POLITECNICO
MILANO 1863

Code Inspection Document

Lo Bianco Riccardo - Manzoni Mirco - Mascellaro Giuseppe

January 11, 2017
v1.0

Contents

1 Inspected classes

This document contains the work of the group in the for fullfilling the inspection of the classes assigned to the group. In particular the classes we took into consideration are:

- **ICalWorker.java**: ../apache-ofbiz-16.11.01/applications/workeffort/src/main/java/org/apache/ofbiz/workeffort/workeffort/ICalWorker.java
- **RenderSubContentCacheTransform.java**: ../apache-ofbiz-16.11.01/applications/content/src/main/java/org/apache/ofbiz/content/webapp/ftl/RenderSubContentCacheTransform.java

2 Functional role of inspected classes

2.1 ICalWorker.java

This class handles the requests generated by the front-end of the application. In particular, it handles the WebDAV requests and responses regarding a calendar manager. WebDAV is an extension of HTTP that allows clients to perform remote Web content authoring operations. In fact, in the class the method *isValidRequest* can be found, which checks if the request is acceptable throwing the method *setupRequest* and if the user is authorized to apport changes to the calendar throwing *logInUser*. We understood the behavior of the class by analyzing all its public methods. We identified Get and Put requests that are typical of the HTTP and also the PropFind request that is typical of WebDAV. All this requests are related to a work effort calendar like reported in the small Javadoc related to this class and the attribute's name of the requests: the method *handleGetRequest* manages the retrieval of data from the calendar, *handlePutRequest* manages to insert data in the calendar and store it for future use, *handlePropFindRequest* is used to retrieve properties from the calendar and store it in a XML file.
Our hypothesis regarding the functionalities of the class were confirmed by the official documentation of the OFBiz project (https://cwiki.apache.org/confluence/display/OFBIZ/OFBiz+Features):

> "OFBiz can be used as an iCalendar server - enabling users to share calendar information using their iCalendar-aware client (Mozilla Sunbird, Apple iCal, Microsoft Outlook, Windows Vista Calendar, cell phone, PDA). Users can publish department calendars, project calendars, company vacation calendars, etc. They can check on the availability of a conference room, or find out when service is due on a fixed asset - right from their calendar client."

2.2 RenderSubContentCacheTransform.java

This class manages the rendering of sub contents which have already been generated by back-end modules. It also handles the transformation of the results of back-end processes in a renderable format. We came up with this conclusion because the (incomplete) JavaDoc describing the class gave us an essential information: "Freemarker Transform for Content rendering". Referencing Wikipedia: "FreeMarker is a Java-based Template Engine, originally focusing on dynamic web page generation with MVC software architecture. However, it is a general purpose template engine, with no dependency on servlets or HTTP or HTML, and so it is often used for generating source code, configuration files or e-mails.". Instructions at lines from 60 to 69 exploit *FreeMarkerWorker* class static methods. The main used method is *getWrappedObject* which retrieves information regarding the environment from some back-end modules.
We can easily point out that RenderSubContentCacheTransform.java is much more a generic purpose class when compared to ICalWorker.java.

3 List of issues found by applying the checklist

In this paragraph we go through the issues we found in the two classes analized by applying the checklist that was provided in the assignement. Please refer to the assignement document (see the references list at the end of this document) for a complete version of the checklist.

3.1 ICalWorker.java

- **Lines 56-58**

```
/** iCalendar worker class. This class handles the WebDAV requests
    and\\
 * delegates the calendar conversion tasks to
     <code>ICalConverter</code>.\\
 */
```

Useless class Javadoc specification: insufficient and too general information is provided.

- **Line 61**

```
public static final String module = ICalWorker.class.getName();
```

This declaration does not follow the common standard regarding variable naming: for static final variables one has to follow this type of pattern (all uppercase with words separated by an underscore). In this case, the correct declaration is:

```
public static final String MODULE = ICalWorker.class.getName();
```

- **Line 65**

```
public static final class ResponseProperties {...}
```

Missing Javadoc for class *public static final class ResponseProperties* and for constructor method *public ResponseProperties(int statusCode, String statusMessage)*.

- **Lines 112, 116**

```
public static ResponseProperties createNotFoundResponse(String
    statusMessage) {...}
```

```
public static ResponseProperties createOkResponse(String
    statusMessage) {...}
```

Missing Javadoc for the two public methods.

- **Lines 132, 134, 147, 167, 236, 255, 303, 318**
Literal *workEffortId* is duplicated eight times. Define a constant instead.

- **Line 134**

```
GenericValue publishProperties =
    EntityQuery.use(delegator).from("WorkEffort").where("workEffortId",
    workEffortId).queryOne();
```

This line's length exceeds the maximum number of characters allowed (120).

- **Line 143**

```
public static void handleGetRequest(HttpServletRequest request,
    HttpServletResponse response, ServletContext context) throws
    ServletException, IOException
```

This line's length exceeds the maximum number of characters allowed (120). Missing Javadoc for the public method.

- **Line 163**

```
public static void handlePropFindRequest(HttpServletRequest
    request, HttpServletResponse response, ServletContext context)
    throws ServletException, IOException
```

This line's length exceeds the maximum number of characters allowed (120). Missing Javadoc for the public method. The parameter *ServletContext context* is never used in the method.

- **Line 163, 188, 192**

```
Element etagElement = helper.createElementSetValue("D:getetag",
    String.valueOf(System.currentTimeMillis()));
```

```
Element lmElement =
    helper.createElementSetValue("D:getlastmodified",
    WebDavUtil.formatDate(WebDavUtil.getRFC1123DateFormat(),
    lastModified));
```

```
unSupportedProps.add(responseDocument.createElementNS(propElement.getNamespaceURI(),
    propElement.getTagName()));
```

These line's length exceeds the maximum number of characters allowed (120).

- **Lines 196, 200**

```
if (supportedProps.size() > 0)
```

The form *if(supportedProps.isEmpty())* should be used , since we can assume *supportedProps* is always a List.

- **Lines 226**

```
public static void handlePutRequest(HttpServletRequest request,
    HttpServletResponse response, ServletContext context) throws
    ServletException, IOException
```

This line's length exceeds the maximum number of characters allowed (120). Missing Javadoc for the public method. Moreover, the parameter *ServletContext context* can be considered useless in this method since it is passed only to the method *writeResponse(responseProps, request, response, context)* (line 246), which never uses it.

- **Lines 208, 263**

```
Element lmElement =
    helper.createElementSetValue("D:getlastmodified",
    WebDavUtil.formatDate(WebDavUtil.getRFC1123DateFormat(),
    lastModified));
```

```
unSupportedProps.add(responseDocument.createElementNS(propElement.getNamespaceURI(),
    propElement.getTagName()));
```

These line's length exceeds the maximum number of characters allowed (120).

- **Line 269**

```
GenericValue userLogin = null;
```

Useless assignment of *userLogin* to null value: the declaration of *userLogin* should be placed at line 276, where it is assigned to a not null value.

- **Line 279**

```
VisitHandler.getVisitor(request, response);
```

Return value of *getVisitor* method is not assigned to any variable.

- **Line 298**

```
workEffortId = workEffortId.substring(0,
    workEffortId.indexOf("/"));
```

Replace double quotes with single quotes to use *indexOf(char)*, which is faster than *indexOf(String)*.

- **Line 311**

```
private static void writeResponse(ResponseProperties
    responseProps, HttpServletRequest request, HttpServletResponse
    response, ServletContext context) throws IOException
```

This line's length exceeds the maximum number of characters allowed (120). The parameter *ServletContext context* is never used in this method.

3.2 RenderSubContentCacheTransform.java

- **Lines 48-51**

```
/*
 * RenderSubContentCacheTransform - Freemarker Transform for
     Content rendering
 * This transform cannot be called recursively (at this time).
*/
```

Useless class Javadoc specification: too general. In particular, line 50 statement seems to be completely meaningless (recursion can or cannot be supported by a method, not by a class).

- **Line 54, 55**

```
public static final String module =
    RenderSubContentCacheTransform.class.getName();
```

```
public static final String [] upSaveKeyNames = {"globalNodeTrail"};
```

These lines do not follow the common standard regarding variable naming (for static final variables the pattern is: all uppercase letters with words separated by an underscore). In this case, it should be:

```
public static final String MODULE =
    RenderSubContentCacheTransform.class.getName();
```

```
public static final String [] UP_SAVE_KEY_NAMES =
    {"globalNodeTrail"};
```

- **Line 57**

```
@SuppressWarnings("unchecked")
```

It is not clear wether there is an unsafe type checking or not.

- **Line 58**

```
public Writer getWriter(final Writer out, Map args) {...}
```

Map is a raw type. References to generic type Map¡K, V¿ should be parametrized.

- **Line 73**

```
Boolean nullThruDatesOnly = (strNullThruDatesOnly != null &&
    strNullThruDatesOnly.equalsIgnoreCase("true")) ? Boolean.TRUE
    :Boolean.FALSE;
```

This line's length exceeds the maximum number of characters allowed (120). Space missing between *:* and *Boolean.FALSE*.

- **Lines 74, 110, 220**
Literal *subContentId* is duplicated three times. Define a constant instead.

- **Line 75**

```
final boolean directAssocMode =
    UtilValidate.isNotEmpty(thisSubContentId) ? true : false;
```

Useless specification *? true : false*. This method already returns a boolean value.

- **Line 78**

```
val = ContentWorker.getCurrentContent(delegator, trail, userLogin,
    templateRoot, nullThruDatesOnly, contentAssocPredicateId);
```

This line's length exceeds the maximum number of characters allowed (120).

- **Lines 92, 161, 191, 200, 201**
Literal *contentId* is duplicated five times. Define a constant instead.

- **Line 94**

```
// This order is taken so that the dataResourceType can be
    overridden in the transform arguments.
```

Meaningless comment: the "order" it refers to is not specified, no instances of "dataResourceType" was found in the code, the expression "transform argument" is not an English well formed sentence.

- **Line 98**

```
subDataResourceTypeId = (String) view.get("drDataResourceTypeId");
```

The object *view* may be nullable, so the try-catch statement might throw a *NullPointException* here.

- **Line 99-101**

```
catch (IllegalArgumentException e) {
   // view may be "Content"
}
```

The object *view* may be nullable, so the try-catch statement might throw an unchecked *NullPointerException* here.

- **Line 102-104**

```
// TODO: If this value is still empty then it is probably
    necessary to get a value from
// the parent context. But it will already have one and it is the
    same context that is
// being passed.
```

TODO task left unsolved. Moreover, single line markers for comments // are used for a multiple line comment.

- **Line 115-117**

```
@Override
public void write(char cbuf[], int off, int len) {
}
```

This method is empty. It should provide a comment explaining why it was not implemented or an UnsupportedOperationException should be thrown. *char cbuf[]* parameter declaration does not follow the standard java variables' declaration pattern which is: *char[] cbuf*.

- **Line 135**

```
List<Map<String, ? extends Object>> passedGlobalNodeTrail =
    UtilGenerics.checkList(templateRoot.get("globalNodeTrail"));
```

This line's length exceeds the maximum number of characters allowed (120).

- **Line 138-140**

```
if (view != null) {
   thisView = view;
} else if (passedGlobalNodeTrail.size() > 0) {...}
```

All if-else-if blocks should have curly brackets after the else statement, according to the standard pattern for if statements. A call like *!passedGlobalNodeTrail.isEmpty()* should be used instead of *passedGlobalNodeTrail.size() ¿ 0*, since *passedGlobalNodeTrail* is of type List.

- **Line 141**

```
Map<String, ? extends Object> map =
    UtilGenerics.checkMap(passedGlobalNodeTrail.get(passedGlobalNodeTrail.size()
    - 1));
```

This line's length exceeds the maximum number of characters allowed (120).

- **Line 152-153**

```
if (locale == null)
   locale = Locale.getDefault();
```

No curly brackets are used according to the standard pattern for if statements.

- **Line 164**

```
ContentWorker.renderContentAsText(dispatcher, delegator,
    contentId, out, templateRoot, locale, mimeTypeId, null, null,
    true);
```

This line's length exceeds the maximum number of characters allowed (120).

- **Lines 193, 208, 209**
  Literal *contentAssocTypeId* is duplicated five times. Define a constant instead.

- **Lines 193, 212, 213**
  Literal *mapKey* is duplicated five times. Define a constant instead.

- **Lines 194, 216, 217**
  Literal *fromDate* is duplicated five times. Define a constant instead.

- **Line 195**

```
if (Debug.infoOn()) Debug.logInfo("in Render(0), view ." + view ,
    module);
```

No curly brackets are used according to the standard pattern for if statements and two statements are on the same line.

- **Line 198-217**
All if statements are neither following the standard pattern for if statements nor using curly brackets.

- **Line 222**

```
if (Debug.infoOn()) Debug.logInfo("in Render(0), contentIdTo ." +
    contentIdTo , module);
```

No curly brackets are used according to the standard pattern for if-then statements, moreover 'if' statement and 'then' statement are on the same line.

- **Line 223**

```
String delim = "?";
```

Declaration out of a beginning block. Moreover, it seems that this variable is used only in a few lines, which highlights the fact that it is a temporary variable.

- **Line 242**

```
delim = "&";
```

This variable is assigned but never used.

- **Line 245**

```
if (Debug.infoOn()) Debug.logInfo("in Render(2), contentIdTo ." +
    contentIdTo , module);
```

'if' statement does not follow the standard pattern for 'if' statements, nor uses curly brackets.

4 Other problems

4.1 ICalWorker.java

In this class it is present also a nested class *ResponseProperties*, representing the properties of the response like *statusCode* and *statusMessage*. *Response Properties* have an issue with these attributes: they are public attributes and they do not expose any getter. It is better to avoid this behavior. Our suggest is to make all the attributes private, then expose the related getters.
Moreover, in *handleProbFindRequest* several nested blocks can be found (*for, try-catch* etc.). This issue makes the code difficult to read and therefore it is responsible for posssible difficulties in manutention.

4.2 RenderSubContentCacheTransform.java

Up until line 111, the method *getWriter* is responsible for collecting information useful for applying the rendering. Starting from line 112, a new *Writer* object has been returned in a very awful fashion: it uses an anonymous inner class definition (to be precise: it is a sub-class extension of "Writer" class). This anonymous inner class takes a very large number of methods and instructions; therefore, it would have been better to define a class separately.
openEditWrap and *closeEditWrap* methods are clearly opening, filling and closing some front-end wrapper which will contain all information involved. Indeed, there are HTML tags at lines 178 and 251 opening and closing a div tag element. The generation of strings containing interested information takes place in the middle.
There are no JavaDocs specified for the entire class and the anonymous inner class, therefore it is very difficult to read the code. Methods should have less lines of code in general than the ones in this class.
In *closeEditWrap* method there is a very long if statements sequence. It would have been better to use a design patter instead, in order to overcome this awful way of programming which does not exploit the object oriented programming advantages.

5 References

- **OFBiz official documentation** (www.OFBiz.apache.org)
- **Wikipedia** (www.wikipedia.org)
- **Code Inspection Assignment Task Description.pdf** (https://beep.metid.polimi.it/documents/121
bc00-45fe-8c0c-56694c552802)

6 Hours of work

The team divided the work into equivalent parts, even when modeling different parts of the document. In the following table we present a resume of the work division.

Lo Bianco Riccardo	9h
Manzoni Mirco	9h
Mascellaro Giuseppe	9h