
Java Servlets with Tomcat

Tomcat

Apache Tomcat is an open source web server which is used as official reference implementation of Java Servlets and Java Server Pages technologies. To install Tomcat

- Download Tomcat Server (e.g. jakarta-tomcat-5.0.28.zip) on your D:\ drive from [\\indus\Common\cs391a05\Servlets & JSP](http://indus.Common/cs391a05/Servlets & JSP)
-
- Unzip Tomcat in D:\

Setting up Environment variables to run Tomcat

There are various files (classes, exe etc) needed by the system to run. Environment variables are used to tell your system (in this case your web server Tomcat) where the required files are.

To run Tomcat (web server) you need to set two environment variables.

- CATALINA_HOME
 - o Right click on My Computer icon. Select the advanced tab and click the environment variables button.
 - o Create a new User Variable
 - o Type CATALINA_HOME as the name of the environment variable.
 - o Its value should be the path till your top-level Tomcat directory. If you have unzipped the Tomcat in D drive. It should be *D:\jakarta-tomcat-5.0.28*
- JAVA_HOME
 - o Next step is to tell Tomcat where JDK is (i.e. to set JAVA_HOME)
 - o Create a new User Variable
 - o Set name of variable JAVA_HOME
 - o The value is the installation directory of JDK (in LUMS lab the value is something like *C:\Program Files\jdk1.4.2*)

Now Tomcat web server setup is complete.

- Open the *D:\jakarta-tomcat-5.0.28\bin* folder and locate the startup.bat file.
- Double clicking on this file will open up a DOS window, which will disappear, and another DOS window will appear, the second window will stay there. If it does not your paths are not correct.
- Now to check whether your server is working or not open up a browser and type <http://localhost:8080>. This should open the default page of Tomcat

Setting up Environment variables to compile servlets

To compile servlets you need to import javax.servlet package, which is not included in the standard JDK. However it comes along Tomcat.

- CLASSPATH
 - o This variable tells compiler where to find Servlets Class Library (i.e. servlet.jar).
 - o Create a new User Variable.
 - o Type CLASSPATH as the name of the environment variable
 - o The value should be the path where servlet.jar file resides. In our case it is *D:\jakarta-tomcat-5.0.28\common\lib\servlet.jar*

Tomcat Standard Directory Structure

A web application is defined as hierarchy of directories and files in a standard layout. Such hierarchies can be accessed in two forms

- Unpacked
 - o Where each directory & file exists in the file system separately
 - o Used mostly during development
- Pack
 - o Known as Web Archive (WAR) file
 - o Mostly used to deploy web applications

The *webapps* folder in top-level Tomcat directory contains all the web applications deployed on the server. Each application is deployed in a separate folder often referred as *Context*.

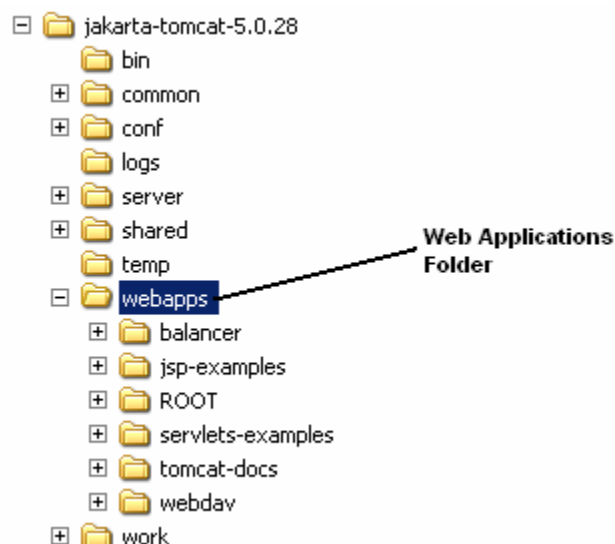


Figure 1: Tomcat Directory Structure

To make a new application in tomcat you need a specific folder hierarchy.

- Create a folder named *myapp* in *D:\jakarta-tomcat-5.0.28\webapps* folder.
- This name will also appear in the URL for your application. For example *http://localhost:8080/myapp/index.html*
- All JSP and html files will be kept in main application folder (*D:\jakarta-tomcat-5.0.28\webapps\myapp*)
- Configuration files such as *web.xml* will go in *WEB-INF* folder (*D:\jakarta-tomcat-5.0.28\webapps\myapp\WEB-INF*)
- Servlets and Java Beans will go in *classes* folder (*D:\jakarta-tomcat-5.0.28\webapps\myapp\WEB-INF\classes*)
- To test application hierarchy, make simple *index.html* file. Place this file in main application directory. Access using URL *http://localhost:8080/myapp/index.html*

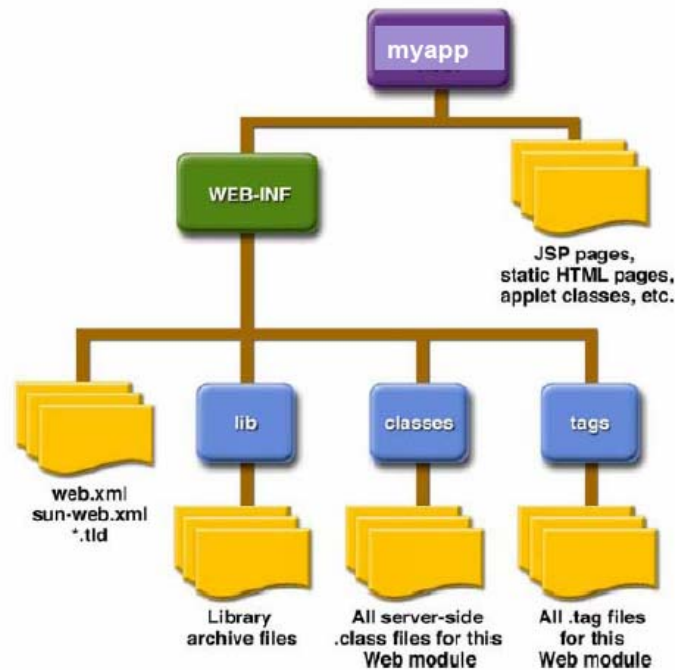


Figure 2: Web application folder hierarchy

Restart Tomcat every time you create a new directory structure, a servlet or a java bean so that it can recognize it. For JSP and html files you don't have to restart the server.

Writing Servlets

Servlet Types

- Servlets are based on two main packages *javax.servlet* and *javax.servlet.http*.
- Every servlet must implement the *javax.servlet.Servlet* interface, it contains the servlet's life cycle methods etc.
- *GenericServlet* class
 - o Implements *javax.servlet.Servlet*
 - o Extend this class for writing protocol independent servlets
- *HttpServlet* class
 - o Extends from *GenericServlet* class
 - o Adds functionality for writing HTTP specific servlets
 - o Extend from this class for writing HTTP based servlets

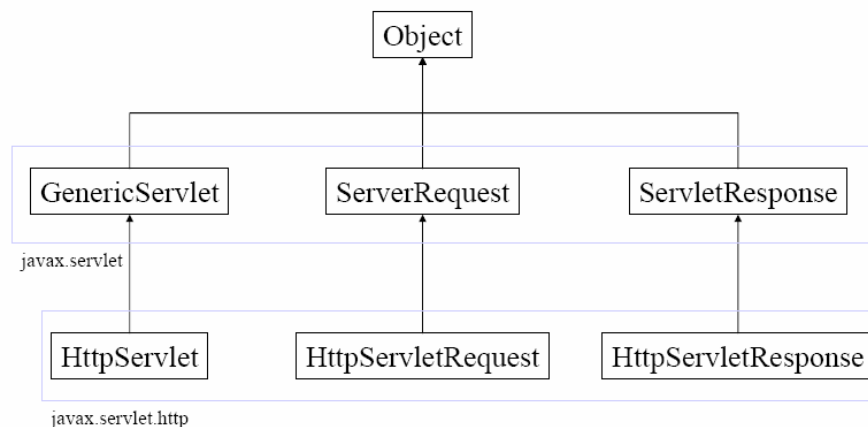


Figure 3: Servlet Class Hierarchy

Types of HTTP requests

- **GET:** Requests a page from the server. This is the normal request used when browsing web pages.
- **POST:** This request is used to pass information to the server. Its most common use is with HTML forms.
- **PUT:** Used to put a new web page on a server.
- **DELETE:** Used to delete a web page from the server.
- **OPTIONS:** Intended for use with the web server, listing the supported options.
- **TRACE:** Used to trace servers

How HTTP sends requests

- GET
 - o Attribute-Value pair is attached with requested URL after '?'.

- For example if attribute is 'name' and value is 'ali' then the request will be <http://www.gmail.com/register?name=ali>
- For HTTP servlet override *doGet()* methods of *HttpServlet* class to handle these type of requests.
- POST
 - Attribute-Value pair attached within the request body
 - Override *doPost()* method of *HttpServlet* class to handle POST requests.

Hello World Servlet

1. Create a directory structure for your application (i.e. helloapp). This is a one time process for any application
2. Create a *HelloWorldServlet* source file by extending this class from *HttpServlet* and overriding your desired method
3. Compile it (If get error of not having required packages, check your class path)
4. Place the class file in the classes folder of your web application (i.e. helloapp). If you have packages then create a complete structure under classes folder
5. Create a deployment descriptor (web.xml) and put it inside WEB-INF folder
6. Restart your server if already running
7. Access it using Web browser

HelloWorldServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorldServlet extends HttpServlet
{
    public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException
    {
        PrintWriter out = response.getWriter();
        out.println("<h1>Hello World!</h1>");
    }
}
```

Web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app>
    <servlet>
        <servlet-name>HelloWorldServlet</servlet-name>
        <servlet-class>HelloWorldServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>HelloWorldServlet</servlet-name>
        <url-pattern>/HelloWorld</url-pattern>
    </servlet-mapping>
</web-app>
```

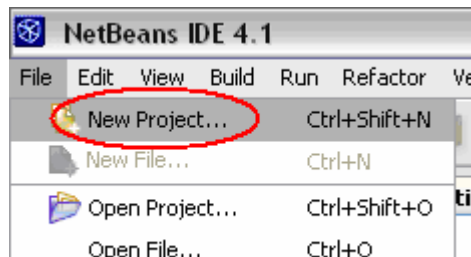
Compiling and Invoking Servlets

- Compile HelloWorldServlet.java using javac command.
- Put HelloWorldServlet class in *D:\jakarta-tomcat-5.0.28\webapps\helloapp\WEB-INF\classes*
- Put web.xml file in *D:\jakarta-tomcat-5.0.28\webapps\helloapp\WEB-INF*
- Invoke your servlet by writing following command in web browser
<http://localhost:8080/helloapp/HelloWorld>

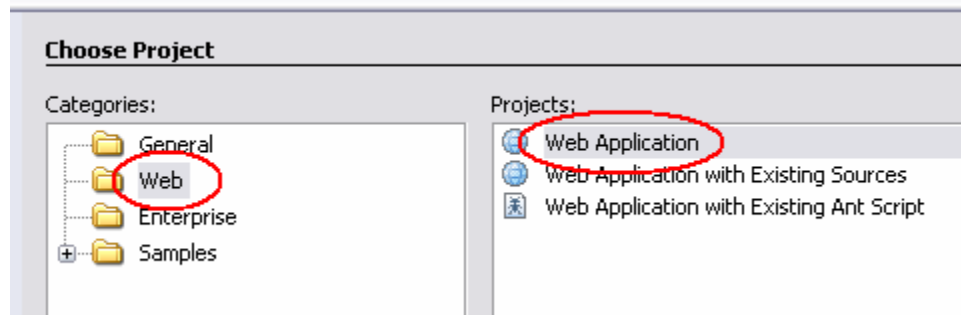
Writing Servlets using NetBeans 4.1

1. Create new Web Application

Create new project by selecting *New Project* in *File* menu



In *New Project* wizard window select *Web* in Categories list and *Web Application* in *Projects* list and press *Next* button



In *New Web Application* window type *MyFirstApp* in *Project Name* field and set *Server* to *Bundled Tomcat* and press *Finish* button.

Name and Location

Project Name: MyFirstApp

Project Location: G:\Documents and Settings\umair

Project Folder: G:\Documents and Settings\umair\MyFirstApp

Source Structure: Java BluePrints

Add to Enterprise Application: <None>

Server: Bundled Tomcat (5.5.7)

J2EE Version: J2EE 1.4

Context Path: /MyFirstApp

Recommendation: Source Level 1.4 should be used in J2EE 1.4 projects.

Set Source Level to 1.4

Set as Main Project

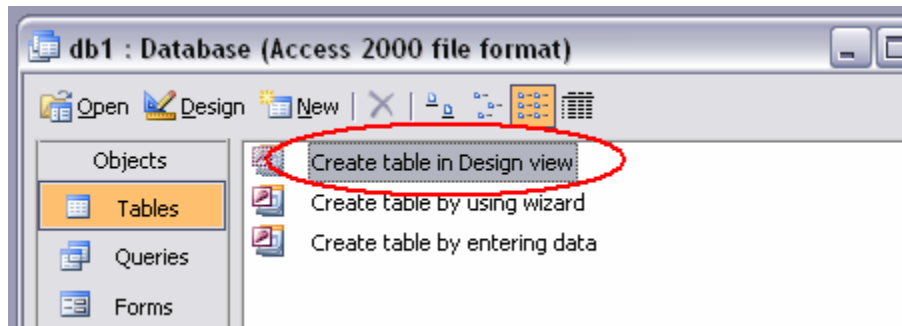
This will create a simple web application with only one file JSP in it i.e. index.jsp. Edit this file by putting following code between `<html></html>` tag.

```
<html>
  <head>
    <title>My First Web Application</title>
  </head>
  <body>
    <form method="POST" action="/MyFirstApp/login">
      Login: <input type="text" name="login"><br>
      Password: <input type="password" name="pswd"><br>
      <input type="submit" value="Login">
    </form>
  </body>
</html>
```

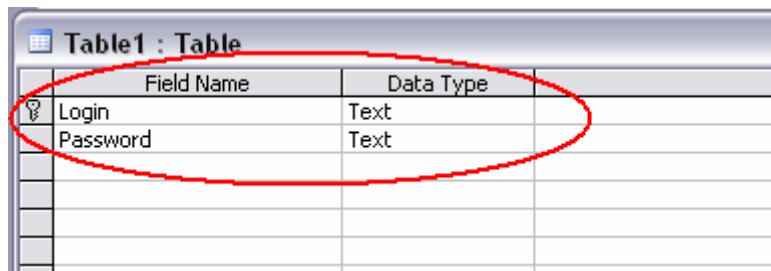
The `action="/MyFirstApp/login"` will allow your JSP page to send a request to the login servlet while `method="POST"` will invoke the `doPost` method in the `LoginServlet` class.

2. Create Database

Create a new Microsoft Access database file. Create new table



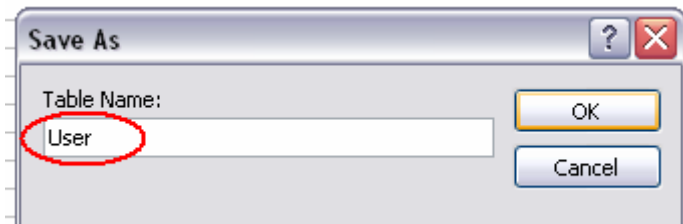
Make two columns *Login* and *Password*. Set *Login* as Primary Key of table



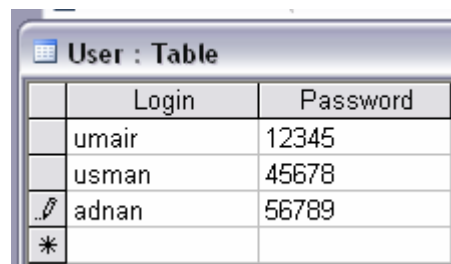
The screenshot shows the 'Table1 : Table' design grid. The 'Login' field is marked as the primary key with a key icon. Both 'Login' and 'Password' fields are set to 'Text' data type. A red circle highlights the first two rows of the table.

	Field Name	Data Type
PK	Login	Text
	Password	Text

Save the table as *Users*



Put some dummy values in table.



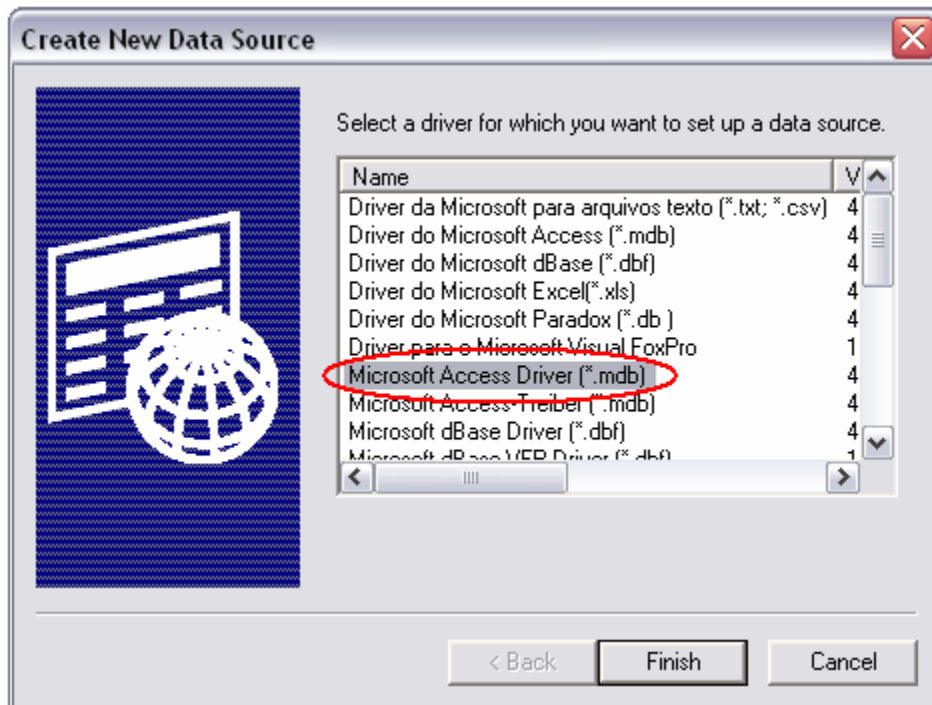
The screenshot shows the 'User : Table' data view. The table contains three rows of dummy data: 'umair' with password '12345', 'usman' with password '45678', and 'adnan' with password '56789'. A red circle highlights the first two rows.

	Login	Password
	umair	12345
	usman	45678
	adnan	56789
*		

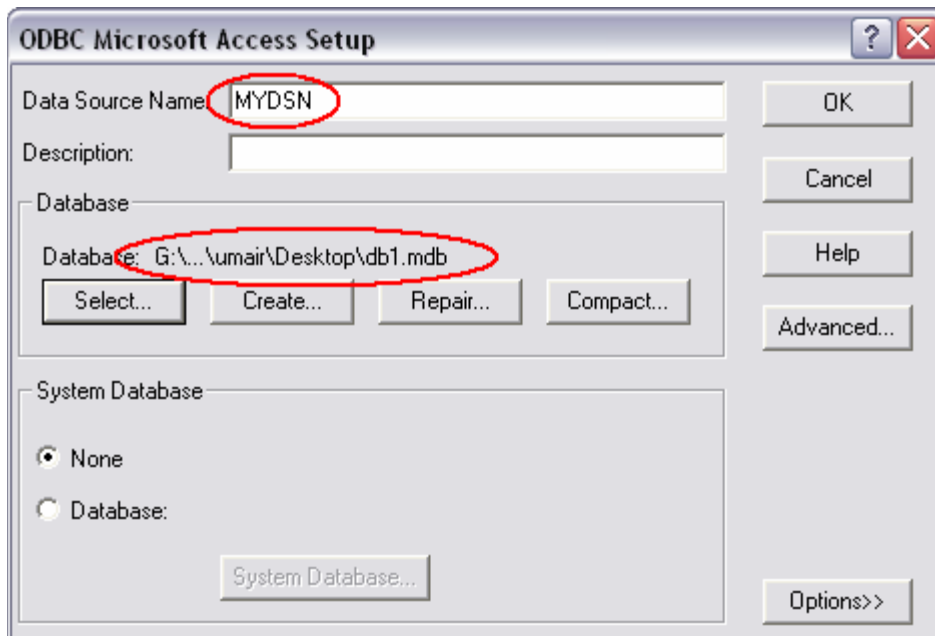
3. Create Data Source Name (DSN)

Go to *Control Panel >> Administrative Tools >> Data Sources (ODBC)*. Add a new *User DSN*.

Select *Microsoft Access Driver (*.mdb)* in drivers list and press *Finish* button.



Set *MYDSN* as *Data Source Name* and select the Microsoft Access file create in previous step as database using *Select* button

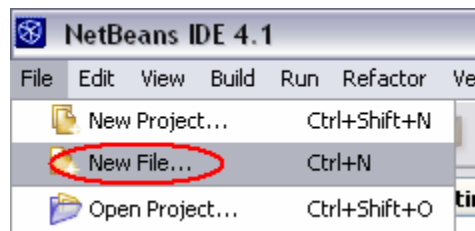


4. Create Verify Bean

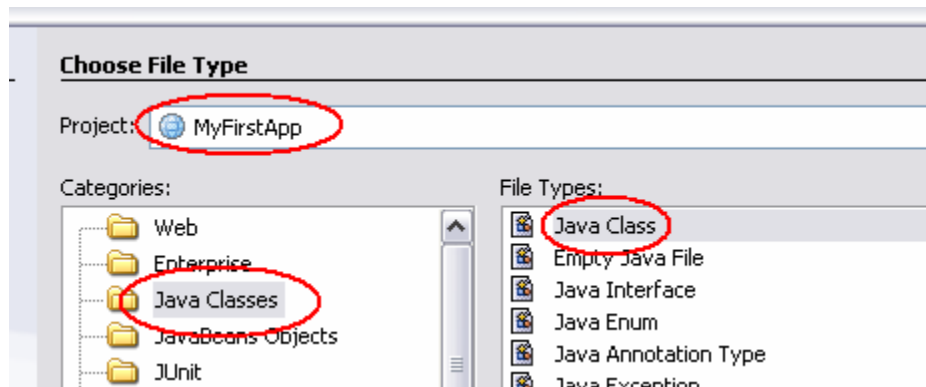
Java bean is a simple java class file built according to some standards. The minimum requirements for java bean are

- implements *Serializable* interface
- must have a default constructor
- provides get/set methods for all member variables

To create verify bean, select *New File* in *File* menu



In *New File* window, set *Project* as *MyFirstApp*, under *Categories* list choose *Java Classes* and select *Java Class* under *File Types* list



In *New Java Class* window, set *Class Name* as *VerifyBean* and *Package* to *lums*. Press *Finish* to create class

Name and Location

Class Name: VerifyBean

Project: MyFirstApp

Location: Source Packages

Package: lums

Created File: G:\Documents and Settings\umair\MyFirstApp\src\java\lums\VerifyBean.java

Put following code in VerifyBean class

```
package lums;

import java.io.*;
import java.sql.*;

public class VerifyBean implements Serializable{

    public VerifyBean(){

    }

    public boolean verify(String name, String pswd){

        try {

            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

            Connection con =
                DriverManager.getConnection("jdbc:odbc:MYDSN", "", "");

            Statement st = con.createStatement();

            ResultSet rs = st.executeQuery("SELECT * FROM User WHERE "
                +" Login='"+name+"' and Password='"+pswd+"'");

            if (rs.next())
                return true;

        }
        catch (Exception e) {

        }

        return false;

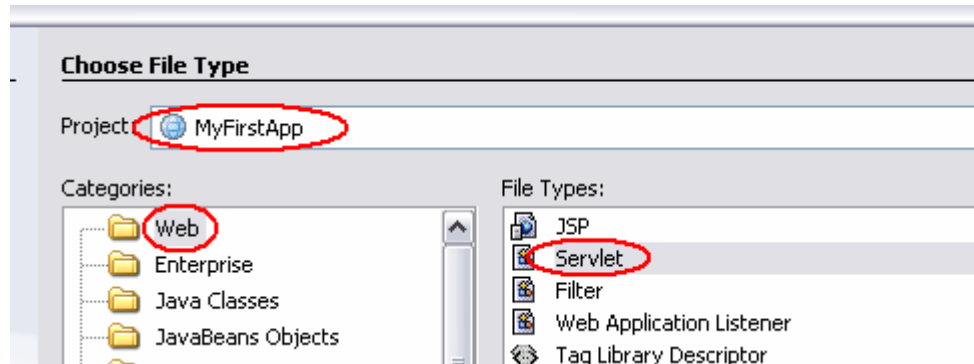
    }

}
```

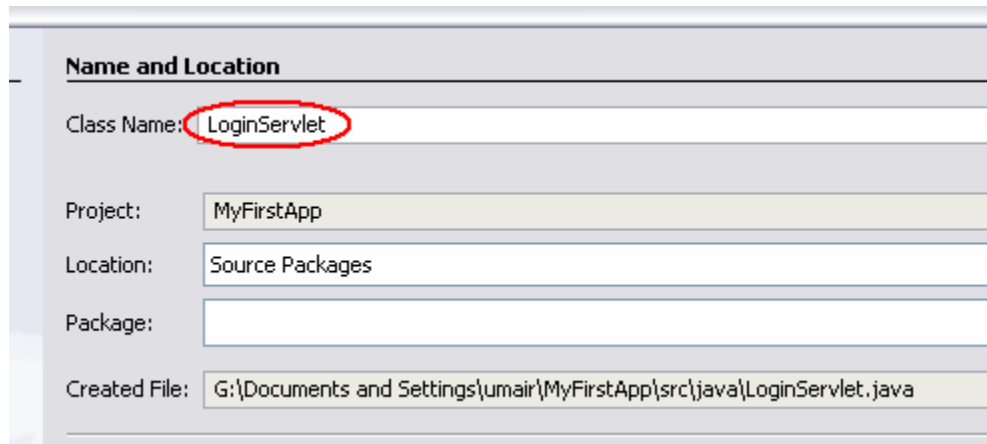
5. Create login servlet

Again select *New File* from *File* menu.

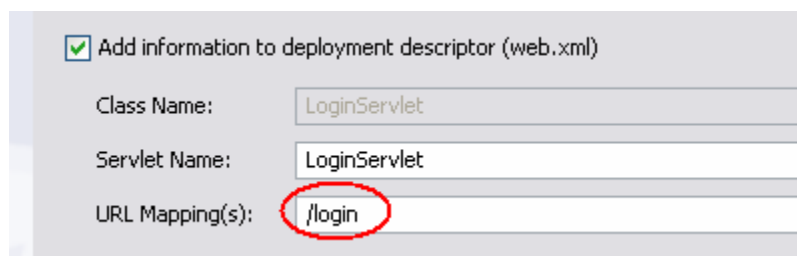
In *New File* window, set *Project* as *MyFirstApp*, under *Categories* list choose *Web* and select *Servlet* under *File Types* list



In *New Servlet* window, set *Class Name* as *LoginServlet* and leave *Package* field empty. Press *Next* button



Set */login* in *URL Mappings* field and press *Finish* to create servlet.



Put following code in login servlet class

```
import javax.servlet.http.*;  
import javax.servlet.*;
```

```
import java.io.*;
import lums.VerifyBean;

public class LoginServlet extends HttpServlet{

    public void doPost(HttpServletRequest req, HttpServletResponse res)
        throws ServletException,IOException{

        String f = req.getParameter("login");
        String s = req.getParameter("pswd");

        VerifyBean v = new VerifyBean();
        boolean verified = v.verify(f,s);

        if (verified) {

            RequestDispatcher disp = getServletContext().
                getRequestDispatcher("/loginsuccess.jsp");
            disp.forward(req,res);

        }
        else {

            RequestDispatcher disp = getServletContext().
                getRequestDispatcher("/loginfailure.jsp");
            disp.forward(req,res);

        }

    }

}
```

6. Run application

Create two new JSP files in main application folder named loginsuccess.jsp and loginfailure.jsp with suitable success and failure messages respectively. Press F6 to run and test the web application.