBase` so it inherits all the field and methods of it. The inherited methods are:

```
@Override
public String getDisplayName() {...}

@Override
public String getDisplayType() {...}

@Override
```

```
public String getType() {...}

@Override
public String getUniqueId() {...}

@Override
public URL getLocationURL() throws MalformedURLException {...}

@Override
public boolean equals(Object obj) {...}
```

Listing 3.1: Inherited methods by ControllerViewArtifactInfo.java

`ControllerViewArtifactInfo` also has other methods, in addition to those inherited by `ArtifactInfoBase`. These methods are:

```
public ControllerViewArtifactInfo(URL controllerXmlUrl, String
    ↪ viewUri, ArtifactInfoFactory aif) throws GeneralException
    ↪ {...}

public URL getControllerXmlUrl() {...}

public String getViewUri() {...}

public Set<ControllerRequestArtifactInfo>
    ↪ getRequestsThatThisViewIsResponseTo() {...}

public ScreenWidgetArtifactInfo getScreenCalledByThisView() {...}
```

Listing 3.2: Other methods of ControllerViewArtifactInfo.java

An Artifact is a product of a software development process.

In this case `ControllerViewArtifactInfo` is a subclass of the Artifact class `ArtifactInfoBase`. As it's possible to notice from its methods listed above, the only actions that `ControllerViewArtifactInfo` makes available are those of finding information about a Controller View class type.

This class is used by the following classes:

- `ArtifactInfoFactory`
- `ControllerRequestArtifactInfo`
- `ScreenWidgetArtifactInfo`

Chapter 4

Issues related to ControllerViewArtifactInfo class

4.1 Naming conventions

4.1.1 Wrong naming conventions

The class attribute `module` is declared at line 36 as static final and therefore its name should be in uppercase.

```
36 public static final String module =
   ↪ ControllerViewArtifactInfo.class.getName();
```

Listing 4.1: Issue

```
36 public static final String MODULE =
   ↪ ControllerViewArtifactInfo.class.getName();
```

Listing 4.2: Possible solution

4.1.2 Meaningless names

The name of the variable `that`, declared at line 114, is meaningless.

```
114 ControllerViewArtifactInfo that = (ControllerViewArtifactInfo)
    ↪ obj;
```

Listing 4.3: Issue

```
114 ControllerViewArtifactInfo cvai = (ControllerViewArtifactInfo)
    ↪ obj;
```

Listing 4.4: Possible solution

4.2 File organization

4.2.1 Lines length more than 80 characters

The following lines exceed 80 characters but are still acceptable:

```
36 public static final String module =
    ↪ ControllerViewArtifactInfo.class.getName();
```

Listing 4.5: Line 36 acceptable violation of the rule

```
59 // populate screenCalledByThisView and reverse in
    ↪ aif.allViewInfosReferringToScreen
```

Listing 4.6: Line 59 acceptable violation of the rule

```
84 String location =
    ↪ UtilURL.getOfbizHomeRelativeLocation(this.controllerXmlUrl);
```

Listing 4.7: Line 84 acceptable violation of the rule

```
115 return UtilObject.equalsHelper(this.controllerXmlUrl,
    ↪ that.controllerXmlUrl) &&
```

Listing 4.8: Line 115 acceptable violation of the rule

```
122 public Set<ControllerRequestArtifactInfo>
    ↪ getRequestsThatThisViewIsResponseTo() {
```

Listing 4.9: Line 122 acceptable violation of the rule

The following lines exceed 80 characters and may be reformatted in a better way:

```
60 if ("screen".equals(this.viewInfoMap.type) ||
   ↪ "screenfop".equals(this.viewInfoMap.type) ||
61                  "screentext".equals(this.viewInfoMap.type) ||
                      ↪ "screenxml".equals(this.viewInfoMap.type))
                      ↪ {
```

Listing 4.10: Line 60 violation of the rule

```
60 if ("screen".equals(this.viewInfoMap.type) ||
61     "screenfop".equals(this.viewInfoMap.type) ||
62     "screentext".equals(this.viewInfoMap.type) ||
63     "screenxml".equals(this.viewInfoMap.type)) {
```

Listing 4.11: Line 60 possible solution

4.2.2 Lines length more than 120 characters

The following lines exceed 120 characters and must be reformatted in a better way:

```
45 public ControllerViewArtifactInfo(URL controllerXmlUrl, String
   ↪ viewUri, ArtifactInfoFactory aif) throws GeneralException {
```

Listing 4.12: Line 45 violation of the rule

```
45 public ControllerViewArtifactInfo(URL controllerXmlUrl,
46                                    String viewUri,
47                                    ArtifactInfoFactory aif)
48     throws GeneralException {
```

Listing 4.13: Line 45 possible solution

```
53 throw new GeneralException("Could not find Controller View [" +
   ↪ viewUri + "] at URL [" + controllerXmlUrl.toExternalForm()
   ↪ + "]");
```

Listing 4.14: Line 53 violation of the rule

```
53 throw new GeneralException("Could not find Controller View [" +
   ↪ viewUri + "]" +
54 " at URL [" + controllerXmlUrl.toExternalForm() + "]");
```

Listing 4.15: Line 53 possible solution

```
57 throw new GeneralException("Controller view with name [" +
   ↪ viewUri + "] is not defined in controller file [" +
   ↪ controllerXmlUrl + "].");
```

Listing 4.16: Line 57 violation of the rule

```
57 throw new GeneralException("Controller view with name [" +
   ↪ viewUri + "]" +
58 " is not defined in controller file [" + controllerXmlUrl +
   ↪ "].");
```

Listing 4.17: Line 57 possible solution

```
65 this.screenCalledByThisView =
   ↪ this.aif.getScreenWidgetArtifactInfo(fullScreenName.substring(poundIndex-
   ↪ fullScreenName.substring(0, poundIndex));
```

Listing 4.18: Line 65 violation of the rule

```
65 this.screenCalledByThisView =
   ↪ this.aif.getScreenWidgetArtifactInfo
66  (fullScreenName.substring(poundIndex+1),
      ↪ fullScreenName.substring(0, poundIndex));
```

Listing 4.19: Line 65 possible solution

```
68 UtilMisc.addToSortedSetInMap(this,
   ↪ aif.allViewInfosReferringToScreen,
   ↪ this.screenCalledByThisView.getUniqueId());
```

Listing 4.20: Line 68 violation of the rule

```
68 UtilMisc.addToSortedSetInMap(this,
69                              aif.allViewInfosReferringToScreen,
70                              this.screenCalledByThisView.getUniqueId());
```

Listing 4.21: Line 68 possible solution

4.2.3 Unuseful blank lines to remove

The following blank lines are not useful to separate declarations of variables:

```
40 protected URL controllerXmlUrl;
41 protected String viewUri;
42
43 protected ConfigXMLReader.ViewMap viewInfoMap;
44
45 protected ScreenWidgetArtifactInfo screenCalledByThisView = null;
```

Listing 4.22: Lines 40-45 violation of the rule

```
47 this.controllerXmlUrl = controllerXmlUrl;
48 this.viewUri = viewUri;
49
50 this.viewInfoMap = aif.getControllerViewMap(controllerXmlUrl,
   ↪ viewUri);
```

Listing 4.23: Lines 47-50 violation of the rule

4.2.4 Useful blank lines to add

The following lines need a blank line to separate sections of code:

```
18 */
19 package org.apache.ofbiz.webtools.artifactinfo;
```

Listing 4.24: Lines 18-19 violation of the rule

```
18 */
19
20 package org.apache.ofbiz.webtools.artifactinfo;
```

Listing 4.25: Lines 18-19 possible solution

4.3 Package and import statements

The following declaration could be improved modifying the import statement:

```
41 protected ConfigXMLReader.ViewMap viewInfoMap;
```

Listing 4.26: Line 41 issue

```
30 import org.apache.ofbiz.webapp.control.ConfigXMLReader.ViewMap;
31
32 /**
```

```
33  *
34  */
35 public class ControllerViewArtifactInfo extends ArtifactInfoBase
     ↪ {
36     public static final String module =
         ↪ ControllerViewArtifactInfo.class.getName();
37
38     protected URL controllerXmlUrl;
39     protected String viewUri;
40
41     protected ViewMap viewInfoMap;
```

Listing 4.27: Line 41 possible solution

4.4 Comments

4.4.1 Classes, interfaces, methods not described

The following classes and methods should be properly described using Javadoc comments:

```
32 /**
33  *
34  */
35 public class ControllerViewArtifactInfo extends ArtifactInfoBase
     ↪ {
```

Listing 4.28: ControllerViewArtifactInfo class description missing

```
44 public ControllerViewArtifactInfo(URL controllerXmlUrl, String
     ↪ viewUri, ArtifactInfoFactory aif) throws GeneralException {
```

Listing 4.29: Constructor description missing

4.5 Java source files

Javadoc descriptions are missing for all the classes and methods.

4.6 Class and interface declarations

4.6.1 Methods grouping by functionality

The following method is in the middle of a group of getter methods, grouped by a functionality:

```
113 @Override
114 public boolean equals(Object obj) {
115     if (obj instanceof ControllerViewArtifactInfo) {
116         ControllerViewArtifactInfo that =
              ↪ (ControllerViewArtifactInfo) obj;
117         return UtilObject.equalsHelper(this.controllerXmlUrl,
              ↪ that.controllerXmlUrl) &&
118             UtilObject.equalsHelper(this.viewUri, that.viewUri);
119     } else {
120         return false;
121     }
122 }
```

Listing 4.30: equals method in the middle of getters

4.7 Initialization and declarations

The following string should be declared as `privatestaticfinal` variable and used instead of the plain text.

```
113 if (location.endsWith("/WEB-INF/controller.xml")) {
```

Listing 4.31: Constant is missing

,

```
112 private static final WEB-INF-CONTROLLER =
      ↪ "/WEB-INF/controller.xml";
113 if (location.endsWith(WEB-INF-CONTROLLER)) {
```

Listing 4.32: Possible solution

4.8 Method calls

4.8.1 Hard-coded values

The following method should have as `endIndex` a parameter related to the string constant proposed in the listing 4.32 and not an hard-coded value.

```
88 location = location.substring(0, location.length() - 23);
```

Listing 4.33: substring invocation

```
86 private static final WEB-INF-CONTROLLER =
     ↪ "/WEB-INF/controller.xml";
87 if (location.endsWith("/WEB-INF/controller.xml")) {
88       location = location.substring(0, location.length() -
           ↪ WEB-INF-CONTROLLER.lenght);
89 }
```

Listing 4.34: substring invocation possible solution

4.8.2 Method invocations on posssible null objects

The following methods are invoked on objects that may be null therefore is needed a check:

```
50 this.viewInfoMap = aif.getControllerViewMap(controllerXmlUrl,
     ↪ viewUri);
```

Listing 4.35: getControllerViewMap invocation

```
50 if(aif!=null){
51      this.viewInfoMap =
           ↪ aif.getControllerViewMap(controllerXmlUrl, viewUri);
52 }
```

Listing 4.36: getControllerViewMap invocation possible solution

---

```
87 if (location.endsWith("/WEB-INF/controller.xml")) {
```

Listing 4.37: getControllerViewMap invocation

```
87 if(location!=null){
88     if (location.endsWith("/WEB-INF/controller.xml")) {
89 }
```

Listing 4.38: getControllerViewMap invocation possible solution

4.9 Other errors

The following second `if` statement is useless. It will never be executed.

```
52 if (this.viewInfoMap == null) {
53     throw new GeneralException("Could not find Controller View
       ↪ [" + viewUri + "] at URL [" +
       ↪ controllerXmlUrl.toExternalForm() + "]");
54 }
55 if (this.viewInfoMap == null) {
56     throw new GeneralException("Controller view with name [" +
       ↪ viewUri + "] is not defined in controller file [" +
       ↪ controllerXmlUrl + "].");
57 }
```

Listing 4.39: Exeption would block the second if