
A.Y. 2015/2016

Software Engineering 2: “My Taxi Service”

Code Inspection Document

Version 1.1

Ivan Antozzi(790962), Riccardo Giambona(788904)

3 February 2016

# Assigned Classes

## Methods:

### Name:

deployWARs( File appBase , String [ ] files )

### Start Line:

662

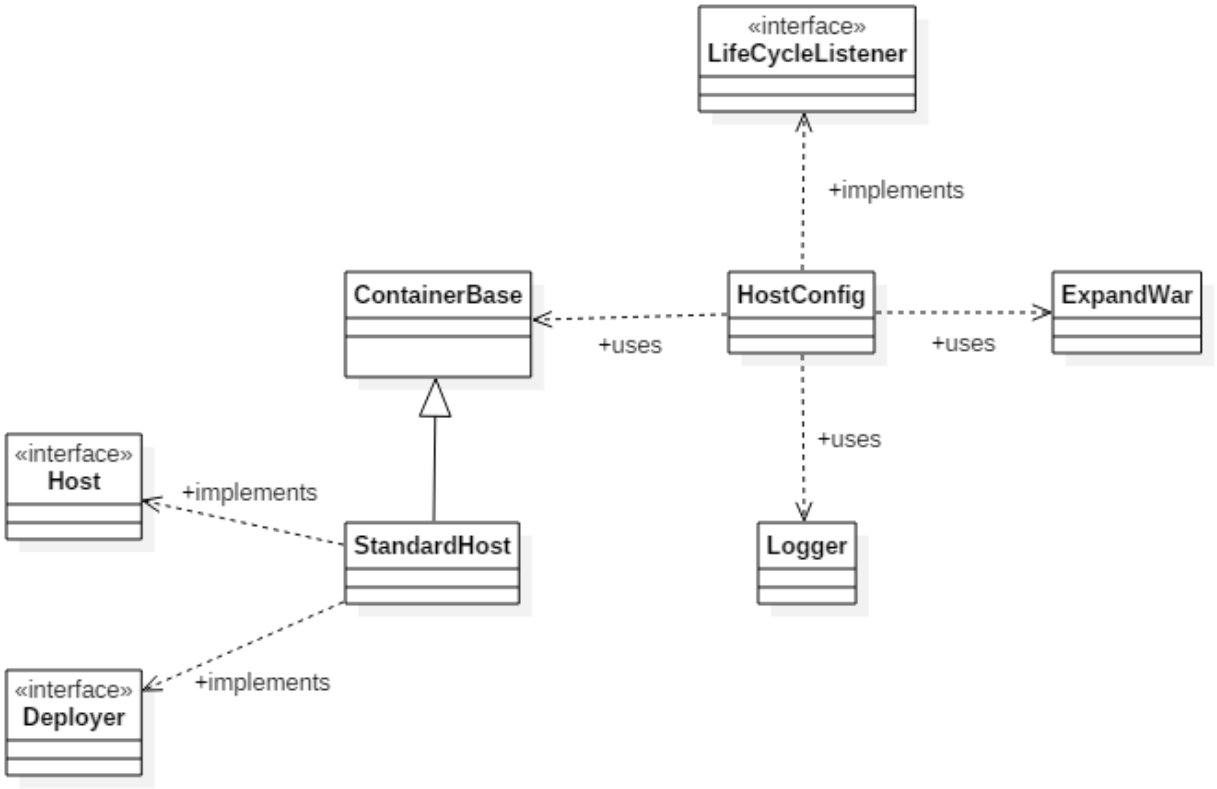
### Location:

appserver/web/web-core/src/main/java/org/apache/catalina/startup/HostConfig.java

### Package:

org.apache.catalina.startup

## Functional Role of Assigned Classes




Host Config is our assigned class and it is a class that manages the server start and stop. To do so it implements an interface whose type is LifeCycleListener. We have understood this by reading the code documentation (that we will report below) of the Host Config and the ones of other classes related to the HostConfig class directly or not,deriving also the uml diagram shown above. The Host Config class also manages the deploy of WARs, that is managed particularly by our assigned method “deployWARs”.

## Classes UML documentation

### LifeCycleListener

This is the interface that we said before that is implemented in HostConfig class to manage the server (or in general a component) start and stop,but this interface also manages other events than this two stated before (but the other events are not better specified).

**Official documentation**

“Interface defining a listener for significant events (including "component start" and "component stop" generated by a component that implements the Lifecycle interface.”

## ExpandWar

This is a class that is used also in our assigned method and it’s used to expand a War file in its internal folder structure (it will be explained better in the code description and analysis below) creating this structure also in the server fileSystem.

**Official documentation**

“Expand out a WAR in a Host's appBase.”

## Logger

This class is used to log important operation result that are made in the HostConfig methods,but especially in our assigned method,this is used to log errors that can occur in the execution of the code and that are caught in the catch branches.

**Offical documentation**

“A Logger is a generic interface for the message and exception logging methods of the ServletContext interface. Loggers can be attached at any Container level, but will typically only be attached to a Context, or higher level, Container.”

## Summary of the usage of the remaining 4 classes

For the other 4 classes we will report the documentation for each one of them,but we will write a summary of what we understood of their usage by the code that we analyzed:
In our code there is a class variable of type Host that we think is an object that contains each single deployed app. To deploy a web app the “install” method exposed by the Deployer interface (casted from the host variable of type Host) is called and we think that this method will launch the web app (contained in a war file) in the host context.

### Container (summarized explanation)

**Official documentation**

“A Container is an object that can execute requests received from a client, and return responses based on those requests.”

### Deployer

**Official documentation**
“A Deployer is a specialized Container into which web applications can be deployed and undeployed. Such a Container will create and install child Context instances for each deployed application. The unique key for each web application will be the context path to which it is attached.”

### Standard Host

**Official documentation**
“Standard implementation of the Host interface. Each child container must be a Context implementation to process the requests directed to a particular web application.”

### Host

**Official documentation**
**“**A Host is a Container that represents a virtual host in the Catalina servlet engine. It is useful in the following types of scenarios:You wish to use Interceptors that see every single request processed by this particular virtual host.You wish to run Catalina in with a standalone HTTP connector, but still want support for multiple virtual hosts.In general, you would not use a Host when deploying Catalina connected to a web server (such as Apache), because the Connector will have utilized the web server's facilities to determine which Context (or perhaps even which Wrapper) should be utilized to process this request.The parent Container attached to a Host is generally an Engine, but may be some other implementation, or may be omitted if it is not necessary.The child containers attached to a Host are generally implementations of Context (representing an individual servlet context).”

## Assigned Methods

### DeployWAR Summary

This section is useful to shortly explain what this method does. This method takes a list of files pathnames that are the pathnames of the web application packages that need to be deployed in the app directory, which is app base.
To understand what the method does we read the documentation of this method in the javaDoc and the documentation of all submethods called by this method and their relative classes (stated above).

### Note about the WAR file structure

To better understand the functionality of the code,it is necessary to understand the structure of a war file:

A war file is structured inside as a folder and so it has an internal file system structure. This can be seen just changing the WAR extension of the file into a ZIP extension. Inside this WAR there are a WEB-INF folder and a META-INF folder. This explaines why the method checks for these two folders at the start of the method. Also this explains why it is possible to expand a WAR into its folder internal structure.

## DeployWAR Commented Code

[AC]= Added comments (these are the comments that we added and used to understand what the method does)

```
    /**
     * Deploy WAR files.
     */
    protected void deployWARs(File appBase, String[] files) {
    //[AC]scan all the specified files of the WARs to be deployed.
        for (int i = 0; i < files.length; i++) {
      //[AC]checks if the files[i] is not a meta inf or web inf directory,
      because we are interested only in WAR files.
            if (files[i].equalsIgnoreCase("META-INF"))
                continue;
            if (files[i].equalsIgnoreCase("WEB-INF"))
                continue;
            //[AC]checks if the WAR hasn't already been deployed.
            if (deployed.contains(files[i]))
                continue;
            File dir = new File(appBase, files[i]);
            //[AC]checks if the files[i] is a path of a WAR file and it's not
            invalid.
            if (files[i].toLowerCase(Locale.ENGLISH).endsWith(".war") &&
            dir.isFile()
            && !invalidWars.contains(files[i])) {
            //[AC]after the checks it add files[i] to the deployed files list.
                deployed.add(files[i]);

                // Calculate the context path and make sure it is unique
                String contextPath = "/" + files[i];
                int period = contextPath.lastIndexOf(".");
                if (period >= 0)
                    contextPath = contextPath.substring(0, period);

                // Check for WARs with /../ /./ or similar sequences in the name
                if (!validateContextPath(appBase, contextPath)) {
                    log.log(Level.SEVERE, INVALID_WAR_NAME_EXCEPTION, files[i]);
                    invalidWars.add(files[i]);
                    continue;
                }
                //[AC] Check if i'm not in the ROOT directory
                if (contextPath.equals("/ROOT"))
                    contextPath = "";
                  //[AC] Check if the host hasn't a WAR file with the specified
                  context path.
                if (host.findChild(contextPath) != null)
                    continue;

                // Checking for a nested /META-INF/context.xml
               //[AC] It extracts the nested META-INF contained in the WAR file
            and saves it in an other XML.
                //[AC] This XML's name is context.xml
               //[AC] For the detailed explanation see the note one below.

                JarFile jar = null;
                JarEntry entry = null;
                InputStream istream = null;
                BufferedOutputStream ostream = null;
```

```
File xml = new File
    (configBase, files[i].substring
     (0, files[i].lastIndexOf(".")) + ".xml");
if (!xml.exists()) {
    try {
        jar = new JarFile(dir);
        entry = jar.getJarEntry("META-INF/context.xml");
        if (entry != null) {
            istream = jar.getInputStream(entry);
            ostream =
                new BufferedOutputStream
                (new FileOutputStream(xml), 1024);
            byte buffer[] = new byte[1024];
            while (true) {
                int n = istream.read(buffer);
                if (n < 0) {
                    break;
                }
                ostream.write(buffer, 0, n);
            }
            ostream.flush();
            ostream.close();
            ostream = null;
            istream.close();
            istream = null;
            entry = null;
            jar.close();
            jar = null;
         //[AC] It deploys the descriptor just created in the
        config_base directory.
            deployDescriptors(configBase(), configBase.list());
            return;
        }
    } catch (IOException e) {
        // Ignore and continue
    } finally {
        //[AC] It manages the closing of all the opened streams.
        if (ostream != null) {
            try {
                ostream.close();
            } catch (Throwable t) {
                ;
            }
            ostream = null;
        }
        if (istream != null) {
            try {
                istream.close();
            } catch (Throwable t) {
                ;
            }
            istream = null;
        }
        entry = null;
        if (jar != null) {
            try {
                jar.close();
            } catch (Throwable t) {
                ;
            }
            jar = null;
        }
    }
```

```
}
  //[AC] It checks if this war needs to be unpacked into its
  internal structure or it has
//[AC] to be deployed as it is.
if (isUnpackWARs()) {
  //[AC] Expanded WAR choice.
 // Expand and deploy this application as a directory
    if (log.isLoggable(Level.FINE)) {
        log.log(Level.FINE, EXPANDING_WEB_APP, files[i]);
    }
    URL url = null;
    String path = null;
    try {
        url = new URL("jar:file:" +
                      dir.getCanonicalPath() + "!/");
       //[AC] Since this is the case that we want to expand
       this WAR into a folder we need
        //[AC] to join the host path to the WAR file name in
       order to create a new path to reach
        //[AC] the root folder of the expanded war.
        path = ExpandWar.expand(host, url);
    } catch (IOException e) {
        // JAR decompression failure
         log.log(Level.WARNING, EXPANDING_WEB_APP_EXCEPTION,
       files[i]);
        continue;
    } catch (Throwable t) {
        String msg =
       MessageFormat.format(rb.getString(EXPANDING_WEB_APP_ARCH
       IVE_EXCEPTION),files[i]);
        log.log(Level.SEVERE, msg, t);
        continue;
    }
    try {
        if (path != null) {
            url = new URL("file:" + path);

         //[AC] This line is used to effectively deploy the
       expanded WAR at the specified URL
       //[AC] and with its context information, that are in the
       context path.
            ((Deployer) host).install(contextPath, url);
        }
    } catch (Throwable t) {
        String msg =
       MessageFormat.format(rb.getString(EXPANDING_WEB_APP_ARCH
       IVE_EXCEPTION), files[i]);
        log.log(Level.SEVERE, msg, t);
    }

} else {
  //[AC] WAR as a file choice.
  // Deploy the application in this WAR file
    if (log.isLoggable(Level.INFO)) {
        log.log(Level.INFO, DEPLOYING_WEB_APP_ARCHIVE,files[i]);
    }
    try {
        URL url = new URL("file", null,
                          dir.getCanonicalPath());
        url = new URL("jar:" + url.toString() + "!/");
       //[AC] This line is used to effectively deploy the
       expanded WAR at the specified URL
```

```
                    //[AC] and with its context information, that are in the
                    context path.
                    ((Deployer) host).install(contextPath, url);
                } catch (Throwable t) {
                    String msg =
                    MessageFormat.format(rb.getString(ERROR_DEPLOYING_WEB_AP
                    P_ARCHIVE_EXCEPTION),
                                                        files[i]);
                    log.log(Level.SEVERE, msg, t);
                }
            }
        }
    }
}
```

## Checklist

1. **Assigned Method**

   Our assigned method name is meaningful and suggests what the method does. About the variables, we suggest, at line 672, to change the name "dir" with fileSystemElement for example, because this variable can represent a file or a directory, depending on the path specified in the costructor.

   **Class**

   All class variables and methods names are meaningful and do what the name suggests.
   We could suggest to change:
   -lifecycleEvent to manageLifecycleEvent
   -appBase (method) to getAppBase
   -configBase (method) to getConfigBase
   -validateContextPath to checkValidPath

2. We didn't find any error at this point. All the variables with one letter in the name are used in the cycles or in the catch branches.
3. The class name is HostConfig, so it respect the naming convention.
4. The interface, implemented by this class, is LifecycleListener, its name respect this notation.
5. Our method is called deployWARs. It doesn't respect the notation, because it should be deployWars.
6. All class variables respect this notation.
7. At line 92 and 93 there are two constant private objects (log and rb) that have the name in lower case,so with this notation (that we suppose has to be applied even to constant objects) these

two names are wrong and should be LOG and RB respectively.
All other constants declared respect this notation.

8. There are some lines that don't respect the three or four spaces for indentation. These lines are: 674, 703, 710, 711, 712, 713, 773, 781, 792, 804, 809.
9. No tabs are used to indent.
10. The bracing style used is the "Kernigan and Richie" style and it's consistent for all if in the deployWarMethod.
11. The lines in which this condition is not respected are: 666, 668, 670,681,691,693.
12. At the beginning of the class there isn't a comment that states the start of the declaration of constant variables. Anyway,except for this, in our class each section is well separated from the others.
13. There are some lines that exceed 80 characters. These lines are: 673, 686,777, 780, 791,800, 808.
14. There are no lines that exceed 120 characters.
15. The lines that are not respecting this convention are: 673, 702, 711.
16. The line that is not respecting this convention is line 702.
17. In each block of our method all the code lines are alligned with the previous line of the same block.
18. Our method is well commented, maybe we could suggest to add some comments that we wrote in the DeployWAR Commented code section.
19. There is no code commented in our method, so this point has no sense for our analysis.
20. The HostConfig.java file contains only one class, that is HostConfig.
21. It's verified because in our class file we have only one class.
22. Since the HostConfig class implements the LifecycleListener interface, it's present the lifecycle event method, that manages the start and the stop event of the host and the checkEvent. In the javadoc, instead, it's written that only the startEvent is handled, so there is a inconsistency between the implementation and the documentation and the stop event and check event are not documented.
23. The javadoc is complete, there aren't classes or method declared that don't appear in the javaDoc. We were interested to these methods, that appear in the javaDOC. These are:
    - deployDescriptors
    - Expandwar.expand
    - Deployer class
    - Deployer.install (method)
    - HostConfig class
    - deployWARs(method)
24. This point has been respected because we only have the package name on top, as the first declaration, and all the import follow this package name.
25. About the order of the interface declaration:
    a. Class/interface documentation comment is in the correct order.

    b/c. These two points are in the inverse order in our code.

    d. The variables are not in the specified order as it is stated in the document, because at first(line 22/23) there are two private variables, instead of starting with the public ones.

e. The variables are not in the specified order as it is stated in the document, because at first(line 269) there are two private variables, instead of starting with the public ones.(app-base /config-base variables)

f/g. After the variables there are the get/set methods instead of the class constructor. Therefore this point is not respected. In our class the default constructor is used, since there is no other one declared.

26. At this point we think that ValidateContextPath should not be in the class since it's just a method that manipulates strings and therefore it is not used to offer a specific functionality of this class which is the host configuration. The other methods in this class are grouped for their functionality.

27. **Long methods + big classes + coupling and cohesion**
Our class is a big class, its length is about 1300 lines. It contains long methods, as the one we are considering (deployWars) and checkContextLastModified. We think that the cohesion and the coupling are adequate, because this class, as concerning our method, uses Deployer and ExpandWar as external classes. So, since they are only two, there's not a strong coupling between this class and the others and there's a good cohesion too , because this class mostly uses the method inside itself.
**Breaking Encapsulation**
The protected variables declared starting from line 251 to 276 and the one at line 327 could break incapsulation because declaring a variable as protected can be accessed directly from derived classes and the classes in the same package,so it depends on the behaviour of the classes that access this variables and how the code that uses this variables in this class behaves if these variables values are corrupted or not valid.
**Duplication**
The code of our assigned method doesn't contain duplicated code that does the same thing in different part of the method.

28. There are at the start of the class some public variables, but since they are static it is correct to have them public because no other classes can modify them. The distinction between private and protected visibility modifier can't be clear before knowing from which class host config is used or it's father of. This thing it's not well documented in the javaDoc. To better understand this point it would be useful a uml diagram that shows the relations and dependencies between the classes.
About the type of the variables everything it's correct, because this code can be executed and so they must be of the correct type.

29. In our class,class variables are declared in the proper scope because each one of them is used in many methods or it is used as a history by the same method. We didn't find in our analysis any class variable that is used only by one method (that happens when the scope is wrong and that variable should be declared only in the method that uses it)

30. In our method the constructors are called just before the new objects are effectively used.

31. The variables in the method are set to NULL value in order to make the code compilable, but they are all initialized to a different value before their usage to avoid NullPointerExceptions.

32. In our method not all the variables are initialized when they are declared. This rule exceptions are at line 697(variable jar), 698(variable entry), 699(variable istream), 700(variable ostream), 769(variable url), 770(variable path).

33. There are some exceptions to this rule that are at line: 672(variable dir), 679(variable contextPath), 680( variable period), 697(variable jar), 698(variable entry), 699(variable istream), 700(variable ostream), 701(variable xml), 713(variable buffer), 769(variable url), 770(variable path).

34. In our method all parameters are presented in the correct order in each submethod called by our method.

35. In order to perform the functionalities of this method the right submethods are called, so we didn't find any error at this point.

36. In our method all the method returned values inside the method are used properly. About the return value of deployWAR method, it can be used since it is a void return.

37. We didn't find any error because the "files" array passed as parameter is scanned by the for cycle with the index that goes from zero to files.length-1 and therefore it is all scanned. There's an other array, at line 713, called buffer, that's used from the first element of it, because of the caractheristics of the InputStream.read method. InvalidWars array is all scanned because our method uses the contains method to find an object in the arrayList and this also for the deployed array.

38. For the same reason as before, the files array, can't go out of bounds, because the indexing variable i has always a value in the range between 0 and files.length-1. The buffer array, instead, can't go out of bounds, because the read method reads maximum 1024 chars and this buffer is never scanned directly. InvalidWars and Deployed array can't go out of bounds because in our method only contains and add operations are performer on those arrays and it doesn't scan them by index.

39. In our method we only have two arrays:
    -Files that is passed as parameter and therefore it doesn't need to be initialized
    -buffer at line 713 that correctly calls the constructor for a new array as soon as it is declared.
    So this point is verified.

40. In our method there are not comparison with ==, but they are all made by equals.

41. In our method there are no system.out.println (direct outputs), but there is a log that will be read and therefore we must check the correctness of the messages written there. These messages in our method are:

INVALID_WAR_NAME_EXCEPTION: no error
EXPANDING_WEB_APP: no error
EXPANDING_WEB_APP_EXCEPTION: no error
EXPANDING_WEB_ARCHIVE_EXCEPTION: error → 'orrur' and not 'occur'
DEPLOYING_WEB_APP_ARCHIVE: no error
ERROR_DEPLOYING_WEB_APP_ARCHIVE_EXCEPTION: no error

42.All the previous error messages are comprehensive.

43. All the previous error messages are formatted correctly.

44. In our method we find only one case of brutish programming: at line 714 in our opinion was better to insert a do-while cycle that verifies in the while the condition (n<0) instead of while(true) and to have inside an if that checks this. In addition to this, our method(deployWAR) is very long and with different functions to complete in it. We suggest to split this method into smaller methods, that are for example: the extraction of the meta_inf context.xml could be done by another method and the deployment of the WAR file as a file in on method, and deployment of the WAR file as a folder in another method.

45/46. In our method there aren't mistakes in the usage of parenthesis, that can take to precedence problems, maybe in an if condition.

47. In our method there aren't denominators and therefore it cannot be this problem.

48. In our method there aren't arithmetic operations and therefore it's impossible the presence of this problem.

49. In our method all the comparisons and the conditions in the if branches are correct in the context of the methods.

50. In our method the try/catch are all useful to handle exception that may occur.

51. In our method there aren't any implicit type conversions, since the initialization of each variable is made directly by calling the constructor or by calling a method whose return type is the same of the variable type, or the value to assign to the variable is assigned directly, for example the NULL value.

52. In our method the relevant exceptions are caught because the try-catch block starting at line 705 catches the I/O exception which is the most important one.In the finally branch of this try/catch block all the closings of files and streams are handled correctly and so it is for the other try/catch blocks. It is to notice that not all the exceptions that can happen are handled. For example, the first lines of the method are not in any try/catch block and therefore if an exception occurs there, it is not handled by the method,which could be correct if our method is called by other methods (maybe in other classes) that handles these uncaught exceptions,but anyway this is not explained in the documentation.

53. At line 732, we think that there is an error because no action is done when an I/O exception occurs. For example, the log is not updated about this error, as it is done in others catch branches. All try/catch in the finally branch at line 734, does not handle any particular action in case of error, but this is correct because the only action performed in each try is just the closing of a file or a string.

54-55 We didn't find any switch in all the method and so there couldn't be any error.

56. In our method there are only two cycles: at line 664, there's a for cycle well declared and at line 714 there's a while cycle well declared. The initialization and the termination conditions are correct for each of the two cycles.

57.There are two files declarations in our method. The first one is at line 672 and is used first to check the file properties for example the path and if this file is a file or a directory. In a second time ,at line 707, it's also opened to get the specified jar entry. The second file declaration is at line 701 and it's used to create a new xml file (in which the entry of the previous file is copied into). This file is actually opened at line 710 when the output buffer is declared.

58.All the two files are closed properly. The first one is closed in the try block at line 724 and in the finally block at line 745. The second one is closed in the try block at line 722 and in the finally block at line 737.So both files are closed in the correct working of the program and in case of an error.

59. The reading of the first file is done in the while cycle at line 714. The EOF condition is handled correctly because if EOF is reached the n variable has -1 as value and the cycle stops.

60.All file exceptions are caught correctly (IO Exceptions) but as stated before at point 53 no action is performed when this exception occurs at line 732,and this is not clear reading the documentation and code comments. In the finally statement starting at line 734 all exceptions that could happen in closing operations are caught correctly since the Throwable exception includes also the IO Exception. This exceptions are not managed (there is no code inside the catch) because the code simply wants to close this files/streams without having an exception raised and therefore we think it is correct that no action is performed in their relative catch.

## Other problems

We think that the null allocation to a method local variable, that happens frequently in our code, can easily lead to nullPointerExceptions. In order to avoid this risk we suggest to set to null these variables at the end of the method, so there is only one section in which this assignments are made and since they are at the end of the method no other line of code can use them and can cause NullPointerExceptions.

## Changes in version 1.1

- Added a more detailed explanation of class role

**Hours of work**

Ivan: 20 hours

Riccardo: 20 hours