**Chapter 4**

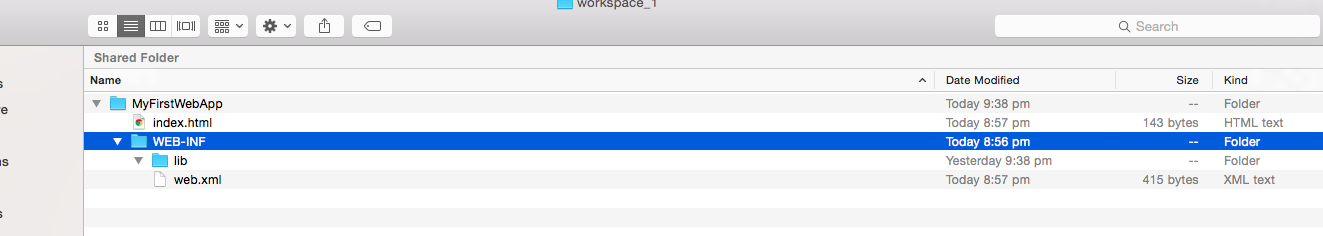
Enough with theories, In this chapter we will dirty our hands for some practical implementation.

For rest of the tutorials we will use Eclipse IDE to write sample applications, but first we will try to create Java dynamic web applications without any IDE. So let's get started... !!!

Note: We are using Tomcat 7 as our servlet container.

Let's create a simple dynamic web application with just one HTML file

First create the directory structure shown below



sample code for index.html is

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>My First Practise</title>

</head>

<body>

Yipeeeeeee It runssss !!!!

</body>

</html>

sample code for web.xml is:

<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns="http://java.sun.com/xml/ns/javaee"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"

id="WebApp\_ID" version="3.0">

<display-name>WebPrac</display-name>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

</welcome-file-list>

</web-app>

Don't worry about the details of web.xml(Deployment Descriptor) as of now we will come back to it later. As of now just check out one element that is <welcome-file-list>. This element is use to tell

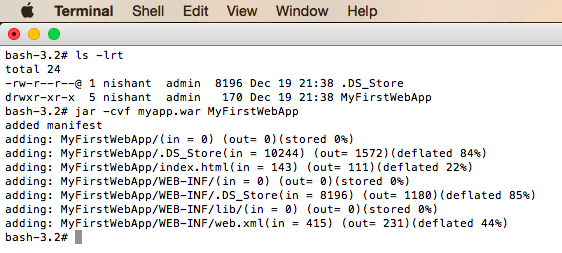
that upon loading this web application, by default control will go to index.html.

Now we need to package all this stuff or need to create a web archive(war) file.

We will use jar command to create our war file

command is : jar -cvf [name of war file with .war extension] [directory to archive]

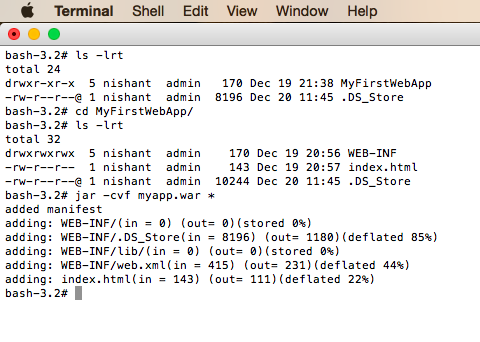
Go to the one level before your web application folder and issue the command.



Here -cvf tells “compress everything in this directory into a file named myapp.war”

[c = create, v = verbose, f = file]

OR you can execute this command from within web application folder.

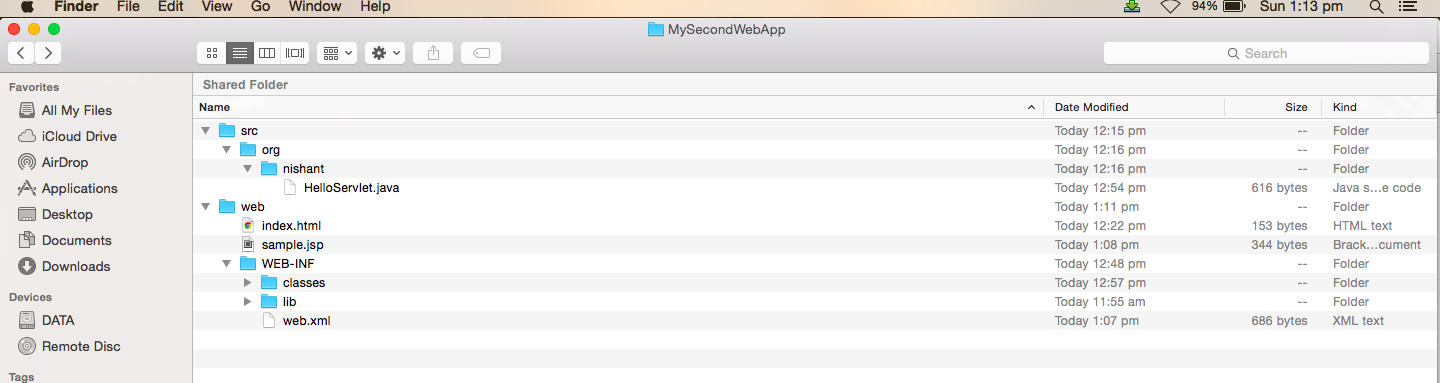


Now we can deploy this to Tomcat 7 using admin console.

**Example 2:**

In the above example we have included just one simple html file, but in the real world scenario we may have some java code, JSP files and many more things. So let's look at an example which contains some servlets and JSP file.

Create the directory structure shown below:



Here we have more generalized directory structure as compare to our previous example.

Inside our web application directory we have two directories “src” and “web”, “src” directory contains our java files and web directory contains our html/JSP files and important one “WEB-INF” directory.

Inside “WEB-INF” directory we have “classes” directory which will contain compiled java classes then we have “lib” directory in case we want some external jar files and in the last we have “web.xml” Deployment descriptor.

Sample HelloServlet.java

package org.nishant;

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class HelloServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException{

PrintWriter out = response.getWriter();

out.println("<html>");

out.println("<head><title>Hello World</title></title>");

out.println("<body>");

out.println("<h1>Hello World from Servlet</h1>");

out.println("</body>");

out.println("</html>");

}

}

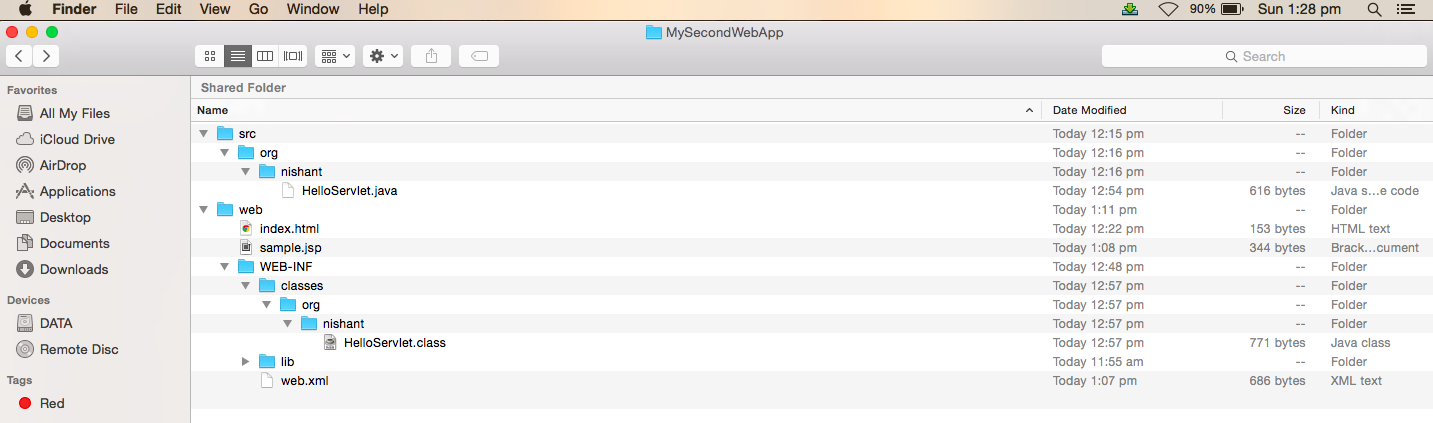
Now we need to compile this java file and place the .class file inside classes directory, below mentioned command will complete this task.

javac -d /Volumes/DATA/MyFolder/Private/study/Java/EnterpriseJava/Servlet\_JSP/workspace\_1/MySecondWebApp/web/WEB-INF/classes -sourcepath src -cp /Applications/apache-tomcat-7.0.63/lib/servlet-api.jar src/org/nishant/HelloServlet.java

-d will decide the output directory

-cp will set the classpath

As our HelloServlet.java contains servlet specific apis so we need servlet-api.jar file in our classpath in order to compile, so we have taken servlet-api.jar from the lib directory of Tomcat 7



Sample code for index.html

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>My Second Practise</title>

</head>

<body>

Hello World from welcome file......

</body>

</html>

Sample code for sample.jsp

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

pageEncoding="ISO-8859-1"%>

<!DOCTYPE html>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title>My Second Practise</title>

</head>

<body>

<%

for (int i = 0; i < 10; i++) {

out.println("Hello World!!!");

}

%>

</body>

</html>

Don't worry about the syntax of JSP, we will come back to it later, as of now just note down in order to write java code inside JSP file we need scriplet <% %>.

And in the Last Sample code for web.xml

<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns="http://java.sun.com/xml/ns/javaee"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"

id="WebApp\_ID" version="3.0">

<display-name>WebPrac</display-name>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

</welcome-file-list>

<servlet>

<servlet-name>HelloServlet</servlet-name>

<servlet-class>org.nishant.HelloServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>HelloServlet</servlet-name>

<url-pattern>/HelloServlet</url-pattern>

</servlet-mapping>

</web-app>

Here we have added some extra code, apart from welcome file that is servlet mapping, which tells container to map HelloServlet with url /HelloServlet.

Now execute the following command to create war file. Make sure that we need to archive all the data inside our “web” directory.

