A **servlet** is a **small Java program** that runs within a **Web server**.

It is used **to develop web based** framework.

Servlets receive and respond to requests from Web clients,

usually across HTTP, (the HyperText Transfer Protocol).

Servlets are most popularly used **for generating dynamic** content on the Web.

A Servlet is **a class**

that **handles** requests,

**processes** them

and **reply back** with a response.

For example, we can use a Servlet to collect input from a user through an HTML form,

query records from a database,

and send response back on web pages

It create web pages dynamically.

**life Cycle** :- init(), service(), destroy()

The init() method is called only once during the life cycle of servlet.

* We can override these methods and provide our own implementation.

// For Eclipse setup

we will need Eclipse & Tomcat for this

we will need to setup downloaded Tomcat server over Eclipse

We have to create "Dynamic web project"

And make sure to select "Generate web.xml deployment descriptor"

1) Java installation

2) Tomcat installation

3) folder structure of servlet in Tomcat

Project Name (Context Root)

WEB-INF

classes

.class files

lib

web.xml (deployment descriptor)

HTML/JSP pages

Static content (CSS/Images)

4) folder structure of servlet in Eclipse

Project Name (Context Root)

SRC (All Java Code)

WEB-INF

lib

web.xml (deployment descriptor)

HTML/JSP pages

Static content (CSS/Images)

5) jar files for servlet (servlet-api.jar)

You need **Servlet-api**.**jar** to compile **servlets** in eclipse but while deploying **servlet** container ( like tomcat ) will have it built in.

Infact it is bad practise to include it inside your WEB-INF/LIB folder.

6) For servlet **"javax.servlet.http.HttpServlet"** is the super class.

we have to **extend** it, in our servlet class.

7) explaining **web.xml** deployment descriptor

- **welcome file list** (if not present, the priority will be - index.html, index.htm, index.jsp)

- servlet **mapping**

- load on startup [pre initialization of servlet] (0,1,2.. If negative value - servlet will be loaded at request time)

8) To map a servlet through deployment descriptor web.xml

<?xml version=......> -- we get it in Eclipse

<web-app>

<display-name>ProjectName</display-name>

<welcome-file-list>

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

</welcome-file-list>

<servlet> -- creating new servlet node

<servlet-name>**xmlServlet**</servlet-name> -- some name

<servlet-class>com.vinay.MyXmlServlet</servlet-class> -- our java class name

<load-on-startup>0</load-on-startup> -- Optional

</servlet>

<servlet-mapping> -- creating another node for mapping

<servlet-name>**xmlServlet**</servlet-name> -- same name as servlet node has

<url-pattern>/SimpleServletPath</url-pattern> -- the url link name

</servlet-mapping>

</web-app>

11) **To print at browser** :

PrintWriter writer = response.getWriter();

writer.println("<h3>Hi first code</h3>");

OR

response.getWriter().print("<h3>Hi first code</h3>");

12) Run a sample Servlet Example.

To compile the java program with CMD you will need to place the “servlet-api.jar” at Java/jdk/jre/lib/ext directory

----------------------------------------------------------------------------------------------------

13) Servlet-3

import **java.io.IOException;**

import **javax.servlet.annotation.WebServlet;**

import **javax.servlet.http.HttpServlet;**

import **javax.servlet.http.HttpServletRequest;**

import **javax.servlet.http.HttpServletResponse;**

@**WebServlet**(description = "A project", urlPatterns = {"/SimpleServletPath"})

OR @**WebServlet**(“/SimpleServletPath”)

OR @**WebServlet**(urlPatterns = {"/oneUrl", “/twoUrl”})

OR @**WebServlet**(

name=”MyOwnServlet”,

description = "A project",

urlPatterns = "/SimpleServletPath”

)

public class SimpleServlet **extends** HttpServlet{

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

// Our code

response.getWriter().println("Hi first code");

}

}

Link : <http://localhost:8080/MyServletProject/SimpleServletPath>

Another Example for retrieving InitParams

import **java.io.IOException;**

import **java.io.PrintWriter;**

import **javax.servlet.annotation.WebInitParam;**

import **javax.servlet.annotation.WebServlet;**

import **javax.servlet.http.HttpServlet;**

import **javax.servlet.http.HttpServletRequest;**

import **javax.servlet.http.HttpServletResponse;**

@**WebServlet**(

urlPatterns = "/imageUpload”,

initParams = {

@WebInitParam(name=”saveDir”, value=”D:/FileUpload”),

@WebInitParam(name=”allowedTypes”, value=”jpg,jpeg,gif,png”)

}

)

public class SimpleServlet **extends** HttpServlet{

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

String saveDir = getInitParameter(“saveDir”);

String fileTypes = getInitParameter(“allowedTypes”);

PrintWriter writer = response.getWriter();

writer.println("saveDir = "+ saveDir);

writer.println("fileTypes = "+ fileTypes);

}

}

// Here doGet method retrieves values of these parameters and prints them out to the client

// asyncsupported

@**WebServlet**(

urlPatterns = "/myController”,

loadOnStartup = 1,

asyncSupported = true

)

// Related annotations

@WebFilter, @WebListener, @WebInitParam, @HandlesTypes, @MultipartConfig, @ServletSecurity, @HttpMethodContraint, @HttpConstraint

7) **urlPatterns** : it is the name by which we will call that java class and execute

8) in above sample code we have used **@WebServlet** annotation to specify **urlPatterns**

instead of that we can specify the **urlpatterns in web.xml** as well.

Attributes of **@WebServlet**

**value** or **urlPatterns** - Required (Only this is required others are optional)

**name** – name of the servlet

**displayName** – Display name of the servlet

**description** - Description of the servlet, IDE/Tools use this

**asyncSupported** – default is False, IDE/Tools use this

**initParams** – one or more initialization parameters of the servlet, Each Parameter is specified by @WebInitParam annotation type

**loadOnStartup** – order of servlet

**smallIcon** – small icon image, IDE/Tools use this

**largeIcon** – large icon image, IDE/Tools use this

//the annotations are introduced in java after 1.5 version.

// earlier there were only xml configurations present also,

// if we want to change anything in annotation we have to recompile and rebuild that class,

// but in web.xml we do not need recompilation or rebuild.

// However we need to restart Tomcat whenever we make changes in web.xml.

while writing annotations we have to write very less code in compare to web.xml

its upto us what we want to use, annotation or deployment descriptor web.xml

13) The HttpServlet have few methods which we have to use, like -

**DoGet** & url parameter passing:-

Link : http://localhost:8080/MyServletProject/SimpleServletPath?**uid**=Vinay

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 SimpleServlet **extends** HttpServlet{

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

response.setContentType("text/html");

PrintWriter out = response.getWriter();

String userName = request.getParameter("**uid**"); // getting the parameter value, uid is passed with url

out.println("<h3>Hi "+userName+"</h3>"); // setting the value

}

}

14) **doPost** : Here we can accept the data from a html <form method="post" action="hello">

- provide proper names to form-fields as, name="uid", name="location"

- default method is 'get', so make it 'post

<form method=”post” action=”hello”>

UID :- <input type=”text” name=”uid”> <br>

<input type=”checkbox” name=”location” value=”Loc1”> Location1 <br>

<input type=”checkbox” name=” location” value=”Loc2”> Location2 <br>

<input type=”checkbox” name=” location” value=”Loc3”> Location3 <br>

<input type=”submit” name=”submit”>

</form>

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 SimpleServlet **extends** HttpServlet{

protected void **doPost**(HttpServletRequest request, HttpServletResponse response) throws IOException{

response.setContentType("text/html");

PrintWriter out = response.getWriter();

String userName = request.getParameter("**uid**"); // getting the parameter value

out.println("<h3>Hi "+userName+"</h3>"); // setting the value

String location[] = request.getParameterValues("**location**"); // to take more than 1 values from multi select, <select name="location" multiple size="3">.... OR from checkboxes

// getParameterValues, returns String array

out.println(“<h3>Selected Locations : “+location.length+”</h3>”);

if(location.length!=0){

for(String l: location){

out.println(“<h3>Location “+l+”</h3>”);

}

}else{

out.println(“<h3>No Location Selected</h3>”);

}

}

}

//Use doGet when you want to get something from application (retrieve the data)

//and use doPost when you want to submit something to the server. (create a user/update user)

when we use "get" method and refresh the page after submit, it will not ask for confirmation

but, if we use "post" method and refresh the page after submit, it will ask for confirmation

15) Tomcat creates the Servlet / HttpServletRequest / HttpServletResponse objects, as well as destroys.

-Servlet runs in the Tomcat container

-Every class we use here that we haven't written is provided by Tomcat

-Request & Response objects are created per access.

-But, Servlet object is not created per access, it get re-used

-Different requests have different servlet threads, not instances.

15) HTTP is a stateless protocol, means this protocol does not expect that the data will be remembered.

For ex - to make some request if we pass some parameter, HTTP will not remeber it for next time.

To make server (Tomcat) remember it, there is a Session object.

Session object is also provided by Tomcat only.

16) to use session object:-

HTTPSession session = request.getSession();

// now we can store any value in this session object, and get remembered

session.setAttribute("savedUserName", userName);

out.println("<h3>Hi "+(String) session.getAttribute("savedUserName")+"</h3>"); // displaying the session value

The session object is created per user / per machine

- session can be used for login & shopping carts

Every request object has a handle to the session object

17) Context object :-

it is also provided by the Tomcat.

it presents across the entire application

can be shared across servlets and users and browsers as well

it will be needed in case of - Initialization code / Common bulletin board

This object can be used to get configuration information from web.xml file.

If any information is shared to many servlet, it is better to provide it from the web.xml file using the **<context-param>** element.

SevletContext context = request.getSevletContext();

context.setAttribute("savedUserName", userName);

out.println("<h3>Hi "+(String) context.getAttribute("savedUserName")+"</h3>"); // displaying the session value

In servlet every request is considered as a different thread not as object

it instantiates only once and serve with threads.

Example1 :-

1. <web-app>
3. <servlet>
4. <servlet-name>sonoojaiswal</servlet-name>
5. <servlet-**class**>DemoServlet</servlet-**class**>
6. </servlet>
8. <context-param>
9. <param-name>dname</param-name>
10. <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
11. </context-param>
13. <servlet-mapping>
14. <servlet-name>sonoojaiswal</servlet-name>
15. <url-pattern>/context</url-pattern>
16. </servlet-mapping>
18. </web-app>
19. **import** java.io.\*;
20. **import** javax.servlet.\*;
21. **import** javax.servlet.http.\*;

24. **public** **class** DemoServlet **extends** HttpServlet{
25. **public** **void** doGet(HttpServletRequest req,HttpServletResponse res)
26. **throws** ServletException,IOException
27. {
28. res.setContentType("text/html");
29. PrintWriter pw=res.getWriter();
31. //creating ServletContext object
32. ServletContext context=getServletContext();
34. //Getting the value of the initialization parameter and printing it
35. String driverName=context.getInitParameter("dname");
36. pw.println("driver name is="+driverName);
38. pw.close();
40. }}

### Example2 :-  get all the initialization parameters

1. <web-app>
3. <servlet>
4. <servlet-name>sonoojaiswal</servlet-name>
5. <servlet-**class**>DemoServlet</servlet-**class**>
6. </servlet>
8. <context-param>
9. <param-name>dname</param-name>
10. <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
11. </context-param>
13. <context-param>
14. <param-name>username</param-name>
15. <param-value>system</param-value>
16. </context-param>
18. <context-param>
19. <param-name>password</param-name>
20. <param-value>oracle</param-value>
21. </context-param>
23. <servlet-mapping>
24. <servlet-name>sonoojaiswal</servlet-name>
25. <url-pattern>/context</url-pattern>
26. </servlet-mapping>
28. </web-app>
29. **import** java.io.\*;
30. **import** javax.servlet.\*;
31. **import** javax.servlet.http.\*;

34. **public** **class** DemoServlet **extends** HttpServlet{
35. **public** **void** doGet(HttpServletRequest req,HttpServletResponse res)
36. **throws** ServletException,IOException
37. {
38. res.setContentType("text/html");
39. PrintWriter out=res.getWriter();
41. ServletContext context=getServletContext();
42. Enumeration<String> e=context.getInitParameterNames();
44. String str="";
45. **while**(e.hasMoreElements()){
46. str=e.nextElement();
47. out.print("<br> "+context.getInitParameter(str));
48. }
49. }}

18) Cookies :-

Cookies are created using Cookie class present in Servlet API. Cookies are added to response object using the addCookie() method.

This method sends cookie information over the HTTP response stream. getCookies() method is used to access the cookies that are added to response object.

Cookie ck = new Cookie(“userName”,name); // creating new cookie object

ck.setMaxAge(30\*60); // setting maximum age of cookie // To delete the cookie set 0 here

response.addCookie(ck); // adding cookie to response object

Cookie[] cks = request.getCookies(); // getting the cookies

for(int i=0;i<cks.length;i++)

{

out.println("Cookie Name is : "+cks[i].getName()+" and value is : "+cks[i].getValue()+"<br>");

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

}

17) Servlet Inheritance :

GenericServlet --> HTTPServlet --> MyServlet

3 level heirarchy

GenericServlet implements Servlet, ServletConfig

Servlet, ServletConfig are interfaces

GenericServlet provides - init(ServletConfig) / init() / service(ServletRequest req, ServletResponse res)

HTTPServlet provides - service(ServletRequest req, ServletResponse res) :~ here the casting takes place, from ServletRequest/ServletResponse to HttpServletRequest / HttpServletResponse

service(HttpServletRequest req, HttpServletResponse resp) :~ so here we have 2 service methods.

doGet(HttpServletRequest req, HttpServletResponse resp)

doPost(HttpServletRequest req, HttpServletResponse resp)

in MyServlet we use - doGet() / doPost()

18) RequestDispatcher

Facility of dispatching the request to another resource (the resource can be html, servlet or jsp)

It is the way of Servlet Collaboration.

Methods –

forward() -- forwards the request

include() -- includes the content of the resource in the response

With forward we send the response of one servlet to another servlet, and send back the response of last servlet.

Client -> Request -> Servlet1 -> Servlet2

Servlet2 -> Response -> Client

* Here we are sending response of servlet1 to servlet2 with forward method.

With include we include the response of another servlet in main servlet and return the response.

Client -> Request -> Servlet1 <- Servlet2

Servlet1 -> Response -> Client

* Here we are including response of servlet2 to servlet1 with include method.

Example :-

Index.html (Login page)

* Validate Servlet
  + If Valid
    - Forward to Welcome Page
  + If Not Valid
    - Include error page

Index.html ::::::

1. <form action="servlet1" method="post">
2. Name:<input type="text" name="userName"/><br/>
3. Password:<input type="password" name="userPass"/><br/>
4. <input type="submit" value="login"/>
5. </form>

Login.java ::::::

1. **import** java.io.\*;
2. **import** javax.servlet.\*;
3. **import** javax.servlet.http.\*;

6. **public** **class** Login **extends** HttpServlet {
8. **public** **void** doPost(HttpServletRequest request, HttpServletResponse response)
9. **throws** ServletException, IOException {
11. response.setContentType("text/html");
12. PrintWriter out = response.getWriter();
14. String n=request.getParameter("userName");
15. String p=request.getParameter("userPass");
17. **if**(p.equals("servlet"){
18. RequestDispatcher rd=request.getRequestDispatcher("servlet2");
19. rd.**forward**(request, response);
20. }
21. **else**{
22. out.print("Sorry UserName or Password Error!");
23. RequestDispatcher rd=request.getRequestDispatcher("/index.html");
24. rd.**include**(request, response);
26. }
27. }
29. }

**WelcomeServlet.java** ::::::

1. **import** java.io.\*;
2. **import** javax.servlet.\*;
3. **import** javax.servlet.http.\*;
5. **public** **class** WelcomeServlet **extends** HttpServlet {
7. **public** **void** doPost(HttpServletRequest request, HttpServletResponse response)
8. **throws** ServletException, IOException {
10. response.setContentType("text/html");
11. PrintWriter out = response.getWriter();
13. String n=request.getParameter("userName");
14. out.print("Welcome "+n);
15. }
17. }

**web.xml** ::::::

1. <web-app>
2. <servlet>
3. <servlet-name>Login</servlet-name>
4. <servlet-**class**>Login</servlet-**class**>
5. </servlet>
6. <servlet>
7. <servlet-name>WelcomeServlet</servlet-name>
8. <servlet-**class**>WelcomeServlet</servlet-**class**>
9. </servlet>

12. <servlet-mapping>
13. <servlet-name>Login</servlet-name>
14. <url-pattern>/servlet1</url-pattern>
15. </servlet-mapping>
16. <servlet-mapping>
17. <servlet-name>WelcomeServlet</servlet-name>
18. <url-pattern>/servlet2</url-pattern>
19. </servlet-mapping>
21. <welcome-file-list>
22. <welcome-file>index.html</welcome-file>
23. </welcome-file-list>
24. </web-app>

18) SendRedirect in servlet

The **sendRedirect()** method of **HttpServletResponse** interface can be used to redirect response to another resource (html, jsp etc).

It accepts relative as well as absolute URL.

|  |  |
| --- | --- |
| **forward() method** | **sendRedirect() method** |
| The forward() method works at server side. | The sendRedirect() method  works at client side. |
| It sends the same request and response  objects to another servlet. | It always sends a new request. |
| It can work within the server only. | It can be used within and  outside the server. |
| Example: request.getRequestDispacher("servlet2")  .forward(request,response); | Example:  response.sendRedirect("servlet2"); |

response.sendRedirect("http://www.anotherUrl.com");   // In our servlet java file

18) \* ServletConfig :- some configuration values which are pre defined in web.xml

This object can be used to get configuration information from web.xml file.

to set default value :-

@WebServlet(description = "A project", urlPatterns = {"/SimpleServletPath"}, **initParams**={@WebInitParam(name="defaultUser", value="Vinay Padole")})

public class SimpleServlet extends HttpServlet{

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException{

PrintWriter out = response.getWriter();

out.println("Init parameter has default username as : "+ getServletConfig().getInitParameter("defaultUser"));

// Another way in detail -

ServletConfig config=getServletConfig();

     String defaultUser =config.getInitParameter("defaultUser ");

     out.print("Default User is: "+ defaultUser);

out.close();

}

}

Same we can do with our web.xml, instead of annotations @WebServlet -

<servlet>

<servlet-name>SimpleServlet</servlet-name>

<servlet-class>SimpleServlet</servlet-class>

<init-param>

<param-name>defaultUser</param-name>

<param-value>Vinay Padole</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>SimpleServlet</servlet-name>

<url-pattern>/SimpleServletPath</url-pattern>

</servlet-mapping>

// This will also act same in above code, just remove annotation @WebServlet

//Another example with JSP web.xml mapping geting initparam from web.xml:-

<servlet>

<servlet-name>InitJSP</servlet-name>

<jsp-file>/initpage.jsp</jsp-file> // instead of servlet-class

<init-param>

<param-name>defaultUser</param-name>

<param-value>Vinay Padole</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>InitJSP</servlet-name>

<url-pattern>/initpage.jsp</url-pattern>

</servlet-mapping>

in initpage.jsp :-

The default user form the servlet config is : <%=getServletConfig().getInitParameter("defaultUser"); %>

### // Another Example - get all the initialization parameters

1. <web-app>
3. <servlet>
4. <servlet-name>DemoServlet</servlet-name>
5. <servlet-**class**>DemoServlet</servlet-**class**>
7. <init-param>
8. <param-name>username</param-name>
9. <param-value>system</param-value>
10. </init-param>
12. <init-param>
13. <param-name>password</param-name>
14. <param-value>oracle</param-value>
15. </init-param>
17. </servlet>
19. <servlet-mapping>
20. <servlet-name>DemoServlet</servlet-name>
21. <url-pattern>/servlet1</url-pattern>
22. </servlet-mapping>
24. </web-app>
25. **import** java.io.IOException;
26. **import** java.io.PrintWriter;
27. **import** java.util.Enumeration;
29. **import** javax.servlet.ServletConfig;
30. **import** javax.servlet.ServletException;
31. **import** javax.servlet.http.HttpServlet;
32. **import** javax.servlet.http.HttpServletRequest;
33. **import** javax.servlet.http.HttpServletResponse;

36. **public** **class** DemoServlet **extends** HttpServlet {
37. **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)
38. **throws** ServletException, IOException {
40. response.setContentType("text/html");
41. PrintWriter out = response.getWriter();
43. ServletConfig config=getServletConfig();
44. Enumeration<String> e=config.getInitParameterNames();
46. String str="";
47. **while**(e.hasMoreElements()){
48. str=e.nextElement();
49. out.print("<br>Name: "+str);
50. out.print(" value: "+config.getInitParameter(str));
51. }
53. out.close();
54. }
56. }

19) Sample Login application:

--------------------------------

in view(login.jsp page):-

<form method="post" action="login">

<br>UserId : <input name="userId" type="text">

<br>Password : <input name="password" type="password">

<br><input type="submit" value="Submit">

</form>

in success.jsp:-

<%@ page import="org.vinay.dto.User" %>

<!DOCTYPE><html><head></head><body>

<h3>Login Successful!</h3>

<%

User user1 = (User) session.getArrtibute("user");

%>

<%

User user2 = (User) request.getArrtibute("user");

%>

From Session :- Hello <%=user1.getUserName() %> // here url will change to /success

From RequestDispatcher :- Hello <%=user2.getUserName() %> // here the url remain same

</body></html>

in Controller(LoginServlet):-

With Session:-

@WebServlet("/login")

public class LoginServlet extends HttpServlet{

private static final long serialVersionUID = 1L;

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws Exception{

String userId, password;

userId = request.getParameter("uesrId");

password = request.getParameter("password");

LoginService loginService = new LoginService();

boolean result = loginService.authenticate(userId,password); // here result is model

if(result){

User user = loginService.getUserDetails(userId); // getting user info from Business service

request.getSession.setAttribute("user",user); // setting object into the session

// redirect to success.jsp

response.sendRedirect("success.jsp"); // if we want to redirect to another servlet pass its name here like :- /success

return;

}else{

response.sendRedirect("login.jsp");

return;

}

}

}

With RequestDispatcher:-

@WebServlet("/login")

public class LoginServlet extends HttpServlet{

private static final long serialVersionUID = 1L;

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws Exception{

String userId, password;

userId = request.getParameter("uesrId");

password = request.getParameter("password");

LoginService loginService = new LoginService();

boolean result = loginService.authenticate(userId,password); // here result is model

if(result){

User user = loginService.getUserDetails(userId); // getting user info from Business service

request.setAttribute("user",user); // setting object into the request, instead of session as above

// server side redirection with "request dispatcher", here Browser will not know about redirection

// RequestDispatcher helps us to redirect internally to our success.jsp , here the url will remain "/login"

RequestDispatcher dispatcher = request.getRequestDispatcher("success.jsp"); // it takes the resource parameter where we are despatching to

dispatcher.forword(request,response); // here we are transfering the request & response from this method to success.jsp

return;

}else{

response.sendRedirect("login.jsp");

return;

}

}

}

in Business Service(LoginService):-

public class LoginService{

HashMap<String, String> users = new HashMap<String, String>();

public LoginService(){

users.put("johndoe","John Doe");

users.put("janedoe","Jane Doe");

users.put("jguru","Java Guru");

}

public boolean authenticate(String userId, String password){

if(password==null || password.trim()==""){

return false;

}

retunr true;

}

public User getUserDetails(String userId){

User user = new User();

user.setUserName(user.get(userId));

user.setUserId(userId);

return user;

}

}

DTO (User class):- (Data Transfer Object) also we can call it as Model

public class User{

private String userName;

private String userId;

public String getUserName(){

return userName;

}

public void setUserName(String userName){

this.userName = userName;

}

public String getUserId(){

return userId;

}

public void setUserId(String userId){

this.userId = userId;

}

}

6) To create war :-

jar -cvf projectname.war \*

Here, -c is used to create file,

-v to generate the verbose output and

-f to specify the archive file name.

The \* (asterisk) symbol signifies that all the files of this directory (including sub directory).

To extract war in another folder :-

jar -xvf projectname.war