Java Servlets and java client App.

1. What are Java Servlets?

In the early days, web servers deliver *static* contents that are indifferent to users' requests. Java servlets are *server-side programs* (running inside a web server) that handle clients' requests and return a *customized* or *dynamic response* for each request. The dynamic response could be based on user's input (e.g., search, online shopping, online transaction) with data retrieved from databases or other applications, or time-sensitive data (such as news and stock prices).

Java servlets typically run on the HTTP protocol. HTTP is an *asymmetrical request-response protocol*. The client sends a *request message* to the server, and the server returns a *response message* as illustrated.

2. What are the Advantage of Servlets Over "Traditional" CGI?

Java servlets are more efficient, easier to use, more powerful, more portable, and cheaper than traditional CGI and than many alternative CGI-like technologies.

Efficient. With traditional CGI, a new process is started for each HTTP request. If the CGI program does a relatively fast operation, the overhead of starting the process can dominate the execution time. With servlets, the Java Virtual Machine stays up, and each request is handled by a lightweight Java thread, not a heavyweight operating system process. Similarly, in traditional CGI, if there are *N* simultaneous request to the same CGI program, then the code for the CGI program is loaded into memory N times. With servlets, however, there are *N* threads but only a single copy of the servlet class. Servlets also have more alternatives than do regular CGI programs for optimizations such as caching previous computations, keeping database connections open, and the like.

Convenient. Hey, you already know Java. Why learn Perl too? Besides the convenience of being able to use a familiar language, servlets have an extensive infrastructure for automatically parsing and decoding HTML form data, reading and setting HTTP headers, handling cookies, tracking sessions, and many other such utilities.

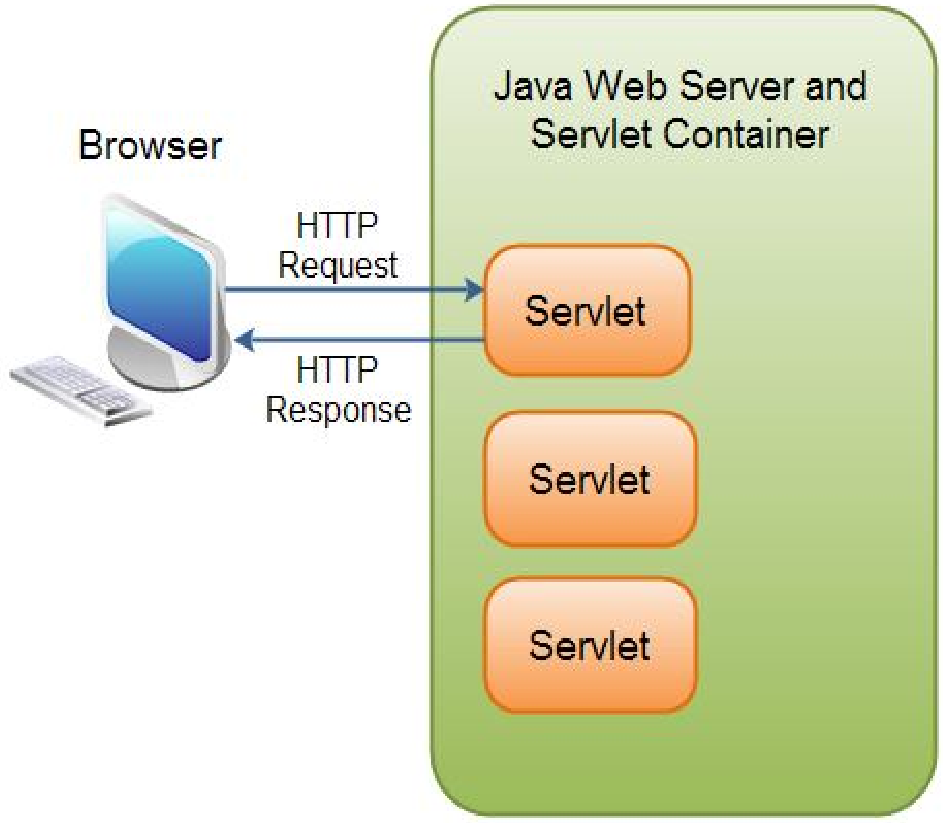
Powerful. Java servlets let you easily do several things that are difficult or impossible with regular CGI. For one thing, servlets can talk directly to the Web server (regular CGI programs can't). This simplifies operations that need to look up images and other data stored in standard places. Servlets can also share data among each other, making useful things like database connection pools easy to implement. They can also maintain information from request to request, simplifying things like session tracking and caching of previous computations.

Portable. Servlets are written in Java and follow a well-standardized API. Consequently, servlets written for, say I-Planet Enterprise Server can run virtually unchanged on Apache, Microsoft IIS, or WebStar. Servlets are supported directly or via a plugin on almost every major Web server.

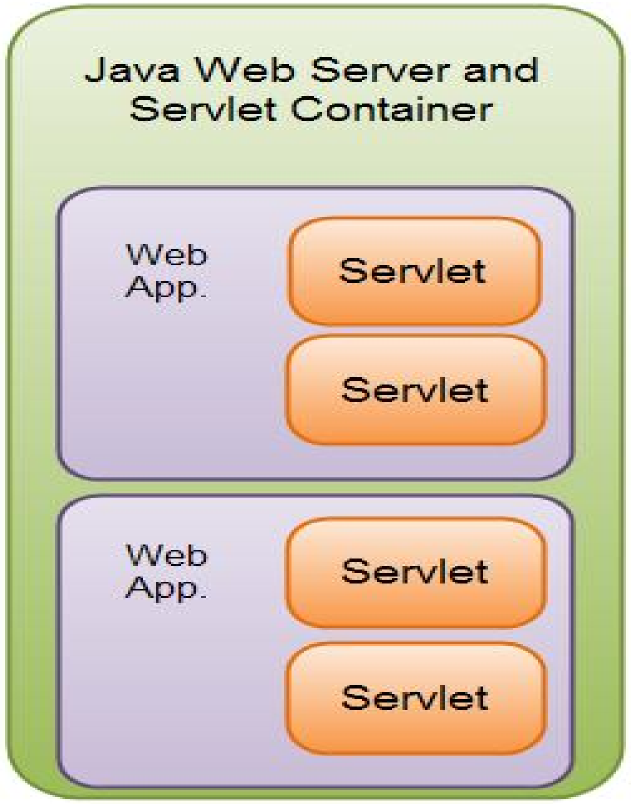
Inexpensive. There are a number of free or very inexpensive Web servers available that are good for "personal" use or low-volume Web sites. However, with the major exception of Apache, which is free, most commercial-quality Web servers are relatively expensive. Nevertheless, once you have a Web server, no matter the cost of that server, adding servlet support to it (if it doesn't come preconfigured to support servlets) is generally free or cheap.

Servlet Overview

- A Java Servlet is a Java object that responds to HTTP requests. It runs inside a Servlet container. Here is an illustration of that:



- A Servlet is part of a Java web application. A Servlet container may run multiple web applications at the same time, each having multiple servlets running inside. Here is an illustration of that:



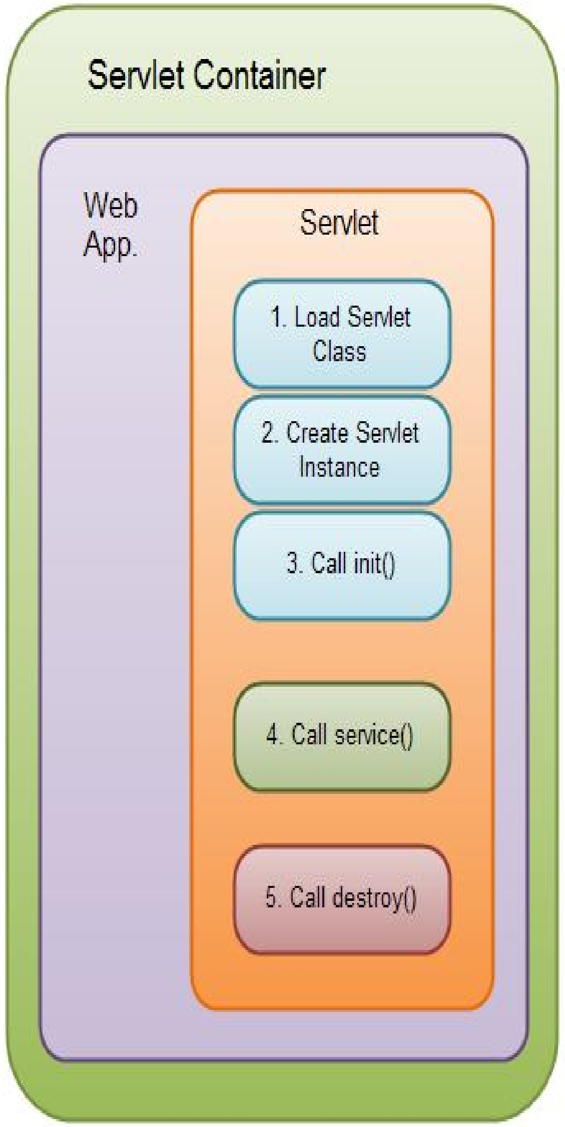
Web applications with multiple servlets inside a Java Servlet container

HTTP Request and Response

- The browser sends an HTTP request to the Java web server. The web server checks if the request is for a servlet. If it is, the servlet container is passed the request. The servlet container will then find out which servlet the request is for, and activate that servlet. The servlet is activated by calling the Servlet.service() method.
- Once the servlet has been activated via the service() method, the servlet processes the request, and generates a response. The response is then sent back to the browser.
- **Servlet Containers**
- Java servlet containers are usually running inside a Java web server. A few common well known, free Java web servers are:
- Jetty
- Tomcat

Servlet Life Cycle

- A servlet follows a certain life cycle. The servlet life cycle is managed by the servlet container. The life cycle contains the following steps:
- Load Servlet Class.
- Create Instance of Servlet.
- Call the servlets init() method.
- Call the servlets service() method.
- Call the servlets destroy() method.
- Step 1, 2 and 3 are executed only once, when the servlet is initially loaded. By default the servlet is not loaded until the first request is received for it. You can force the container to load the servlet when the container starts up though. See **web.xml Servlet Configuration** for more details about that.
- Step 4 is executed multiple times - once for every HTTP request to the servlet.
  Step 5 is executed when the servlet container unloads the servlet.
  Each step is described in more detail :



The Java Servlet life cycle

❖ **Load Servlet Class**

- Before a servlet can be invoked the servlet container must first load its class definition. This is done just like any other class is loaded.

❖ **Create Instance of Servlet**

- When the servlet class is loaded, the servlet container creates an instance of the servlet.
- Typically, only a single isntance of the servlet is created, and concurrent requests to the servlet are executed on the same servlet instance. This is really up to the servlet container to decide, though. But typically, there is just one instance.

❖ **Call the Servlets init() Method**

- When a servlet instance is created, its init() method is invoked. The init() method allows a servlet to initialize itself before the first request is processed.
- You can specify init parameters to the servlet in the web.xml file. See **web.xml Servlet Configuration** for more details.

❖ **Call the Servlets service() Method**

- For every request received to the servlet, the servlets service() method is called. ForHttpServlet subclasses, one of the doGet(), doPost() etc. methods are typically called.
- As long as the servlet is active in the servlet container, the service() method can be called. Thus, this step in the life cycle can be executed multiple times.

❖ **Call the Servlets destroy() Method**

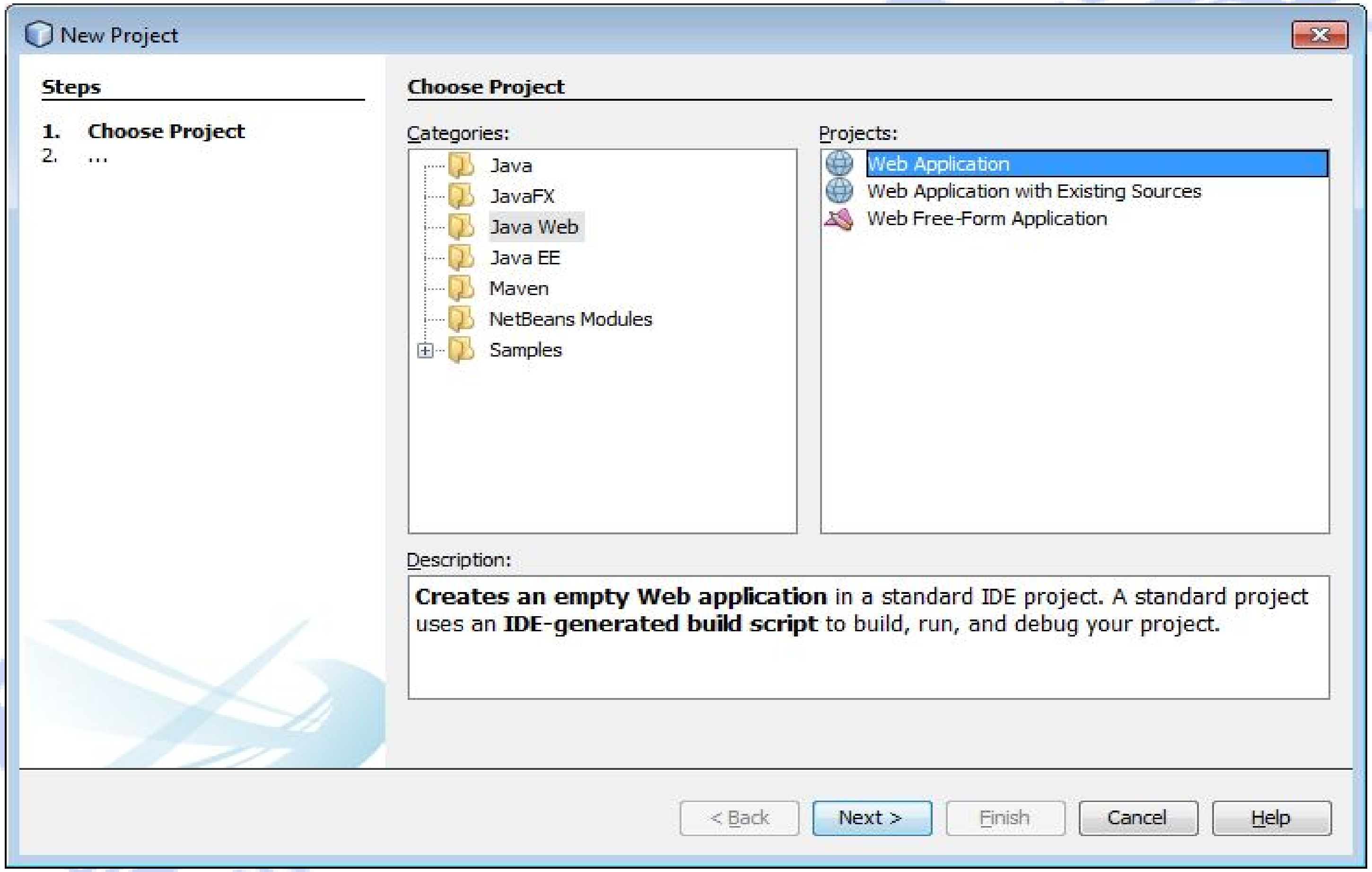
- When a servlet is unloaded by the servlet container, its destroy() method is called. This step is only executed once, since a servlet is only unloaded once.
- A servlet is unloaded by the container if the container shuts down, or if the container reloads the whole web application at runtime.

For more details you also refer links below

- do read
- **Networking** and **Servlets**
  chapters from the book: **Java Complete Reference**.
- **Basic Servlet Tutorial**
  ttp://www.apl.jhu.edu/~hall/java/Servlet-Tutorial/
- **Servlets Working Theory:**
  http://www.cse.iitb.ac.in/dbms/Data/Courses/DBIS/Software/servlets/servlet_tutorial.html
- **Implementing Servlets In NetBeans 5.0 and ahead (you can use latest version of net beans as well)**
  http://www.java-tips.org/java-tutorials/tutorials/introduction-to-java-servlets-with-netbeans.html

Now Lets Start The Implementation Of Servlet Using IDE Netbeans

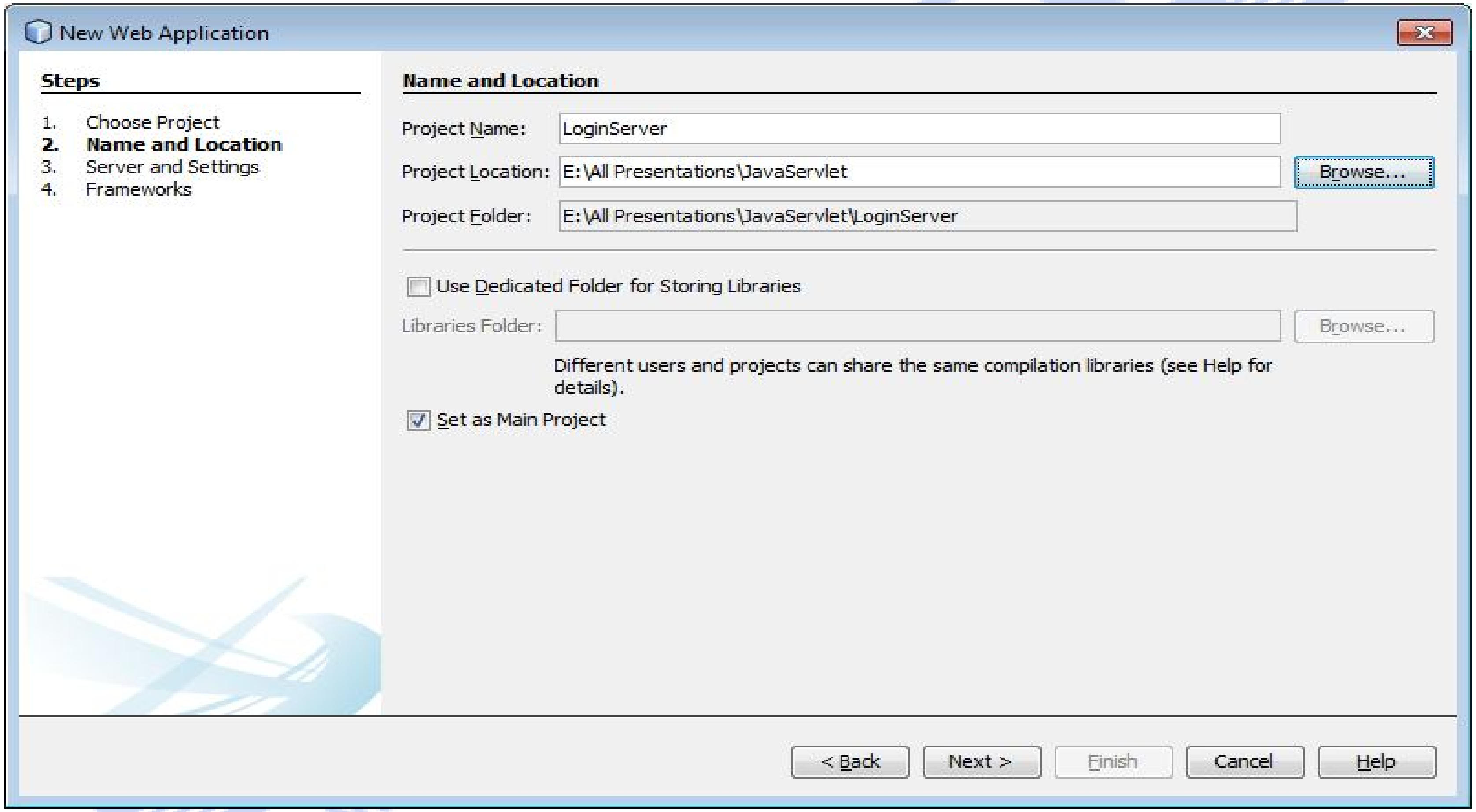
- Step 1: Creating Server.



Creates an empty Web application in a standard IDE project. A standard project uses an **IDE-generated build script** to build, run, and debug your project.

Click File→New Project →Java Web→Web Application.

- Click Next
- Assign the server application a name as given "LoginServer" and choose the project location where you want to save the project.



New Web Application

Steps

1. Choose Project
2. **Name and Location**
3. Server and Settings
4. Frameworks

Name and Location

Project Name:	LoginServer
Project Location:	E:\All Presentations\JavaServlet
Project Folder:	E:\All Presentations\JavaServlet\LoginServer

☐ Use Dedicated Folder for Storing Libraries

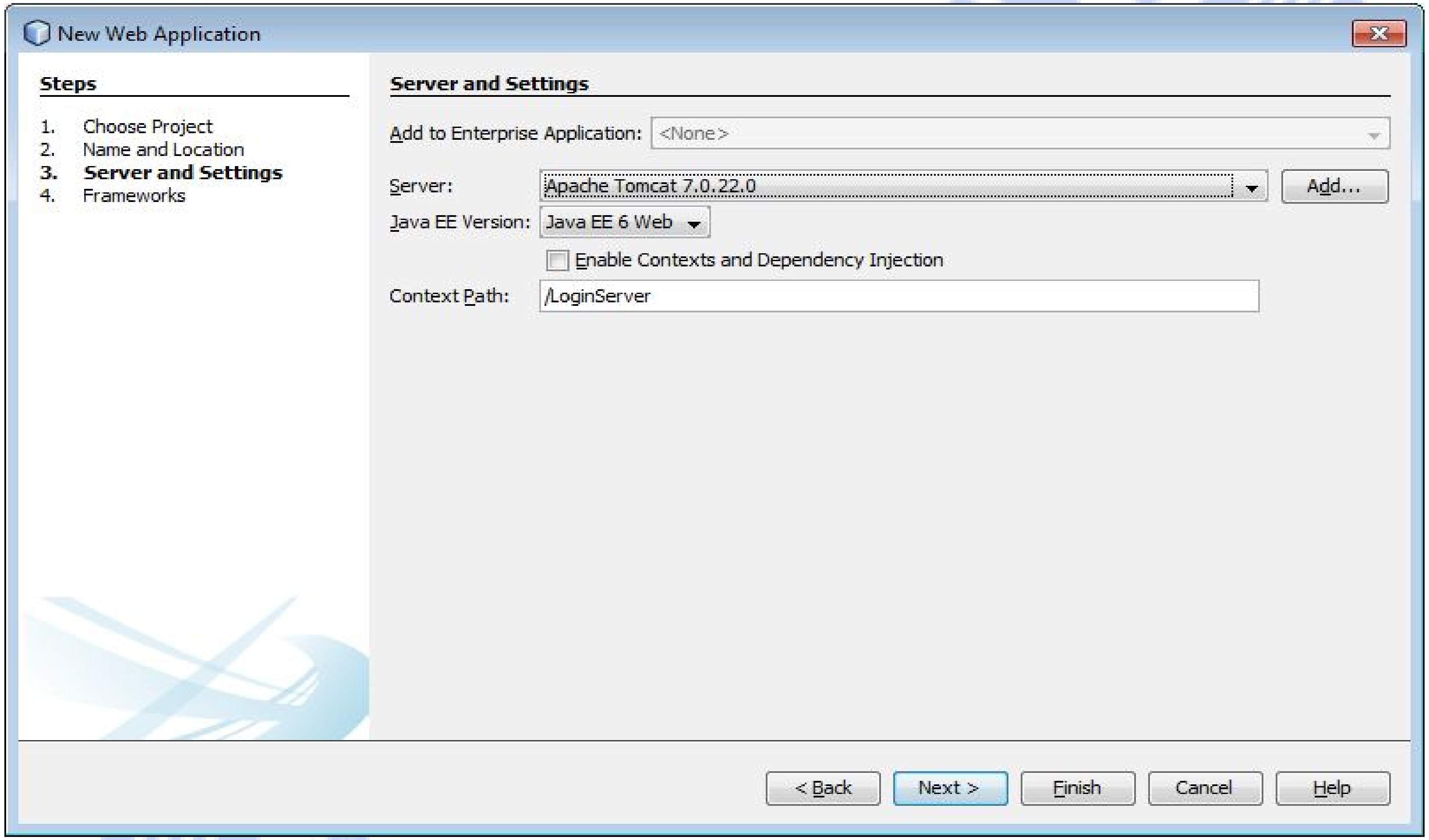
Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

☑ Set as Main Project

Click Next.

Select Apache tomcat as a server where our server application will be deployed.



Add to Enterprise Application: <None>

Server: Apache Tomcat 7.0.22.0

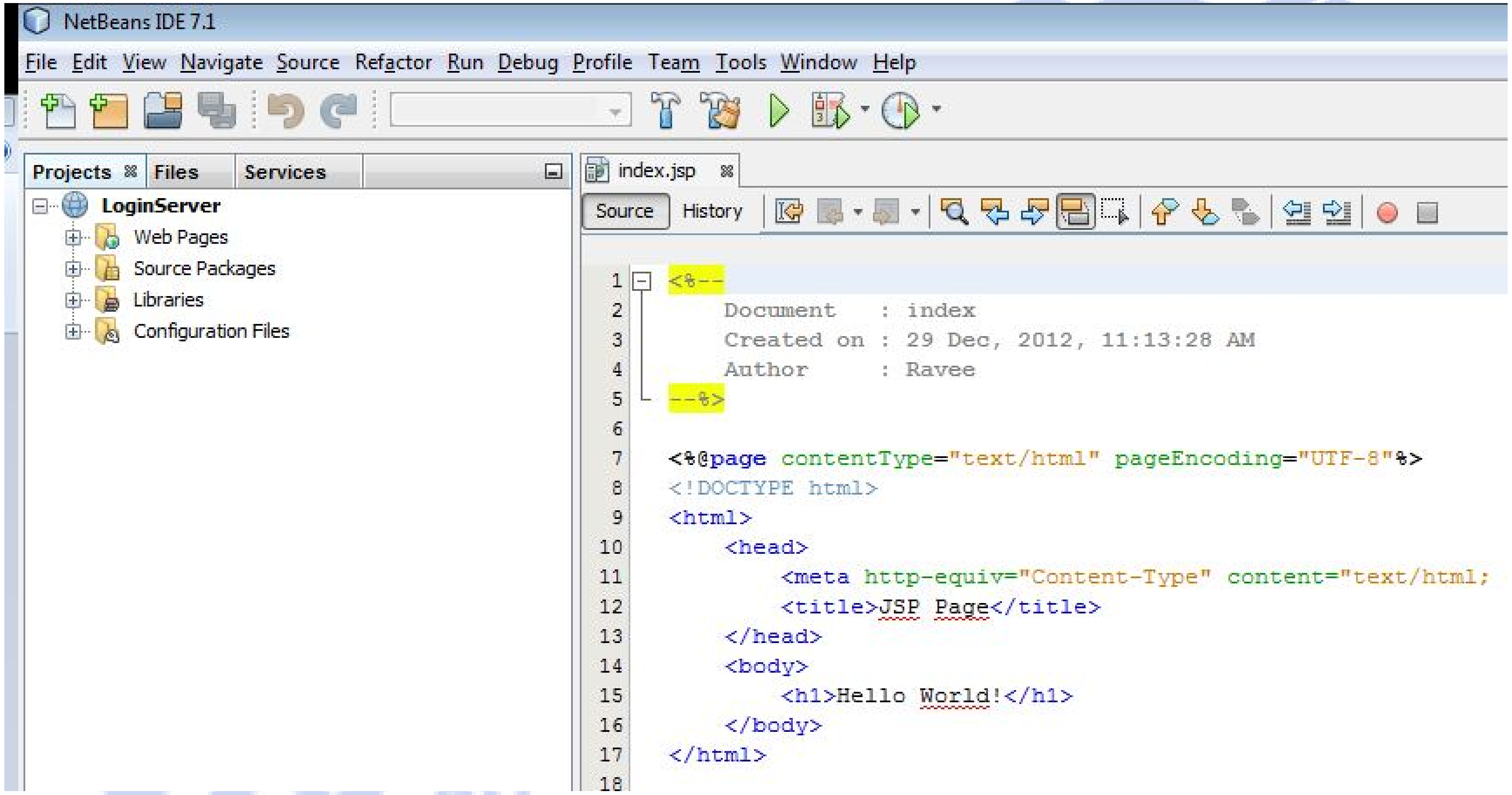
Java EE Version: Java EE 6 Web

Enable Contexts and Dependency Injection

Context Path: /LoginServer

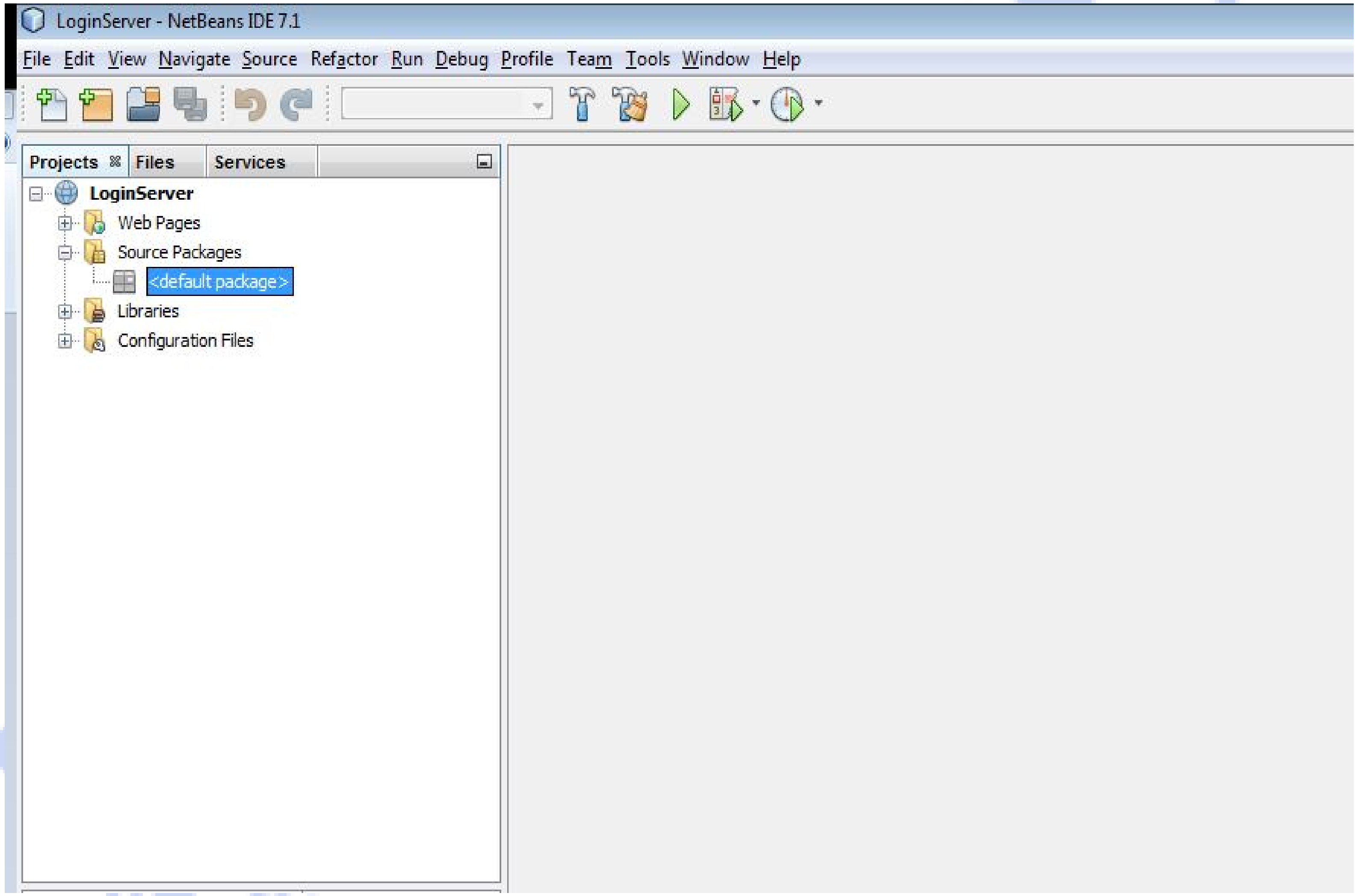
Apache Tomcat is preferred for servlet application, so we have selected it. Click Finish.

An application as shown in the image will be created. Close index.jsp



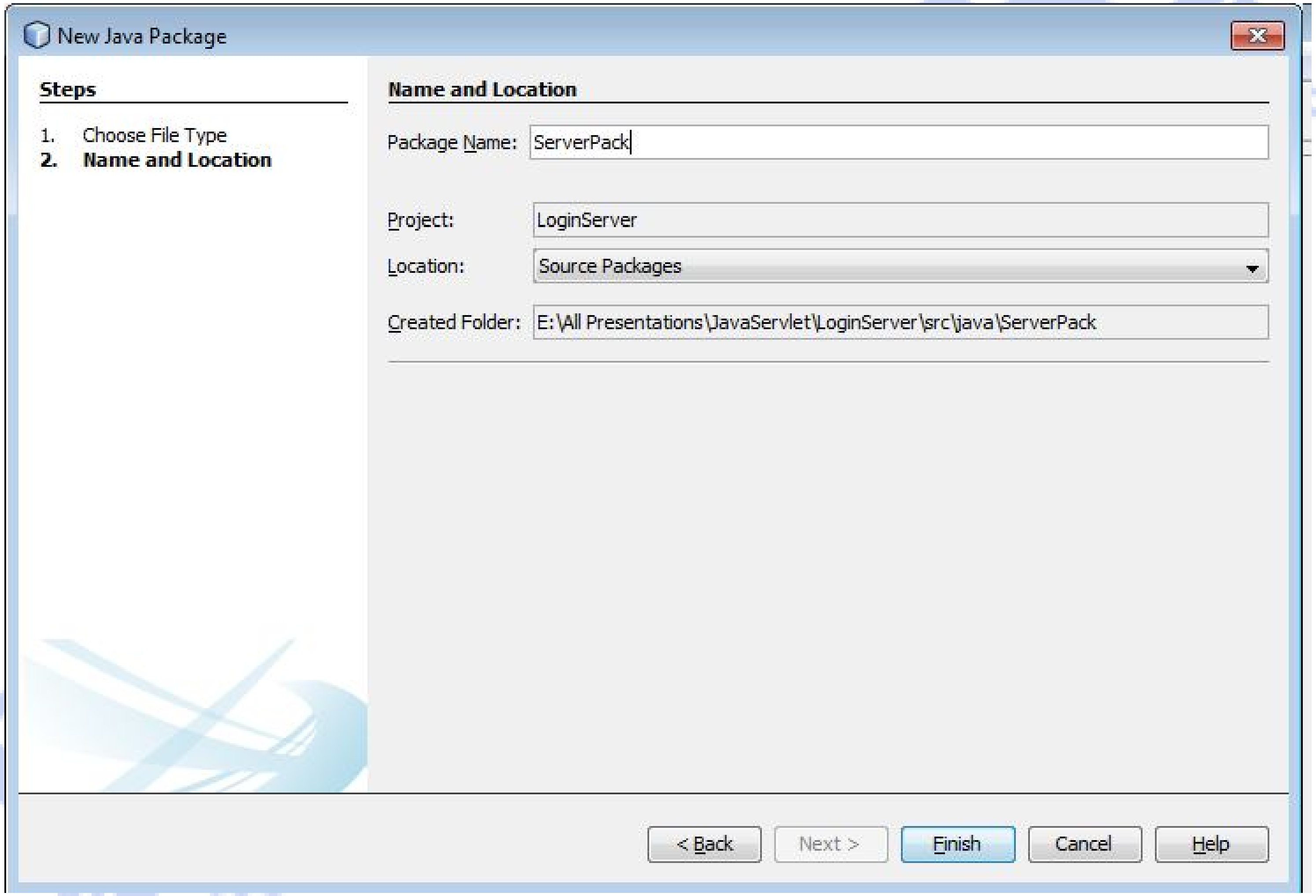
```
<%--
    Document   : index
    Created on : 29 Dec, 2012, 11:13:28 AM
    Author     : Ravee
--%>

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Hello World!</h1>
    </body>
</html>
```

- Now Right click on default package.


LoginServer - NetBeans IDE 7.1
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Projects
Files
Services
LoginServer
Web Pages
Source Packages
<default package>
Libraries
Configuration Files

Click On New→Java Package.

A package is collection of classes.
If talk about it at operating system level its just a folder for managing our classes.


New Java Package

Steps
1. Choose File Type
2. **Name and Location**

Name and Location

Package Name: ServerPack

Project: LoginServer

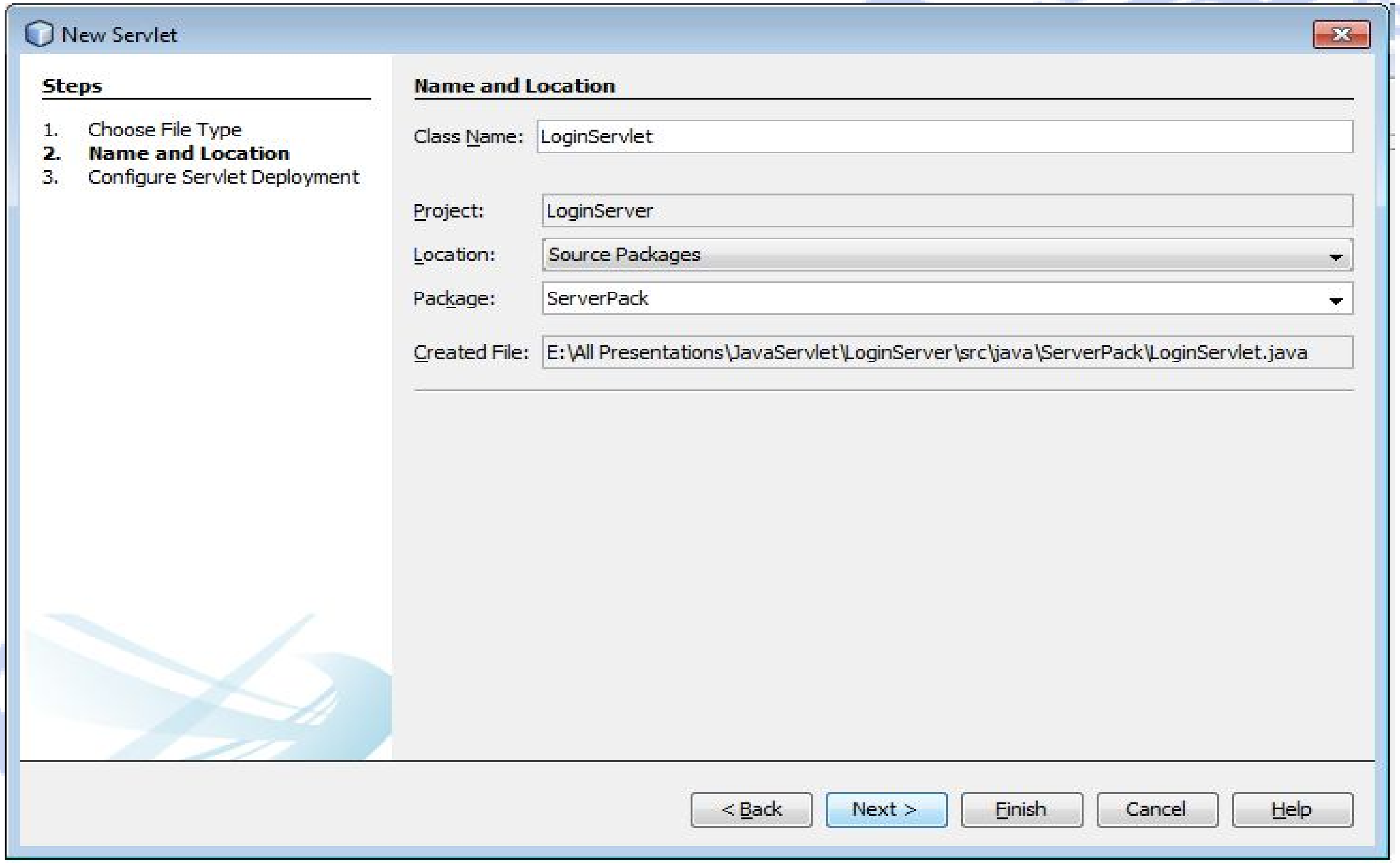
Location: Source Packages

Created Folder: E:\All Presentations\JavaServlet\LoginServer\src\java\ServerPack

Assign package a name as assigned “ServerPack”. Click Finish.

Now lets add servlets to the created package.

- Right click on serverPack new→ select servlet.



Class Name: LoginServlet

Project: LoginServer

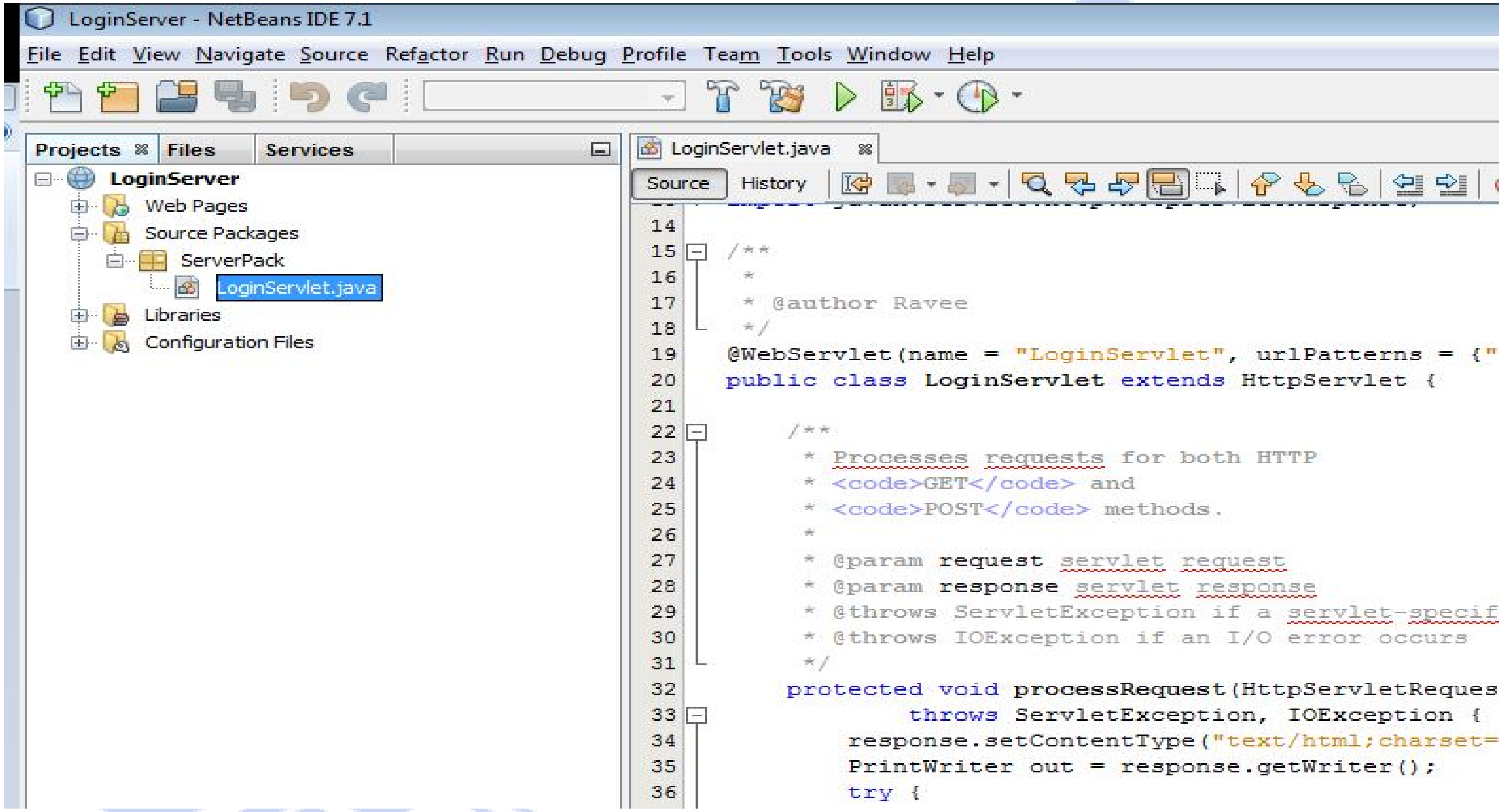
Location: Source Packages

Package: ServerPack

Created File: E:\All Presentations\JavaServlet\LoginServer\src\java\ServerPack\LoginServlet.java

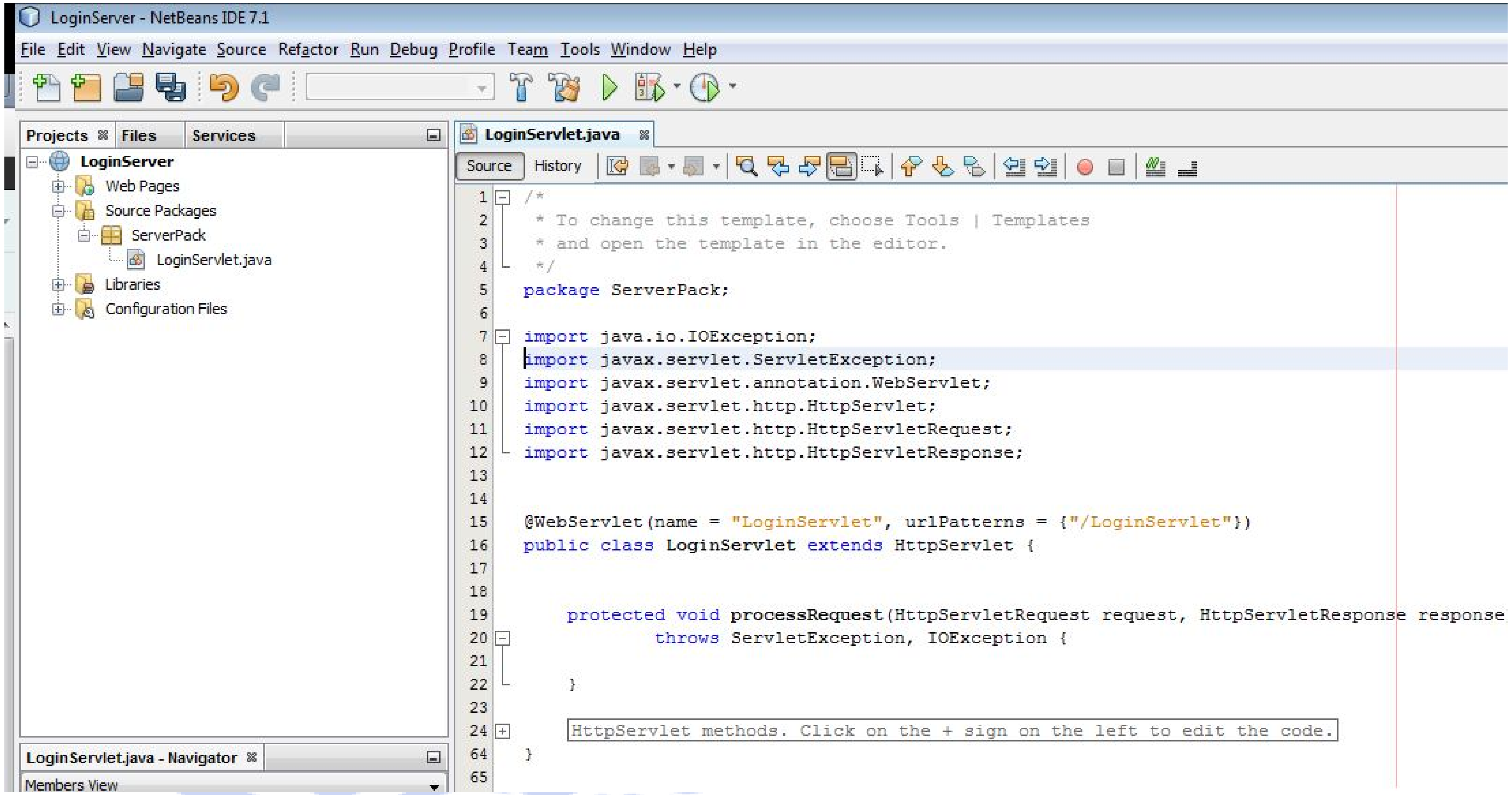
Assign servlet a name. As given “LoginServlet”. Click on finish.

A LoginServlet.java file will be created as shown in the image.



```
/**
 *
 * @author Ravee
 */
@WebServlet(name = "LoginServlet", urlPatterns = {"
public class LoginServlet extends HttpServlet {

    /**
     * Processes requests for both HTTP
     * <code>GET</code> and
     * <code>POST</code> methods.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specif
     * @throws IOException if an I/O error occurs
     */
    protected void processRequest(HttpServletReques
            throws ServletException, IOException {
        response.setContentType("text/html;charset=
        PrintWriter out = response.getWriter();
        try {
```

Now delete the unwanted block of code in process request function. After deleting it will look like.



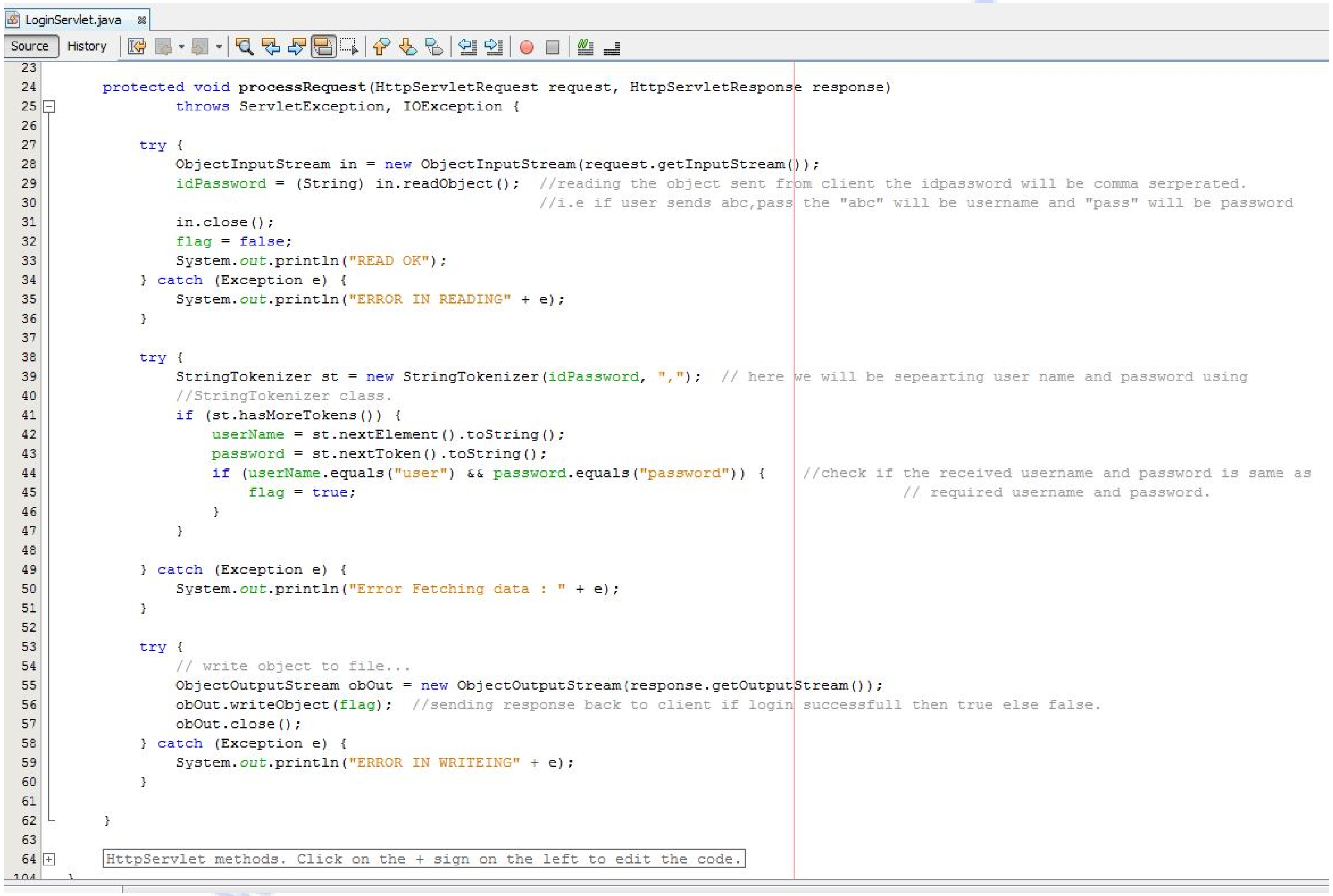
```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package ServerPack;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


@WebServlet(name = "LoginServlet", urlPatterns = {"/LoginServlet"})
public class LoginServlet extends HttpServlet {


    protected void processRequest(HttpServletRequest request, HttpServletResponse response
            throws ServletException, IOException {

    }

    HttpServlet methods. Click on the + sign on the left to edit the code.
}
```

Now we will be writing code for checking username as “user” and password as “password”

Why create the class library ?

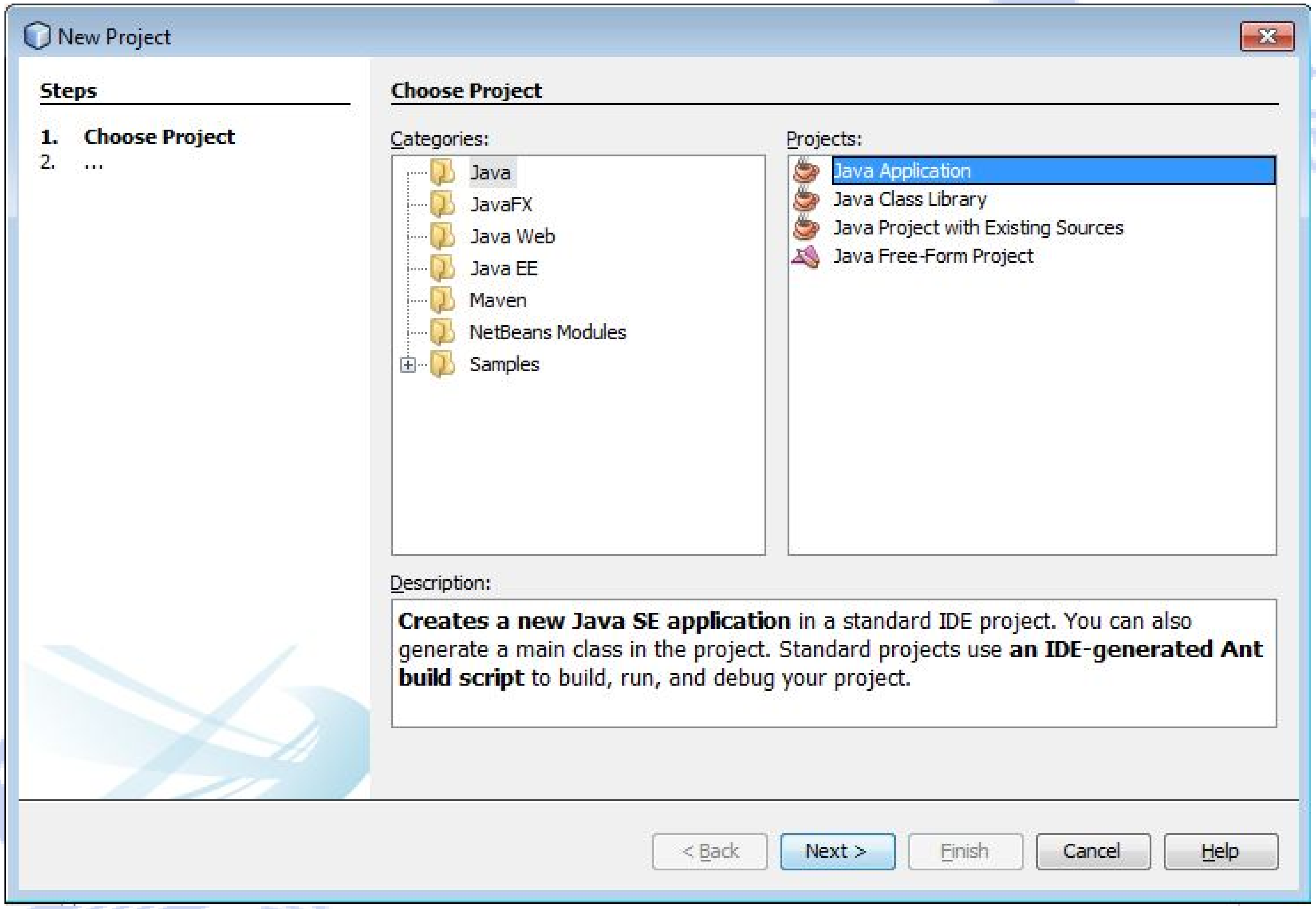
- A class library is a collection of class. And this classes will only contain the structure of the database for the particular entity.
- For example:- As we are creating login application the required information for login is login ID and password. So a class named “loginInfo” will be created a it will contain this to variable.
- When actually logging in we will be creating object of this class and passing it to server. For this class has to serializable.

Writing code for authenticating.....



```java
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        try {
            ObjectInputStream in = new ObjectInputStream(request.getInputStream());
            idPassword = (String) in.readObject();  //reading the object sent from client the idpassword will be comma serperated.
                                                    //i.e if user sends abc,pass the "abc" will be username and "pass" will be password
            in.close();
            flag = false;
            System.out.println("READ OK");
        } catch (Exception e) {
            System.out.println("ERROR IN READING" + e);
        }

        try {
            StringTokenizer st = new StringTokenizer(idPassword, ",");  // here we will be sepearting user name and password using
            //StringTokenizer class.
            if (st.hasMoreTokens()) {
                userName = st.nextElement().toString();
                password = st.nextToken().toString();
                if (userName.equals("user") && password.equals("password")) {    //check if the received username and password is same as
                    flag = true;                                                           // required username and password.
                }
            }

        } catch (Exception e) {
            System.out.println("Error Fetching data : " + e);
        }

        try {
            // write object to file...
            ObjectOutputStream obOut = new ObjectOutputStream(response.getOutputStream());
            obOut.writeObject(flag);  //sending response back to client if login successfull then true else false.
            obOut.close();
        } catch (Exception e) {
            System.out.println("ERROR IN WRITEING" + e);
        }

    }

    HttpServlet methods. Click on the + sign on the left to edit the code.
}
```

Now create a client application in java to call this servlet.

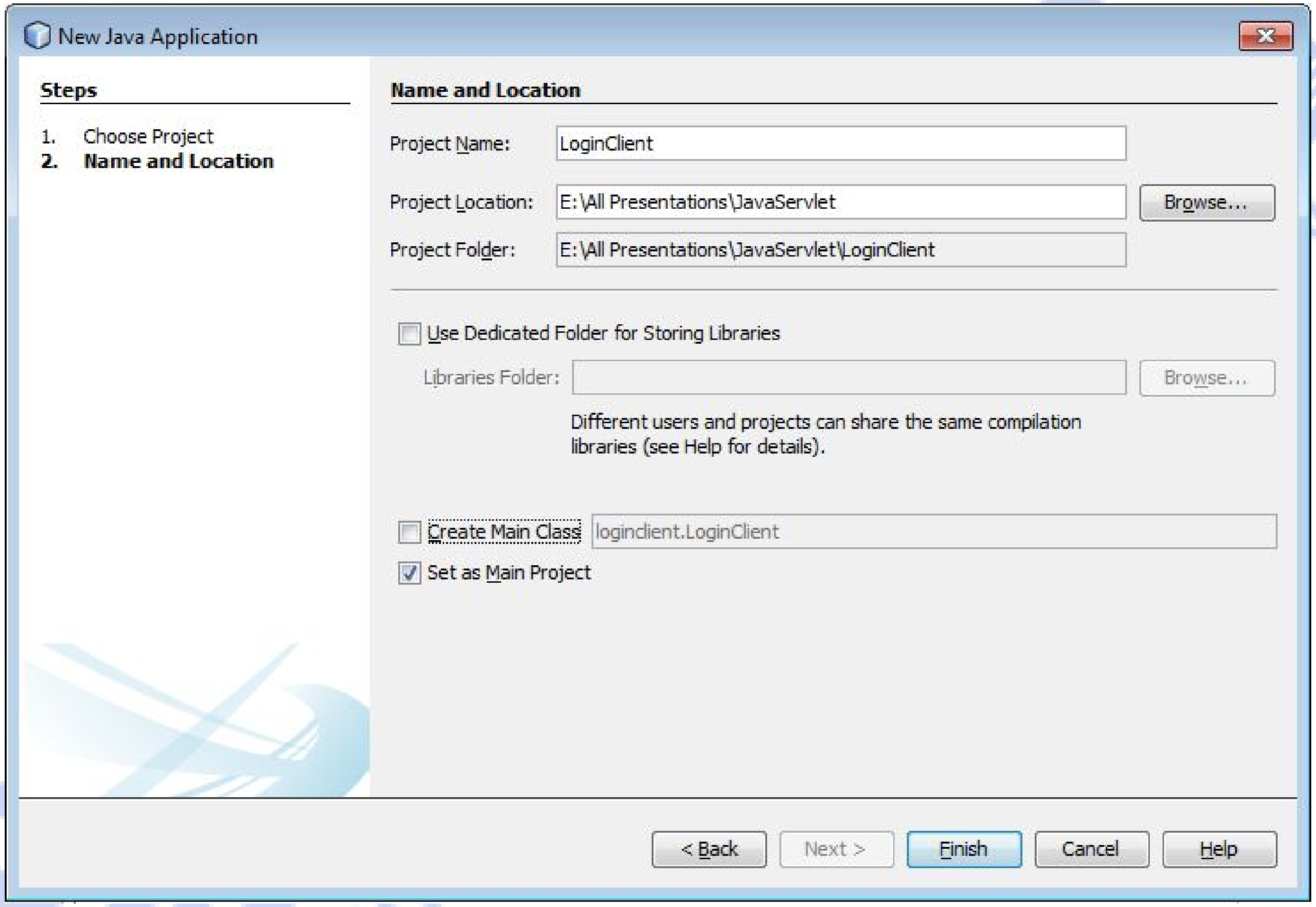


Description:

Creates a new Java SE application in a standard IDE project. You can also generate a main class in the project. Standard projects use **an IDE-generated Ant build script** to build, run, and debug your project.

Click Next.

Now assign a name to the client app.


New Java Application

Steps

1. Choose Project
2. **Name and Location**

Name and Location

Project Name: LoginClient

Project Location: E:\All Presentations\JavaServlet

Project Folder: E:\All Presentations\JavaServlet\LoginClient

[ ] Use Dedicated Folder for Storing Libraries

Libraries Folder:

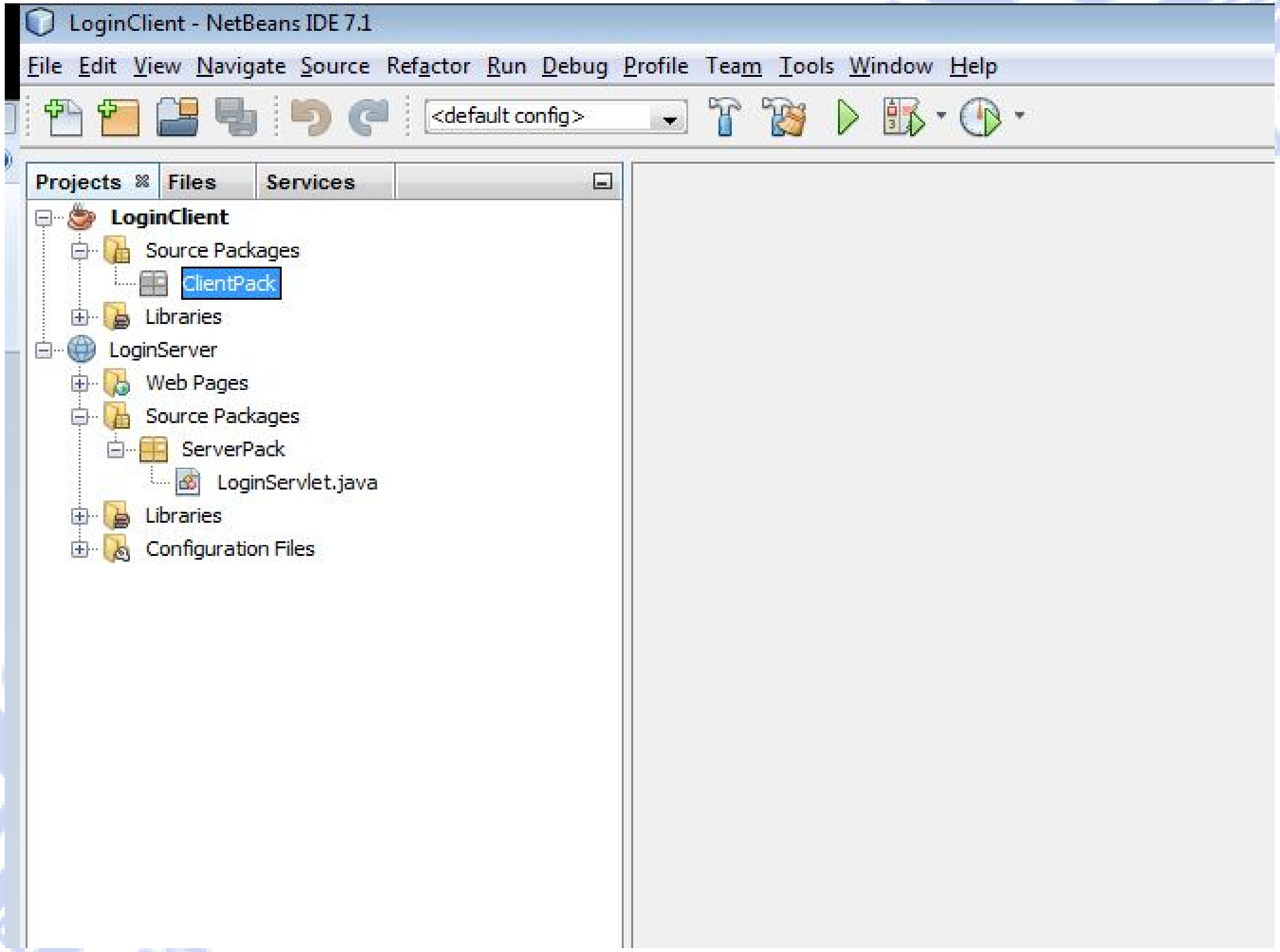
Different users and projects can share the same compilation libraries (see Help for details).

[ ] Create Main Class logindient.LoginClient

[✓] Set as Main Project

Click Finish.

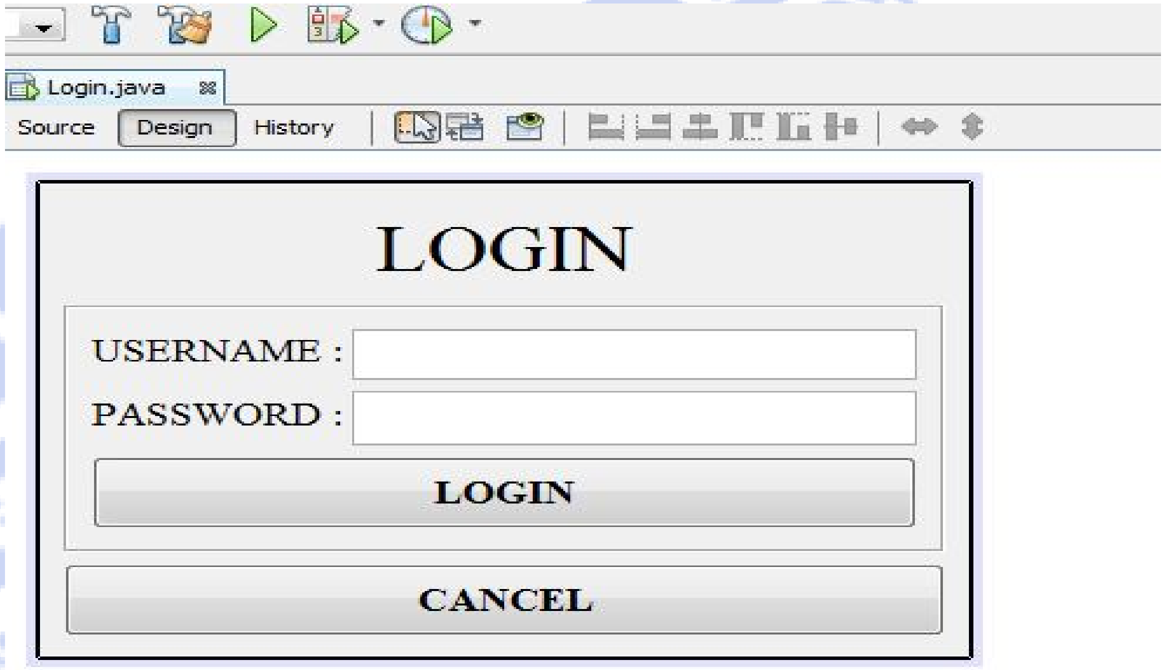
Create a package as created in the previous module.


LoginClient - NetBeans IDE 7.1
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config>
Projects
Files
Services

```
LoginClient
  Source Packages
    ClientPack
  Libraries
LoginServer
  Web Pages
  Source Packages
    ServerPack
      LoginServlet.java
  Libraries
  Configuration Files
```

Now add Jframe Form to the clientPack.

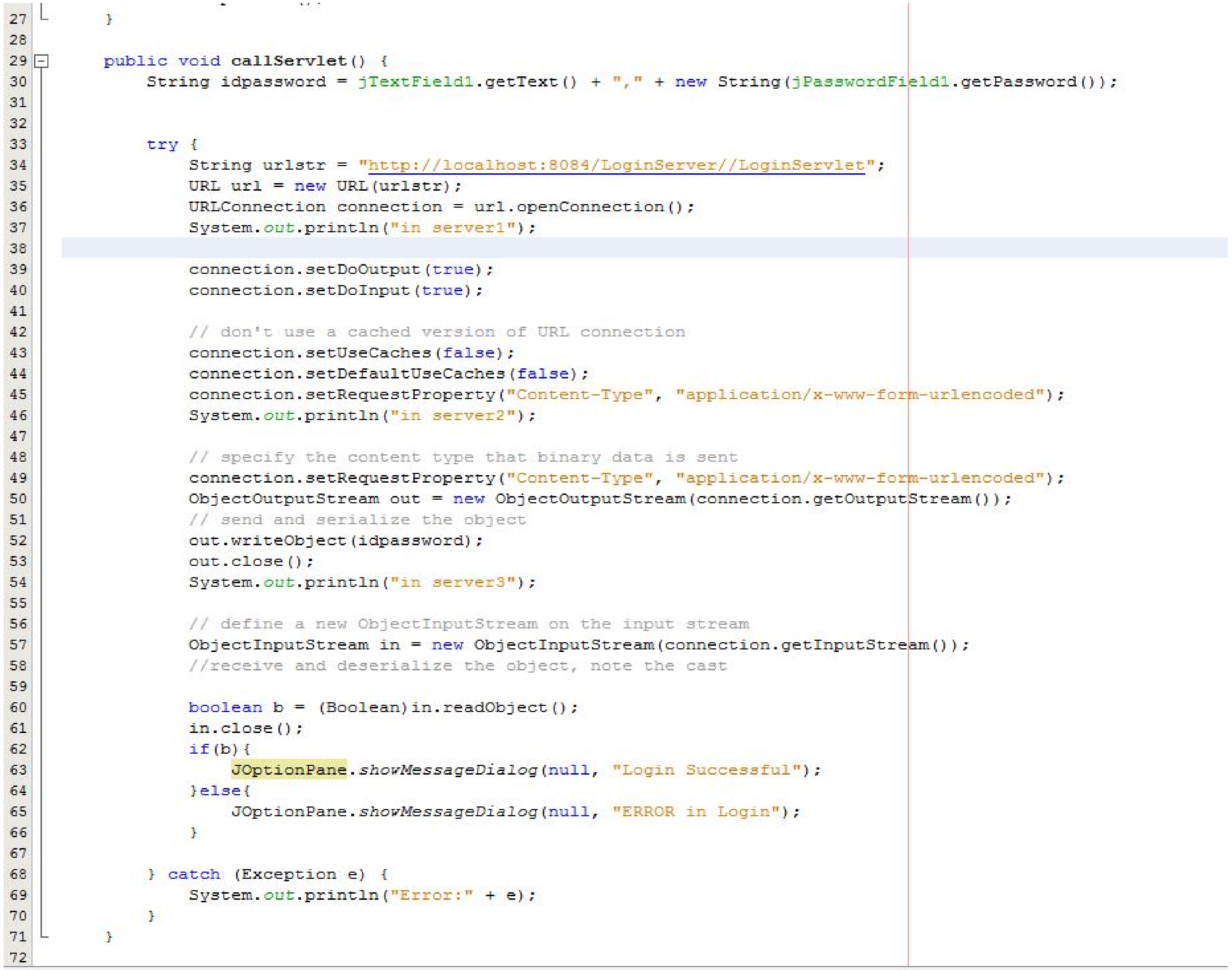
- Right click on clientpack new→Jframe Form.
- Assign a name to it as assigned “Login”.
- Now create GUI as shown Below.



Lets deploy server.

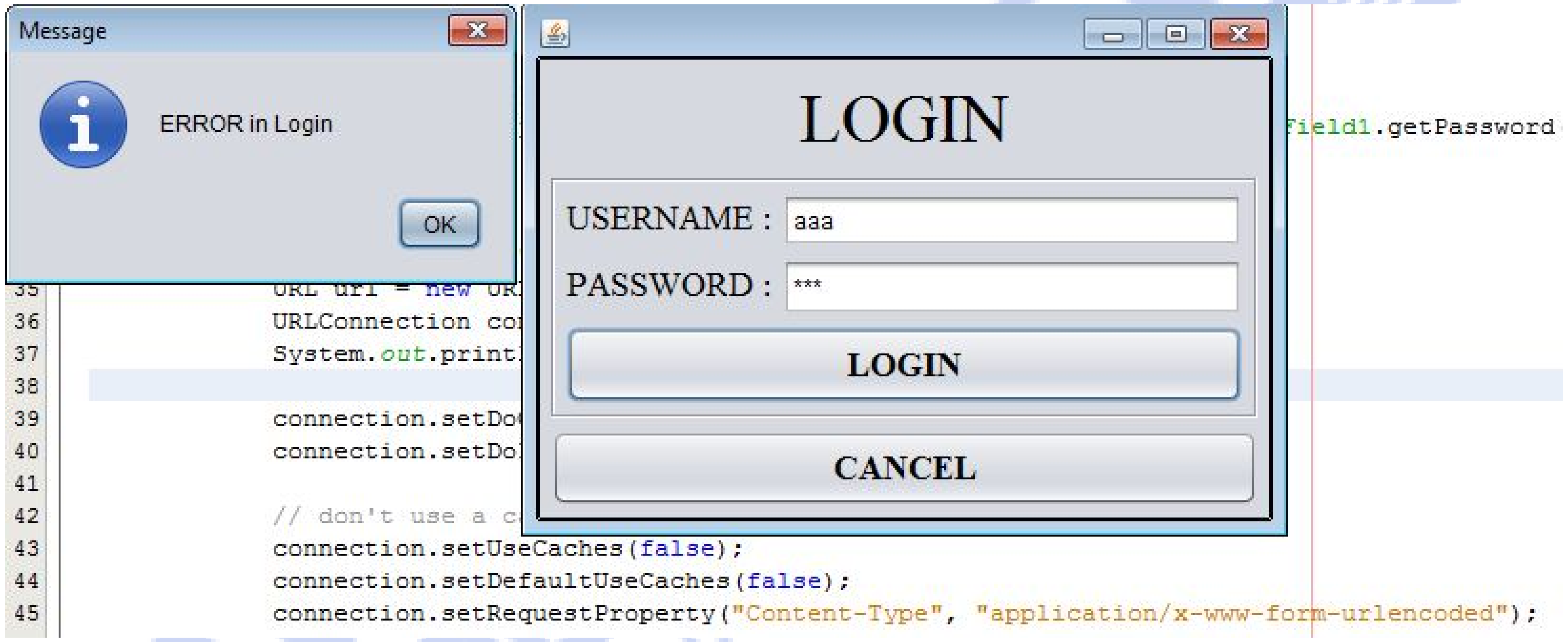
- Right click on LoginServer →clean and build→deploy.

Writing code to call the servlet.
Below is function created callServlet(), this function contains the code to call the servlet. Just call this function in click of login button.



```java
    }

    public void callServlet() {
        String idpassword = jTextField1.getText() + "," + new String(jPasswordField1.getPassword());


        try {
            String urlstr = "http://localhost:8084/LoginServer//LoginServlet";
            URL url = new URL(urlstr);
            URLConnection connection = url.openConnection();
            System.out.println("in server1");

            connection.setDoOutput(true);
            connection.setDoInput(true);

            // don't use a cached version of URL connection
            connection.setUseCaches(false);
            connection.setDefaultUseCaches(false);
            connection.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
            System.out.println("in server2");

            // specify the content type that binary data is sent
            connection.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
            ObjectOutputStream out = new ObjectOutputStream(connection.getOutputStream());
            // send and serialize the object
            out.writeObject(idpassword);
            out.close();
            System.out.println("in server3");

            // define a new ObjectInputStream on the input stream
            ObjectInputStream in = new ObjectInputStream(connection.getInputStream());
            //receive and deserialize the object, note the cast

            boolean b = (Boolean)in.readObject();
            in.close();
            if(b){
                JOptionPane.showMessageDialog(null, "Login Successful");
            }else{
                JOptionPane.showMessageDialog(null, "ERROR in Login");
            }

        } catch (Exception e) {
            System.out.println("Error:" + e);
        }
    }

```

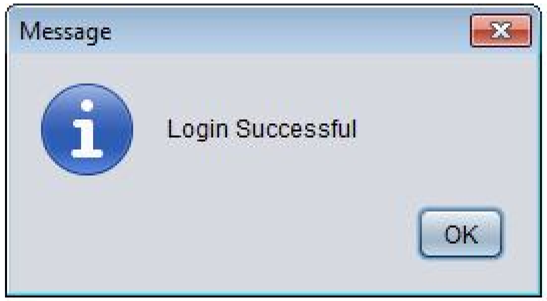
Now here is the output.



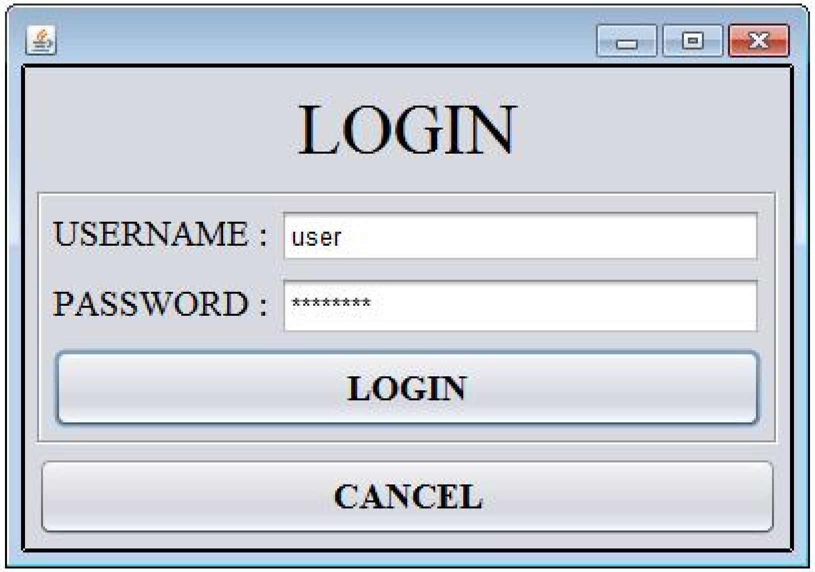
```
                                                        ield1.getPassword

            URL url = new UR
            URLConnection co
            System.out.print

            connection.setDo
            connection.setDo

            // don't use a c
            connection.setUseCaches(false);
            connection.setDefaultUseCaches(false);
            connection.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
```

When given any random user or password.

When given USERNAME as “user” and password as “password”.


Message

Login Successful


LOGIN

USERNAME : user

PASSWORD : ********

So we have created and tested a login servlet here. There is a working module for the same attached with this pdf for refer. If you face any issue in development you can refer it.....