Politecnico di Milano

A.A. 2015-2016

Software Engineering 2 Project – Third Assignment

Prof. Raffaela Mirandola

Code **I**nspection **D**ocument

Version 1.0

Christian Zichichi (mat. 840565), Luigi Marrocco (mat. 854884)

January 5, 2016

Table of Contents

1. ASSIGNED METHODS..3

2. FUNCTIONAL ROLE..4

2.1 WebApplication::start..4

2.2 WebApplication::stop..7

2.3 WebApplication::isKeepState..9

2.4 WebApplication::applyCustomizations..11

3. LIST OF ISSUES..14

3.1 WebApplication::start..14

3.2 WebApplication::stop..14

3.3 WebApplication::isKeepState..15

3.4 WebApplication::applyCustomizations..15

4. ADDITIONAL PROBLEMS..16

4.1 WebApplication::start..16

4.2 WebApplication::stop..16

4.3 WebApplication::isKeepState..17

4.4 WebApplication::applyCustomizations..17

6. USED TOOLS..18

7. HOURS OF WORK..18

8. REFERENCES..18

1. ASSIGNED METHODS

The following list represents all the methods we are going to inspect in the assigned class (that is underlined):

1. **Name:** start(ApplicationContext appContext)

   Location: appserver/web/web-glue/src/main/java/com/sun/enterprise/web/WebApplication.java

   Inspector: Christian Zichichi

2. **Name:** stop(ApplicationContext stopContext)

   Location: appserver/web/web-glue/src/main/java/com/sun/enterprise/web/WebApplication.java

   Inspector: Christian Zichichi

3. **Name:** isKeepState(DeploymentContext deployContext , boolean isDeploy)

   Location: appserver/web/web-glue/src/main/java/com/sun/enterprise/web/WebApplication.java

   Inspector: Luigi Marrocco

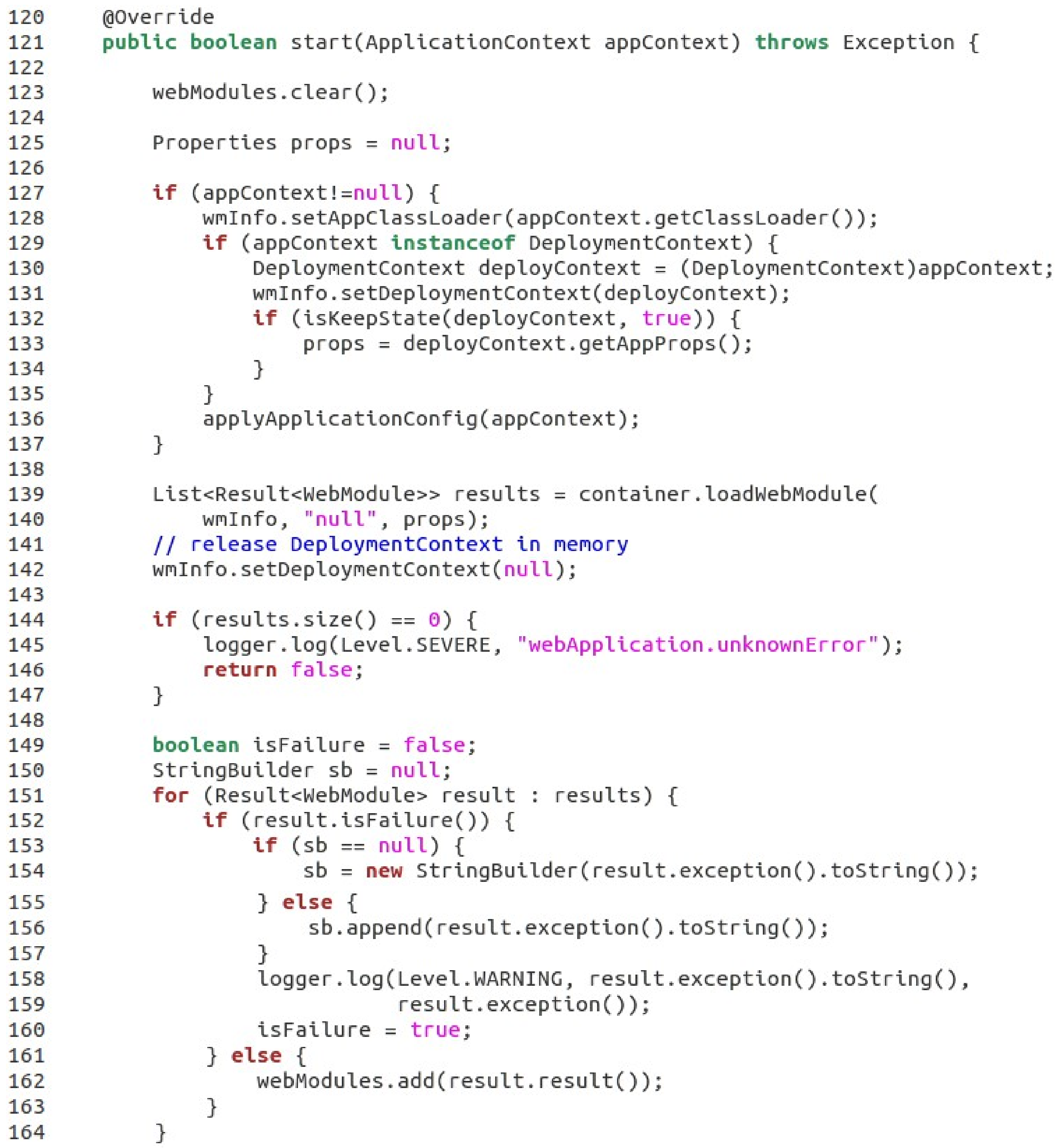
4. **Name:** applyCustomizations( )

   Location: appserver/web/web-glue/src/main/java/com/sun/enterprise/web/WebApplication.java

   Inspector: Luigi Marrocco

2. FUNCTIONAL ROLE

2.1 WebApplication::start



```java
    @Override
    public boolean start(ApplicationContext appContext) throws Exception {

        webModules.clear();

        Properties props = null;

        if (appContext!=null) {
            wmInfo.setAppClassLoader(appContext.getClassLoader());
            if (appContext instanceof DeploymentContext) {
                DeploymentContext deployContext = (DeploymentContext)appContext;
                wmInfo.setDeploymentContext(deployContext);
                if (isKeepState(deployContext, true)) {
                    props = deployContext.getAppProps();
                }
            }
            applyApplicationConfig(appContext);
        }

        List<Result<WebModule>> results = container.loadWebModule(
            wmInfo, "null", props);
        // release DeploymentContext in memory
        wmInfo.setDeploymentContext(null);

        if (results.size() == 0) {
            logger.log(Level.SEVERE, "webApplication.unknownError");
            return false;
        }

        boolean isFailure = false;
        StringBuilder sb = null;
        for (Result<WebModule> result : results) {
            if (result.isFailure()) {
                if (sb == null) {
                    sb = new StringBuilder(result.exception().toString());
                } else {
                    sb.append(result.exception().toString());
                }
                logger.log(Level.WARNING, result.exception().toString(),
                           result.exception());
                isFailure = true;
            } else {
                webModules.add(result.result());
            }
        }
```

```
166         if (isFailure) {
167             webModules.clear();
168             throw new Exception(sb.toString());
169         }
170
171         if (logger.isLoggable(Level.INFO)) {
172             logger.log(Level.INFO, LOADING_APP, new Object[] {wmInfo.getDescriptor().getName(),
173         }
174
175         return true;
176     }
```

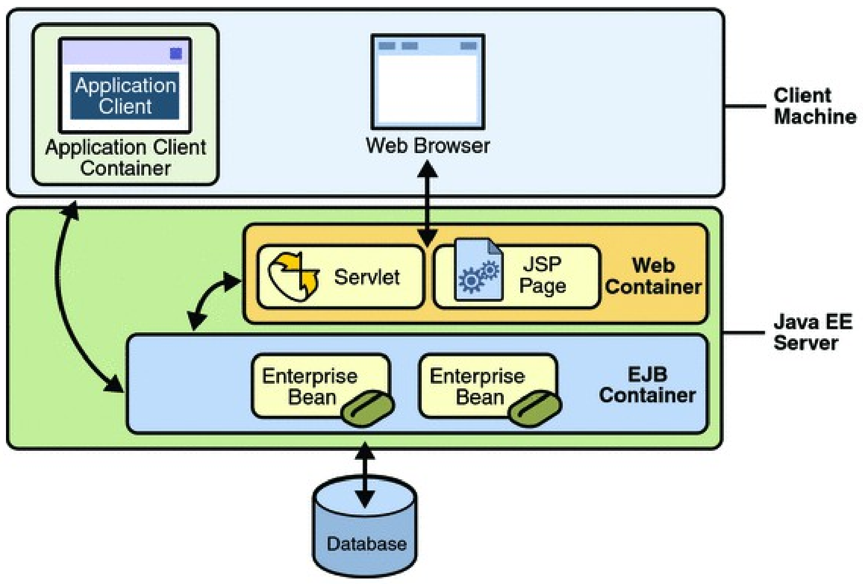
This method is contained in the class WebApplication.java and there is no documentation at all (javadoc) about it, so to fully understand its functional role we have to consider also some external lines of code (import statements and class variables) and, more generally, the class in its entirety. Fortunately, all the used names are meaningful, so we can begin observing the annotation at line 120 that checks if the method is an override, implying that it actually is. To find out what method it overrides, it suffices to check the documentation (at http://glassfish.pompel.me/) about the interface implemented by the class

```
70 public class WebApplication implements ApplicationContainer<WebBundleDescriptorImpl> {
```

and, to better understand the method, the instance variables defined in the class and the class constructor:

```
108     private final WebContainer container;
109     private final WebModuleConfig wmInfo;
110     private Set<WebModule> webModules = new HashSet<WebModule>();
111     private final org.glassfish.web.config.serverbeans.WebModuleConfig appConfigCustomizations;
112
113     public WebApplication(WebContainer container, WebModuleConfig config,
114             final ApplicationConfigInfo appConfigInfo) {
115         this.container = container;
116         this.wmInfo = config;
117         this.appConfigCustomizations = extractCustomizations(appConfigInfo);
118     }
```

We discover that the method's main functionality is to initialize and deploy a web container for a given Web Application (that manages the execution of JSP page and servlet components for Java EE applications. Web components and their container run on the Java EE server). Containers are the interface between a component and the low-level platform-specific functionality that supports the component. Before a web, enterprise bean, or application client component can be executed, it must be assembled into a Java EE module and deployed into its container. The assembly process involves specifying container settings for each component in the Java EE application and for the Java EE application itself.



Code Analysis:

Line 121 – The method is declared public and boolean and receives an Application Context (set of java beans used in a given web application) in input.

Lines 123-125 – Set of deployed web modules and related properties' variable (used to transfer information between deployment clients and server) initialization. In the Java EE architecture, a web module is the smallest deployable and usable unit of web resources. A web module contains web components and static web content files, such as images, which are called web resources. A Java EE web module corresponds to a web application.

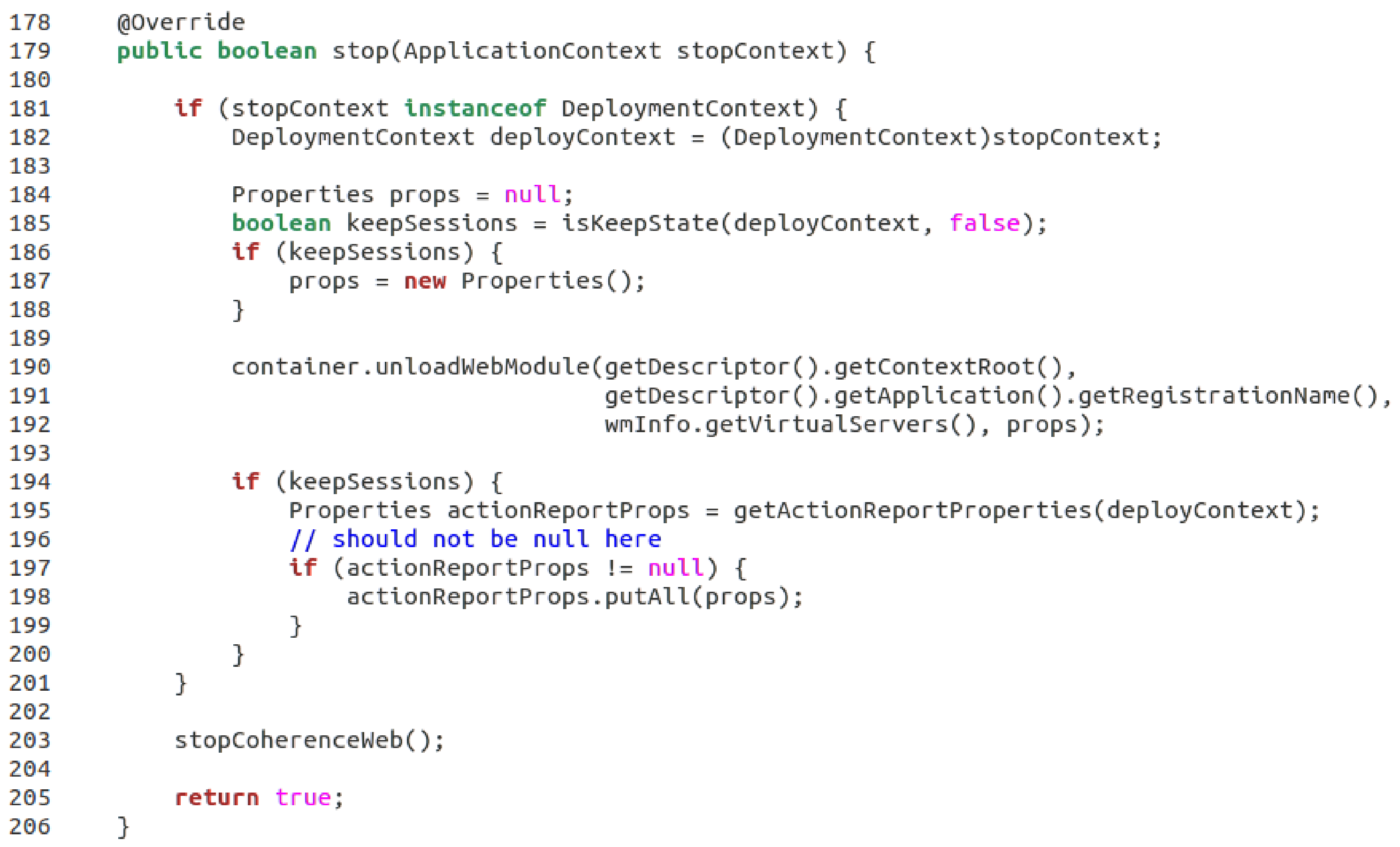
Lines 127-137 – If the Application Context is valid, proper configuration parameters, which are required to create and install the web application into the server run-time, are set. If an instance of the application is already deployed, deployment command parameters are retrieved. If session data across redeployments must be preserved, they are retained in the properties' variable (section 2.3, method WebApplication::isKeepState).

Lines 139-142 – Creation and configuration of a web module for each virtual server that the web module is hosted under. As the comment at line 141 states, after the configuration the Deployment Context is released in memory.

Lines 144-147 – If the web application is not loaded, a log with an error message is created and the method returns false

Lines 149-176 – For each virtual server that the web module is hosted under, these lines check if there has been a failure in the deployment. If it is so, a string containing only the faulty cases (each failure is appended) and a log with the level WARNING for each failure are created. If there is no failure for a web module, then it is added to the list of deployed web modules. One failure is enough to cause the throw of an exception (that shows all the failures on the created string) and the set of deployed web modules to be cleared. If no exceptions are thrown (if the logged message level is INFO it means that the application is loading correctly), a log with all the information regarding the loaded web application is created. Eventually, if no exceptions occur, the method returns true.

2.2 WebApplication::stop



```
178    @Override
179    public boolean stop(ApplicationContext stopContext) {
180
181        if (stopContext instanceof DeploymentContext) {
182            DeploymentContext deployContext = (DeploymentContext)stopContext;
183
184            Properties props = null;
185            boolean keepSessions = isKeepState(deployContext, false);
186            if (keepSessions) {
187                props = new Properties();
188            }
189
190            container.unloadWebModule(getDescriptor().getContextRoot(),
191                                      getDescriptor().getApplication().getRegistrationName(),
192                                      wmInfo.getVirtualServers(), props);
193
194            if (keepSessions) {
195                Properties actionReportProps = getActionReportProperties(deployContext);
196                // should not be null here
197                if (actionReportProps != null) {
198                    actionReportProps.putAll(props);
199                }
200            }
201        }
202
203        stopCoherenceWeb();
204
205        return true;
206    }
```

This method is the counterpart of the previous one and its very similar to it. Like WebApplication::stop it is an override of an existing method and its function is to clear and undeploy a web container for a deployed web application.

Code Analysis:

Line 179 – The method is declared public and boolean and receives as input the Application Context (set of java beans used in a given web application) of the web application to undeploy.

Lines 181-188 – If the application is deployed, its Context is considered as Deployment Context. If session data must be preserved across redeployments, a properties object is created.

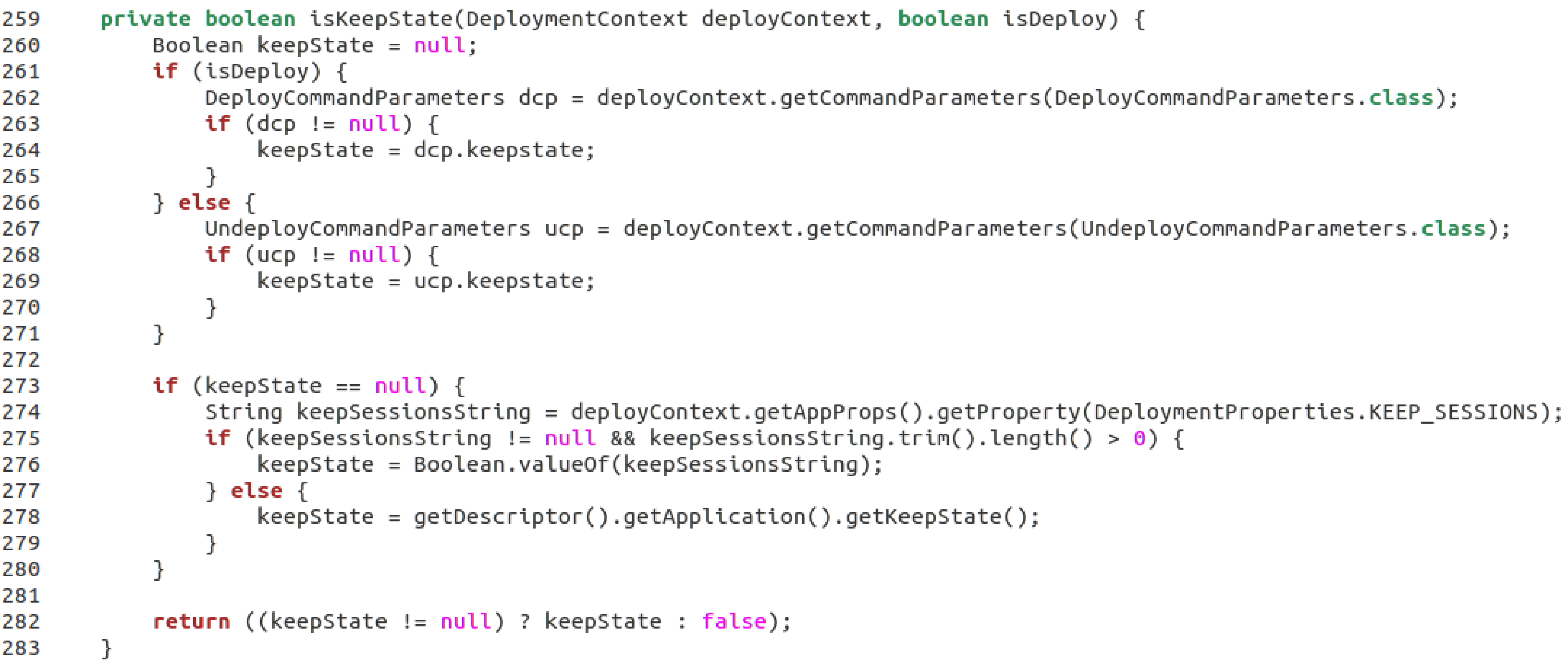
Lines 190-192 – The web module is unloaded from each virtual server that it is hosted under.

Lines 194-201 – If session data must be preserved across redeployments, properties of the action report are retrieved from the deployed web application. Obviously, as the comment states, these properties should exist for a session and this is verified with an additional check. An action report is an interface allowing any type of server side action like a service execution, a command execution to report on its execution to the originator of the action. Implementations of this interface should provide a good reporting experience based on the user's interface like a browser or a command line shell.

Line 203 – Coherence*Web is shut down since the web application is being undeployed. Coherence*Web is an HTTP session management module dedicated to managing session state in clustered environments. Oracle Coherence is a proprietary Java-based in-memory data grid, designed to have better reliability, scalability and performance than traditional relational database management systems. While HTTP session replication is a powerful way to enable applications to continue running even in the event of server failure, the main issue is that the HTTP session information is held in the JVM memory of the running Application server. If the web application has lots or users, and/or large HTTP session objects, the server could soon run out of memory and therefore stop it from scaling well. A solution to this problem is to off load the HTTP session management into a cache (Coherence).

Lines 205-206 – Eventually, if no exceptions are thrown (from the called methods), this method returns true (the web application is undeployed).

2.3 WebApplication::isKeepState



```java
private boolean isKeepState(DeploymentContext deployContext, boolean isDeploy) {
    Boolean keepState = null;
    if (isDeploy) {
        DeployCommandParameters dcp = deployContext.getCommandParameters(DeployCommandParameters.class);
        if (dcp != null) {
            keepState = dcp.keepstate;
        }
    } else {
        UndeployCommandParameters ucp = deployContext.getCommandParameters(UndeployCommandParameters.class);
        if (ucp != null) {
            keepState = ucp.keepstate;
        }
    }

    if (keepState == null) {
        String keepSessionsString = deployContext.getAppProps().getProperty(DeploymentProperties.KEEP_SESSIONS);
        if (keepSessionsString != null && keepSessionsString.trim().length() > 0) {
            keepState = Boolean.valueOf(keepSessionsString);
        } else {
            keepState = getDescriptor().getApplication().getKeepState();
        }
    }

    return ((keepState != null) ? keepState : false);
}
```

A Java EE application maintains session state in several places. The most widely known places are HTTP Web sessions and stateful Enterprise JavaBeans (EJB) sessions. Starting with Java EE 6 and the introduction of timer services, persistent EJB timers are also associated with session data. The problem with such data is that it is lost whenever an application is redeployed. Because iterating over the development of an application requires frequent redeployments, a significant productivity loss is suffered. This results from the need to manually replay common use-case steps, such as logging in to an application, filling in some data, and performing certain steps to get back to the state before the redeployment. GlassFish offers an option to preserve session data across redeployments. It is turned off by default, but it can be explicitly enabled it by passing the –keepstate=true flag to the redeploy command.

This method returns the value of the keepstate flag of the relative deployment context passed as input.

Code Analysis:

Lines 261-271 – If the Deployment Context received as input is running (the boolean variable isDeploy determines this), the relative deployment command parameters are saved. After that, if these parameters are not null, their keepstate is assigned to the variable keepState. In the same way, if the Deployment Context is not running the Undeployment command parameters are saved and. if these parameters are not null, their keepstate is assigned to the variable keepState.

Lines 273-280 – If the command parameters are not present (supposing that the session data are cleared) the keepstate is taken from the properties of the Deployment Context (That allows individual deployers' implementations to store some information that should be available upon server restart). If it is not null, the keepstate is assigned to the variable keepState. If even the properties of the Deployment Context are set up, the program takes the keepstate directly from the deployment descriptor of the application which the Deployment Context belongs to.

Line 282 – Eventually, the method either returns the value of the keepstate found or false if the keepstate is not found (supposing that the deployment context passed as input is wrong).

2.4 WebApplication::applyCustomizations

```
        /**
         * Applies the set of customizations to the descriptor's set of
         * items.
         */
        void applyCustomizations () {
            boolean isFiner = logger.isLoggable(Level.FINER);

          nextCustomization:
            for (U customization : customizations) {
                /*
                 * For each customization try to find a descriptor item with
                 * the same name.  If there is one, either ignore the descriptor
                 * item (if that is what the customization specifies) or override
                 * the descriptor items'a value with the value from the
                 * customization.
                 */
                for (Iterator<T> it = descriptorItems.iterator(); it.hasNext();) {
                    T descriptorItem = it.next();
                    String dItemName = getName(descriptorItem);
                    String customizationItemName = getCustomizationName(customization);
                    if (dItemName.equals(customizationItemName)) {
                        /*
                         * We found a descriptor item that matches this
                         * customization's name.
                         */
                        if (isIgnoreDescriptorItem(customization)) {
                            /*
                             * The user wants to ignore this descriptor item
                             * so remove it from the descriptor's collection
                             * of items.
                             */
                            it.remove();
                            if (isFiner) {
                                logger.log(Level.FINER,
                                        IGNORE_DESCRIPTOR,
                                        new Object[]{descriptorItemName, getName(descriptorItem)});
                            }
                        } else {
                            /*
                             * The user wants to override the setting of this
                             * descriptor item using the customized settings.
                             */
                            String oldValue = getValue(descriptorItem); // for logging purposes only
                            try {
                                setDescriptorItemValue(descriptorItem, customization);
                                if (isFiner) {
                                    logger.log(Level.FINER, OVERIDE_DESCRIPTOR,
                                            descriptorItemName + " " +
                                            getName(descriptorItem) + "=" +
                                            oldValue +
                                            " with " + toString(customization));
                                }
                            } catch (Exception e) {
                                logger.warning(toString(customization) + " " + e.getLocalizedMessage());
                            }
                        }
```

```
502                         /*
503                          * We have matched this customization with a descriptor
504                          * item, so we can skip to the next customization.
505                          */
506                         continue nextCustomization;
507                     }
508                 }
509                 /*
510                  * The customization matched no existing descriptor item, so
511                  * add a new descriptor item.
512                  */
513                 try {
514                     T newItem = addDescriptorItem(customization);
515                     if (isFiner) {
516                         logger.log(Level.FINER,
517                                 CREATE_DESCRIPTOR,
518                                 descriptorItemName + getName(newItem) + "=" + getValue(newItem));
519                     }
520                 } catch (Exception e) {
521                     logger.warning(toString(customization) + " " + e.getLocalizedMessage());
522                 }
523             }
524         }
525     }
```

This method is contained in a nested abstract class within the considered class WebApplication. Since this nested class is non-static, it is called inner class. Even if the entire inner class is not reported in this document, it is important to observe that there are no problems in accessing the variables of its enclosing class because they are declared as static.

As the initial comment reports, the functional role of this method (declared void) is to apply the set of customizations to the descriptor's set of items.

Code Analysis:

Line 451 – The boolean variable isFiner becomes true if a message of level FINER would actually be logged by this logger, false otherwise. The level FINER indicates a fairly detailed tracing message.

Lines 453-523 – For each customization try to find a descriptor item with the same name. If there is one, either ignore the descriptor item (if that is what the customization specifies) or override the descriptor items' value with the value from the customization.

Lines 462-508 – Descriptors are iterated with the for loop.

Lines 464-465 – These lines get the name of the descriptor and the customization.

Lines 466-507 – If a descriptor item matches with the customization's name we have two possibilities:

- Lines 471-483 – the user wants to ignore this descriptor item so it is removed from the descriptor's collection of items. If the logged message level is FINER, this ignore operation is logged.
- Lines 483-501 – since the user does not want to ignore this descriptor the settings of this descriptor item are overridden using the customized settings. If the logged message level is FINER, this override operation is logged. If an error occurs, a warning message is logged.

Lines 513-522 – The customization matched no existing descriptor item, so a new descriptor item is added. These lines also write in the log if the message level of the logger is FINER. if an error occurs, a warning message is logged.

3. LIST OF ISSUES

All issues are found following the Checklist for inspections of Java code by Christopher Fox.

3.1 WebApplication::start

Checklist[13] – Line 172 has 126 out of 80 characters, due to many arguments passed to a method.

Checklist[14] – Line 172 has 126 out of 120 characters, due to many arguments passed to a method.

Checklist[15] – Line 139, Line break after an opening brace.

Checklist[18] – There is only one comment (Line 141) that explains the following line. No comment is present to adequately explain what the method is doing.

Checklist[23] – No javadoc is present for this method (and in general for the whole class WebApplication). For this reason it has been particularly hard to analyse the functional role of this method, having no explanations on how it works (even though meaningful names are used).

Checklist[33] – Lines 125, 149, 150 do not respect this point. Declarations do not appear at the beginning of blocks.

Checklist[40] – Lines 127 and 153. Objects are compared with "!=" and "==" and not with "!equals()" and "equals()".

3.2 WebApplication::stop

Checklist[16] – Lines 190-192. Lower levels break are used instead of a higher level ones

Checklist[18] – There is only one comment (Line 196) regarding the following line. No comment is present to adequately explain what the method is doing.

Checklist[23] – No javadoc is present for this method (and in general for the whole class WebApplication). For this reason it has been particularly hard to analyse the functional role of this method, having no explanations on how it works (except for the meaningful names that are used)

Checklist[40] – Line 197. Objects are compared with "!=" and not with "!equals()".

3.3 WebApplication::isKeepState

Checklist[13] – Line 262 has 96 out of 80 characters, Line 267 has 100 out of 80 character, Line 274 has 113 out of characters.

Checklist[18] – No comment is present to adequately explain what the method is doing.

Checklist[23] – No javadoc is present for this method (and in general for the whole class WebApplication). For this reason it has been particularly hard to analyse the functional role of this method, having no explanations on how it works (except for the meaningful names that are used).

Checklist[33] – Lines 262, 267 and 274. Declarations do not appear at the beginning of blocks.

Checklist[40] – Lines 263, 268 and 275. Objects are compared with "!=" and not with "!equals()".

3.4 WebApplication::applyCustomizations

Checklist[1] – Line 492, there is a misspelt constant (OVERIDE_DESCRIPTOR that should be called OVERRIDE_DESCRIPTOR).

Checklist[8] – Line 454, only 2 spaces are used to indent instead of 4 (as it is done in all the other lines of the method).

Checklist[16] – Lines 479-481, 492-496 and 516-518. Lower levels break are used instead of a higher level ones.

Checklist[18] – Line 459, there is a typo in a sentence of the comment.

Checklist[23] – Javadoc are present for this method (outside it), but not sufficient to explain in detail all the used variables. Fortunately, there are comments inside the method useful to analyse its functional role and the used names are meaningful.

4. ADDITIONAL PROBLEMS

4.1 WebApplication::start

- Lines 125 and 150 – Assigning a ‘null‘ value to a new declared variable is useless, since this is the default behavior of Java.
- Lines 149-150 – These declarations and initializations should be put in the top of the body of the method since they do not depend from the result of previous instructions.

4.2 WebApplication::stop

- Lines 184 – Assigning a ‘null‘ value to a new declared variable is useless, since this is the default behavior of Java.
- Lines 196-199 – Instead of assuming that actionReportProps should not be null as the comment reports, an else block should be added after the if block to manage the opposite case or, alternatively, a specific exception should be thrown if the null case occurs when it is not expected to.

4.3 WebApplication::isKeepState

- Lines 260 – Assigning a ‘null‘ value to a new declared variable is useless, since this is the default behavior of Java.

4.4 WebApplication::applyCustomizations

- Lines 453 and 506 – There is a label and a “continue” expression that jump to it. This construct is usually used when we want to skip two iterations of a for loop. In this case the method wants to skip to the next customization when one with a proper descriptorItem is found. Though, the method doesn't skip directly to the next customization, but repeat the entire process from the first customization. Therefore, it would be more appropriate to use a simple “break” expression because only one iteration has to be skipped, not two.

- Lines 499 and 521 – The method getLocalizedMessage() should be used to produce a locale-specific message changing the language of the message that will be invoked. In order to do it this method should be overridden but here it is not. In fact, the default implementation returns the same result as the method getMessage() does. Therefore, it is more correct to use getMessage() instead of getLocalizedMessage().

- Methods and properties of the current class must be referenced within the class using the “this” prefix. This should be done to improve the readability and distinguish the manipulation of methods and properties of the current class from those of the superclass of the current class. Moreover, it is also useful for distinguish between properties of the current class and local variables. This behavior is not followed in this method.

6. USED TOOLS

- **LibreOffice Writer 5.0:** to redact and format the document
- **Gedit text editor:** to inspect the code
- **Eclipse:** to inspect the code

7. HOURS OF WORK

This is the division of the tasks with the relative hours of work:

- **Christian Zichichi - 15.5 hours**

  Assigned methods: *WebApplication::start* and *WebApplication::stop*

- **Luigi Marrocco - 13 hours**

  Assigned methods: *WebApplication::isKeepState* and *WebApplication::applyCustomizations*

8. REFERENCES

Here is a short list of the references for this document:

- Checklist for inspections of Java code by Christopher Fox
- Slides of the Software Engineering 2 course (from the Beep Platform)
- http://glassfish.pompel.me/
- http://docs.oracle.com/
- https://glassfish.java.net