



POLITECNICO
MILANO 1863

M.Sc. in Computer Science and Engineering

Software Engineering 2



**Code Inspection Document
(CID)**

Version 1.1

Released on: 4th February 2017

Authors:

Marilena Coluccia
Burcu Cesur
Mustafa Çeçe

Reference Professor: Elisabetta Di Nitto

INDEX

1	Introduction.....	5
1.1	Revision History.....	5
1.2	Purpose and Scope.....	5
1.3	List of Definitions and Abbreviations	5
1.4	List of Reference Documents	6
2	OfBiz.....	7
2.1	The project.....	7
2.2	UspsMockApiServlet	7
2.3	ControlServlet	8
3	Code Inspection.....	9
3.1	UspsMockApiServlet	9
3.1.1	Naming Conventions.....	9
3.1.2	Indentation	9
3.1.3	Braces.....	9
3.1.4	File Organization.....	10
3.1.5	Wrapping Lines.....	10
3.1.6	Comments	11
3.1.7	Java Source Files.....	11
3.1.8	Package and Import Statements	11
3.1.9	Class and Interface Declarations	11
3.1.10	Initialization and Declarations	11
3.1.11	Method Calls	11
3.1.12	Collections	12
3.1.13	Object Comparison	12
3.1.14	Output Format	12
3.1.15	Computation, Comparisons and Assignments	12
3.1.16	Exceptions.....	13
3.1.17	Flow of Control.....	13
3.2	ControlServlet	13
3.2.1	Naming Conventions.....	13
3.2.2	Indentation	14

3.2.3	Braces	14
3.2.4	File Organization.....	15
3.2.5	Wrapping Lines.....	15
3.2.6	Comments	15
3.2.7	Java Source Files.....	15
3.2.8	Package and Import Statements	16
3.2.9	Class and Interface Declarations	16
3.2.10	Initialization and Declarations	17
3.2.11	Method Calls	17
3.2.12	Collections	17
3.2.13	Object Comparison	17
3.2.14	Output Format	17
3.2.15	Computation, Comparisons and Assignments	18
3.2.16	Exceptions.....	18
3.2.17	Flow of Control	18
4	Effort	18

TABLE OF CONTENT

Table 1.1 - Revision History	5
Figure 3.1 – UspsMockApiServlet (Line 96-101)	9
Figure 3.2 – UspsMockApiServlet (Line 82-89)	10
Figure 3.3 – UspsMockApiServlet (Line 90-95)	10
Figure 3.4 – UspsMockApiServlet (Line 67-71)	12
Figure 3.5 – ControlServlet (Line 116-126)	14
Figure 3.6 – ControlServlet (Line 238-240)	14
Figure 3.7 – ControlServle (Line 140-143)	14
Figure 3.8 – ControlServlet (Line 308-311)	14
Figure 3.9 – ControlServlet (Line 212-218)	15
Figure 3.10 – ControlServlet (Line 260-263)	15
Figure 3.11 – ControlServlet (Line 323-332)	16
Figure 3.12 – ControlServlet (Line 236-267)	16
Figure 3.13 – ControlServlet (Line 151-156)	17

1 INTRODUCTION

1.1 Revision History

Release No.	Date	Revision Description
Rev. 0	23/01/2017	Code Inspection Template and Checklist
Rev. 0.1	26/01/2017	First inspection
Rev. 1.0	02/02/2017	Second inspection
Rev. 1.1	04/02/2017	Internal approval

Table 1.1 - Revision History

1.2 Purpose and Scope

Code Inspection (or Code review) is systematic examination of computer source code. It is intended to find mistakes overlooked in the initial development phase, improving the overall quality of software.

An inspection report lists the findings, which include metrics that can be used to aid improvements to the process as well as correcting defects in the document under review.

Inspections are often led by a trained moderator, who is not the author of the code, and divides the code to inspect between inspection teams. In this document the assignment, given by Prof. Di Nitto to team 91, will be described.

In particular, the code inspection of the following classes of OfBiz Apache project will be done:

- UspsMockApiServlet
- ControlServlet

1.3 List of Definitions and Abbreviations

API: Application Programming Interface is used for interacting and communicating with other existing systems.

Servlet: Servlet is a Java object that extends the capabilities of a server

DOM: Document Object Model. It is an API for valid HTML and well-formed XML documents.

JAXP: Java API for XML Processing

TrAX: Transformation API for XML (now considered a part of JAXP)

HTML: HyperText Markup Language. It is the standard markup language for creating web pages and web applications.

XML: Extensible Markup Language. It is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

1.4 List of Reference Documents

The following references were used to specify this document:

- Verification and validation I – slides;
- OFBiz project;
- Code Inspection Assignment.

2 OFBiz

2.1 The project

Open For Business (OFBiz) is an open source product for the automation of enterprise processes which contains business applications and framework components for ERP (Enterprise Resource Planning), CRM (Customer Relationship Management), E-Business/E-Commerce, SCM (Supply Chain Management), MRP (Manufacturing Resource Planning), MMS/EAM (Maintenance Management System/Enterprise Asset Management).

2.2 UspsMockApiServlet

USPS is a third party shipping company for OfBiz. With USPS web tools api library, shipping rates are gathered. In order to offer this process, UspsMockApiServlet class is created as USPS Webtools API Mock API Servlet which extends HttpServlet.

UspsMockApiServlet is designed based on the process below:

1. Gets a HTTP request from USPS Shipping API Server which came from client.
2. Creates a response and sends to USPS Shipping API Server.

UspsMockApiServlet create base for ZipOrigination, ZipDestination, Pounds, Ounces, Container, Size, Zone and Postage data.

The code flow of UspsMockApiServlet class is like following:

XML of HttpServletRequest request is obtained as xmlvalue. Furthermore, this xmlvalue is read as an xmldocument and named requestDument.

With the document element of the request document and Package string, a List of Element objects is computed and named as packageElementList

If packageElementList is not empty then, create and return an empty xml document with taking RateResponse string as root element name and adding it to the node. The returned document is defined as responseDocument.

The Element interface represents an element in an HTML or XML document.

For each packageElement in the packageElementList followings are defined

1. for the response Document a new child element named Package is created and appended to document element of the response document.
2. By having packageElement's ID, new ID attribute is added to the responsePackageElement.
3. Child elements are created with and appended to the element child node list with following strings: ZipOrigination, ZipDestination, Pounds, Ounces, Container, Size, Zone and Postage.

An output stream named os which accepts output bytes and sends them to some sink is created with as a new byte array output stream in which the

buffer capacity initially 32 bytes.

With parameters of: output stream os, UTF-8 encoding, true value as xml declaration, output will not be indented. Zero number of spaces to indent, responseDocument which is the DOM node is serialized to an OutputStream using JAXP TrAX.

This serialization is made under try block which follows with a catch block for the exception occurs while getting the Javadoc.

In case of the exception, in catch block, this exception is logged into a Configurable Debug logging with mentioning the certain exception and module which named UspsMockApiServlet.

HttpServletResponse response's type is defined as text/xml. A ServletOutputStream named sos is created by the OutputStream of the response. os string is written to the client by sos. Afterwards, sos is flushed.

2.3 ControlServlet

In this part, we are going to analyze the ControlServlet class and its methods. First of all, the ControlServlet is the most important of all request processing. When a request is received, the Control Servlet constitutes an environment for the helper classes. This environment contains setting up an initial session object and storing useful knowledge about the beginning request and setting a reference to the Entity Delegator, Service Dispatcher, and Security Handler for use by the helper classes. The request is then passed to the Request Handler for processing. The Request Handler processes the request and returns to the ControlServlet when finished. When the Control Servlet is first loaded it will create objects used by the web application and store them in the application context (ServletContext). These objects can be found by accessing the property in the context. These objects include: Entity Delegator, Security object, Service Dispatcher, and the Request Handler.

At the beginning of the code, we can see HttpServletRequest and HttpServletResponse which are objects of the Servlet. We use them for transfer of requests and responses between user and system. The HttpServlet class provides methods, such as doGet and doPost, for handling HTTP-specific services. “public void init(ServletConfig config) throws ServletException” called by the servlet container to indicate to a servlet that the servlet is being placed into service. The servlet container calls the init method exactly once after instantiating the servlet. The init method must complete successfully before the servlet can receive any requests. “public void destroy()” called by the servlet container to indicate to a servlet that the servlet is being taken out of service. This method is only called once all threads within the servlet's service method have exited or after a timeout period has passed. After the servlet container calls this method, it will not call the service method again on this servlet. This method gives the servlet an opportunity to clean up any resources that are being held (for example, memory, file handles, threads) and make sure that any persistent state is synchronized with the servlet's current state in memory.

3 CODE INSPECTION

In the following paragraph, the code inspection about the classes, that this team work on, will be presented.

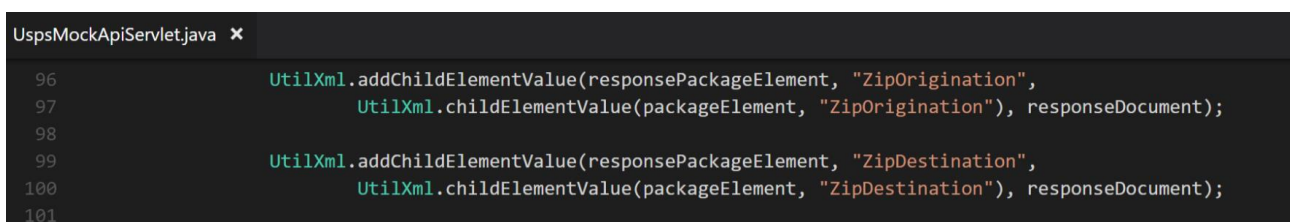
3.1 UspsMockApiServlet

3.1.1 Naming Conventions

All class names, interface names, method names, class variables, method variables, and constants used have meaningful names and do what the name suggests. Class names and interface names are nouns, in mixed case, with the first letter of each word in capitalized. Method names are verbs, with the first letter of each addition word capitalized. Class variables, also called attributes, are mixed case and start with a lower case. Constants are declared using all uppercase with words separated by an underscore. If a one-character variable is used, it is utilized only for temporary “throwaway” variables.

3.1.2 Indentation

Multiple of three or four spaces are used for indentation and made in a compliant way. Usually, consecutive multiples are used for successive levels. In some occasion, this rule is not respected leading to an excessive spacing in the lower level. It happens in line 93, 97, 100, 103, 109 and 112 and an example is in Figure 3.1. The decision, to consider that a serious issue or not, is left to the moderator, because there could be the internal rule to indent at the first dot operator. No tabs are used to indent.



```
UspsMockApiServlet.java x
96         UtilXml.addChildElementValue(responsePackageElement, "ZipOrigination",
97             UtilXml.childElementValue(packageElement, "ZipOrigination"), responseDocument);
98
99         UtilXml.addChildElementValue(responsePackageElement, "ZipDestination",
100             UtilXml.childElementValue(packageElement, "ZipDestination"), responseDocument);
101
```

Figure 3.1 – UspsMockApiServlet (Line 96-101)

3.1.3 Braces

In this class the “Kernighan and Ritchie” style is used and in a correct way. Also all if, while, do-while, try-catch, and statements that have only one statement to execute are surrounded by curly braces.

3.1.4 File Organization

Sometimes lines exceed the 120 characters' limit such as line 83 and 87(Figure 3.2). In this cases lines need a line breaker to use according to the wrapping rules such as:

- when a line is broken at a non-assignment operator the break comes before the symbol. This also applies to the dot operator “.” (*recommended for line 87*);
- a comma (,) stays attached to the token that precedes it. (*recommended for line 83*)

A blank line between two consecutive fields (having no other code between them) is optional. Such blank lines are used as needed to create logical groupings of fields.

```
UspsMockApiServlet.java ✕
82     if (requestDocument == null) {
83         Debug.logError("In UspsMockApiServlet No XML document found in request, quitting now; XML parameter is: " + xmlValue, module);
84         return;
85     }
86
87     List<? extends Element> packageElementList = UtilXml.childElementList(requestDocument.getDocumentElement(), "Package");
88     if (UtilValidate.isEmpty(packageElementList)) {
89
```

Figure 3.2 – UspsMockApiServlet (Line 82-89)

3.1.5 Wrapping Lines

Line break occurs after a comma or an operator. Higher-level breaks are used. Every new statement is aligned with the beginning of the expression at the same level as the previous line

It would be advisable not to insert a line break immediately after the assignment symbol “=” (Figure 3.3 a), but, if necessary, break the content of the assignment according to the lines wrapping rules (Figure 3.3 b).

```
UspsMockApiServlet.java ✕
90     Document responseDocument = UtilXml.makeEmptyXmlDocument("RateResponse");
91     for (Element packageElement: packageElementList) {
92         Element responsePackageElement =
93             UtilXml.addChildElement(responseDocument.getDocumentElement(), "Package", responseDocument);
94         responsePackageElement.setAttribute("ID", packageElement.getAttribute("ID"));
95     }
(a)

UspsMockApiServlet.java ●
90     Document responseDocument = UtilXml.makeEmptyXmlDocument("RateResponse");
91     for (Element packageElement: packageElementList) {
92         Element responsePackageElement =UtilXml
93             .addChildElement(responseDocument.getDocumentElement(), "Package", responseDocument);
94         responsePackageElement.setAttribute("ID", packageElement.getAttribute("ID"));
95     }
(b)
```

Figure 3.3 – UspsMockApiServlet (Line 90-95)

3.1.6 Comments

Comments are not used adequately. Only the public class presents a small comment (line 42) that can be further worked out. Methods should not be necessarily commented since override methods inherit comments from the supertype, but it would be suitable to add `@inheritDoc`. The method `doGet()` could be an exception because it presents a more complex implementation with some controls that should be highlighted.

3.1.7 Java Source Files

The Java source file contains a single public class. The public class is the first class in the file.

The external program interfaces are implemented consistently with what is described in the Javadoc. As said in par.3.1.6, there are no references to Javadoc file because it is inherited from the supertype. Some of source classes' Javadoc files of source classes are not always complete (such as `org.apache.ofbiz.base.util.UtilXml` where every `readXmlDocument` methods have no explanations).

3.1.8 Package and Import Statements

Package statements are the first non-comment statements and import statements follow.

3.1.9 Class and Interface Declarations

The class declarations are in the correct order. Methods are grouped by functionality and the nesting depth does not exceed the third level. The `doGet()` method is quite long but not enough to request a partitioning.

3.1.10 Initialization and Declarations

All Initialization and Declaration rules are fulfilled.

3.1.11 Method Calls

Due to missing information in the Javadoc, the possibility that developer has used wrong methods cannot be examined. For sure each used method presents the correct parameters and returns the correct data type and this last it's properly used.

3.1.12 Collections

All Collections'¹ rules are fulfilled.

3.1.13 Object Comparison

In the line 82 of Figure 3.2 the comparison is done with the “==” operator and not with the equal method due to the assignment of the variable to `null`. In order to use the equal method a `NullPointerException` should be handled.

3.1.14 Output Format

Error messages are comprehensive, but not always provide guidance as to how to correct the problem, probably due to the fact they are printed on a logfile. They are in line 69(Figure 3.4) and 83 (Figure 3.2)

```

UspsMockApiServlet.java x
67      // we're only testing the Rate API right now
68      if (!"Rate".equals(request.getParameter("API"))) {
69          Debug.logError("Unsupported API [" + request.getParameter("API") + "]", module);
70          return;
71      }

```

Figure 3.4 – *UspsMockApiServlet* (Line 67-71)

3.1.15 Computation, Comparisons and Assignments

“Brutish programming” is not used and there are not mathematical operations, so related checks will not be performed. In throw-catch expressions, error conditions are legitimate:

- In line 76 the method used was implemented throw several exceptions (Figure 3.5);
- In Figure 3.6 it’s explained why that exception is used in line 122.

The code is free of any implicit type conversions.

¹ i.e. List, Queue, Set and Map

```

75     try {
76         requestDocument = UtilXml.readXmlDocument(xmlValue, false);
77     } catch (Exception e) {
78         Debug.logError(e, module);
79         return;
80     }

```

Figure 3.5 – UspsMockApiServlet (Line 75-80)

```

117     OutputStream os = new ByteArrayOutputStream();
118     try {
119         Util transformation process.
120     } catch (TransformerException e) {
121         Debug.logInfo(e, module);
122         return;
123     }

```

Figure 3.6 – UspsMockApiServlet (Line 117-125)

3.1.16 Exceptions

All relevant exceptions are caught and, usually, recorded.

3.1.17 Flow of Control

All loops are correctly formed, with the appropriate initialization, increment and termination expressions.

3.2 ControlServlet

3.2.1 Naming Conventions

All class names, interface names, method names, class variables and constants used have meaningful names and do what the name suggests. Class names and interface names are nouns, in mixed case, with the first letter of each word in capitalized. Method names are verbs, with the first letter of each additional word capitalized. Class variables, also called attributes, are mixed case and start with a lower case. Constants are declared using all uppercase with words separated by an underscore. If one-character variables are used, they are utilized only for temporary “throwaway” variables and sometimes they are numbered (i.e. line 255 and 260).

Methods variables should respect the same rules of class variables, but some exception have been found:

- webapp is currently considered a noun but, being a contraction of web application, the correct way to write this variable consists in capitalizing the first letter of each additional word → webAppName (Figure 3.7 – ControlServlet (Line 116-126) line 118);

- rname should be changed in rootName in order to have a meaningful name (Figure 3.7Figure 3.8 line 120) and for the same reason **rd** should be changed in requestDispatcher (Figure 3.8 line 239).

```
ControlServlet.java ✕
116
117     // workaround if we are in the root webapp
118     String webappName = UtilHttp.getApplicationName(request);
119
120     String rname = "";
121     if (request.getPathInfo() != null) {
122         rname = request.getPathInfo().substring(1);
123     }
124     if (rname.indexOf('/') > 0) {
125         rname = rname.substring(0, rname.indexOf('/'));
126     }
```

Figure 3.7 – ControlServlet (Line 116-126)

```
ControlServlet.java ✕
238
239     RequestDispatcher rd = request.getRequestDispatcher(errorPage);
240
```

Figure 3.8 – ControlServlet (Line 238-240)

3.2.2 Indentation

Multiple of three or four spaces are used for indentation and made in a compliant way. Usually, consecutive multiples are used for successive levels. No tabs are used to indent.

3.2.3 Braces

In this class the “Kernighan and Ritchie” style is used and in a correct way. Unfortunately, not all if, while, do-while and try-catch statements, that have only one statement to execute, are surrounded by curly braces such as lines 141,310, 216 and 217.

```
ControlServlet.java ✕
140     request.setAttribute("_CONTROL_PATH_", contextPath + request.getServletPath());
141     if (Debug.verboseOn())
142         Debug.logVerbose("Control Path: " + request.getAttribute("_CONTROL_PATH_"), module);
143
```

Figure 3.9 – ControlServlet (Line 140-143)

```
ControlServlet.java ✕
308     }
309 }
310 if (Debug.timingOn()) timer.timerString("[ " + rname + " (Domain: " + request.getScheme() + "://" + request.getServerName() + ") Request Done", module);
311
```

Figure 3.10 – ControlServlet (Line 308-311)



```

212     Throwable throwable = e.getNested() != null ? e.getNested() : e;
213     if (throwable instanceof IOException) {
214         // when an IOException occurs (most of the times caused by the browser window being closed before the request is received)
215         // the connection with the browser is lost and so there is no need to serve the error page; a message is sent to the client
216         if (Debug.warningOn()) Debug.logWarning(e, "Communication error with the client while processing the request");
217         if (Debug.verboseOn()) Debug.logVerbose(throwable, module);
218     } else {

```

Figure 3.11 – ControlServlet (Line 212-218)

3.2.4 File Organization

Lines often exceed the 120 characters' limit i.e. lines 72, 73, 84, 92, 132, 186, 216, 250, 252, 258, 272, 281, 298, 304, 307 and 310 (Some can be seen in Figure 3.10 and Figure 3.11). In this cases lines need a line breaker to use according to the wrapping rules.

Comments almost never respect that limit and then they should be split across multiple lines.

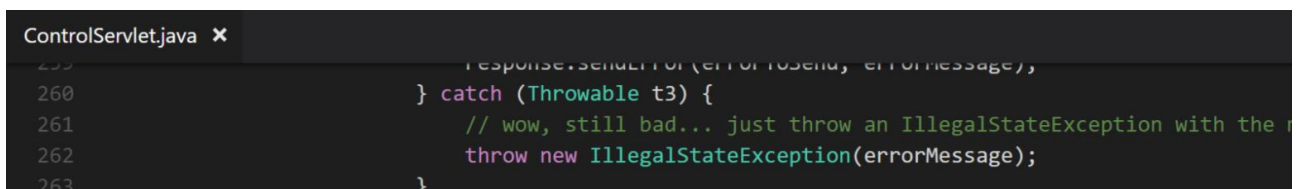
A blank line between two consecutive fields (having no other code between them) is optional. Such blank lines are used as needed to create logical groupings of fields.

3.2.5 Wrapping Lines

Line break occurs after a comma or an operator. Higher-level breaks are used. Every new statement is aligned with the beginning of the expression at the same level as the previous line.

3.2.6 Comments

Comments are used adequately. Sometimes they present unnecessary information like in line 261 (Figure 3.12) and it would be better to ignore subjective opinion and write only how and why that part of code is written.



```

259         response.sendError(400, errorMessage);
260     } catch (Throwable t3) {
261         // wow, still bad... just throw an IllegalStateException with the reason
262         throw new IllegalStateException(errorMessage);
263     }

```

Figure 3.12 – ControlServlet (Line 260-263)

3.2.7 Java Source Files

The Java source file contains a single public class. The public class is the first class in the file.

The external program interfaces are implemented consistently with what is described in the Javadoc. Only two methods have no references: `getRequestHandler()` and `logRequestInfo()` (Figure 3.13 lines 325 and 329).


```

ControlServlet.java ✕
323     }
324
325     protected RequestHandler getRequestHandler() {
326         return RequestHandler.getRequestHandler(getServletContext());
327     }
328
329     protected void logRequestInfo(HttpServletRequest request) {
330         ServletContext servletContext = this.getServletContext();
331         HttpSession session = request.getSession();
332

```

Figure 3.13 – ControlServlet (Line 323-332)

Some Javadoc files of source classes are not always complete (like [org.apache.ofbiz.entity.GenericDelegator](#) where `getDelegatorBaseName`, `getDelegatorName` and other methods have no explanations).

3.2.8 Package and Import Statements

Package statements are the first non-comment statements and import statements follow.

3.2.9 Class and Interface Declarations

The class declarations are in the correct order. Methods are grouped by functionality. Nesting depth exceeds the third level ().

```

ControlServlet.java ✕
236     if (errorPage != null) {
237         Debug.logError("An error occurred, going to the errorPage: " + errorPage, module);
238
239         RequestDispatcher rd = request.getRequestDispatcher(errorPage);
240
241         // use this request parameter to avoid infinite looping on errors in the error page...
242         if (request.getAttribute("_ERROR_OCCURRED") == null && rd != null) {
243             request.setAttribute("_ERROR_OCCURRED", Boolean.TRUE);
244             Debug.logError("Including errorPage: " + errorPage, module);
245
246             // NOTE DEJ20070727 after having trouble with all of these, try to get the page out and as a last resort just s
247             try {
248                 rd.include(request, response);
249             } catch (Throwable t) {
250                 Debug.logWarning("Error while trying to send error page using rd.include (will try response.getOutputStream
251
252                 String errorMessage = "ERROR rendering error page [" + errorPage + "], but here is the error text: " + requ
253                 try {
254                     response.getWriter().print(errorMessage);
255                 } catch (Throwable t2) {
256                     try {
257                         int errorToSend = HttpServletResponse.SC_INTERNAL_SERVER_ERROR;
258                         Debug.logWarning("Error while trying to write error message using response.getOutputStream or respo
259                         response.sendError(errorToSend, errorMessage);
260                     } catch (Throwable t3) {
261                         // wow, still bad... just throw an IllegalStateException with the message and let the servlet conta
262                         throw new IllegalStateException(errorMessage);
263                     }
264                 }
265             }
266         }
267     } else {

```

Figure 3.14 – ControlServlet (Line 236-267)

3.2.10 Initialization and Declarations

All Initialization and Declaration's rules are fulfilled.

3.2.11 Method Calls

Due to missing information in the Javadoc, the possibility that developer has used wrong methods cannot be examined. For sure each used method presents the correct parameters and returns the correct data type and this last it's properly used.

3.2.12 Collections

All Collections'² rules are fulfilled.

3.2.13 Object Comparison

In line 137, 151,154, 167, 170, 176. 179, 242, 268 and 295 the comparisons are done with the “==” operator and not with the equal method due to the assignment of the variables to `null` (Example in Figure 3.15). To use the equal method a NullPointerException should be handled.

Figure 3.15 – ControlServlet (Line 151-156)

3.2.14 Output Format

Error messages are comprehensive, but not always provide guidance as to how to correct the problem, probably due to the fact they are printed on a logfile.

² i.e. List, Queue, Set and Map

3.2.15 Computation, Comparisons and Assignments

“Brutish programming” is not used and there are not mathematical operations, so related checks will not be performed. In throw-catch expressions, error conditions are legitimate. The code is free of any implicit type conversions.

3.2.16 Exceptions

All relevant exceptions are caught and well handled. Moreover they are usually recorded.

3.2.17 Flow of Control

All loops are correctly formed, with the appropriate initialization, increment and termination expressions.

4 EFFORT

<i>Name & Surname</i>	<i>23.01.17</i>	<i>26.01.17</i>	<i>02.02.17</i>	<i>04.02.17</i>	<i>Outside of Group</i>	<i>Total</i>
<i>Marilena COLUCCIA</i>	1h	1h	1h	1h	16h	20h
<i>Burcu CESUR</i>	1h	1h	1h	1h	8h	12h
<i>Mustafa ÇEÇE</i>	1h	1h	1h	1h	8h	12h