1. Create a simple login form using JavaFX.

This Java servlet program will display "Hello, World!" in the browser. It demonstrates the use of a simple servlet to handle HTTP requests and send responses back to the browser.  
For this I first defined a class HelloServlet as:

package com.example.helloworld;

import java.io.\*;

import jakarta.servlet.ServletException;

import jakarta.servlet.http.\*;

import jakarta.servlet.annotation.\*;

@WebServlet(name = "helloServlet", value = "/hello")

public class HelloServlet extends HttpServlet {

@Override

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

response.setContentType("text/html");

// Get the output stream to write the response

PrintWriter out = response.getWriter();

// Display "Hello, World!" in the browser

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

out.println("<h1>Hello, World!</h1>");

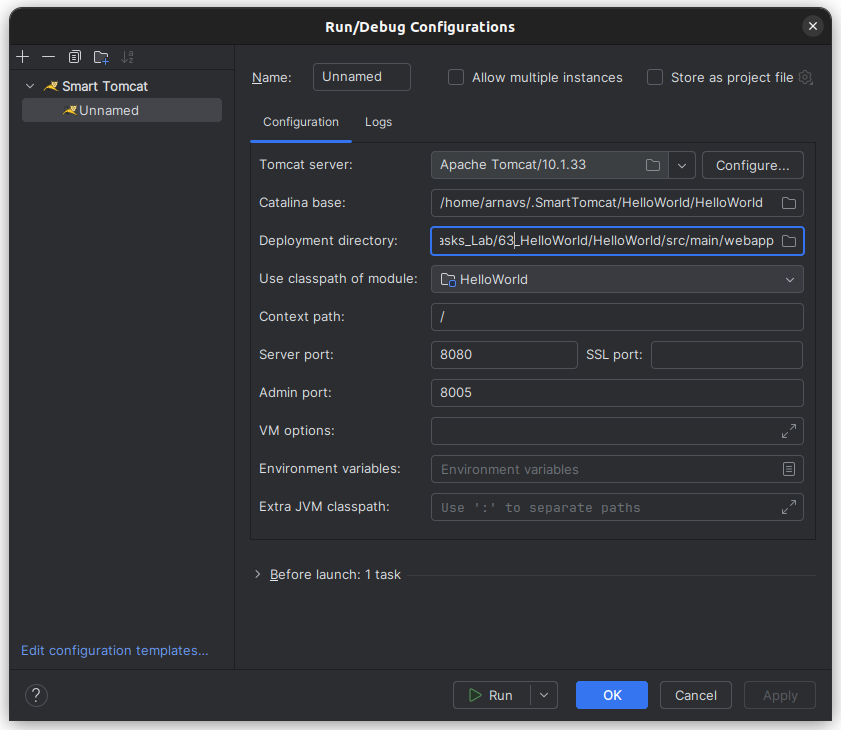
out.println("</body></html>");

}

}

And then defined a web.xml file as:  
<web-app xmlns="https://jakarta.ee/xml/ns/jakartaee"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee https://jakarta.ee/xml/ns/jakartaee/web-app\_6\_0.xsd"  
 version="6.0">  
</web-app>

Then, I configured Apache Tomacat server as:



And on running the application, we can see the result as:  


1. Create a servlet that processes a form submission and displays the submitted data (e.g., name, email).

This servlet program processes form submissions, extracting and displaying the submitted data, such as the user's name and email. It demonstrates handling HTTP POST requests in a servlet to receive and respond with user input.

form.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Form Submission</title>

</head>

<body>

<h1>Submit Your Details</h1>

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

Name: <input type="text" name="name"><br><br>

Email: <input type="email" name="email"><br><br>

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

</form>

</body>

</html>

FormServlet.java

package com.example.formservlet;

import jakarta.servlet.\*;

import jakarta.servlet.http.\*;

import java.io.IOException;

import java.io.PrintWriter;

public class FormServlet extends HttpServlet {

@Override

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

String name = request.getParameter("name");

String email = request.getParameter("email");

response.setContentType("text/html");

// Get the output stream to write the response

PrintWriter out = response.getWriter();

// Display the submitted data

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

out.println("<h1>Form Submission Result</h1>");

out.println("<p>Name: " + name + "</p>");

out.println("<p>Email: " + email + "</p>");

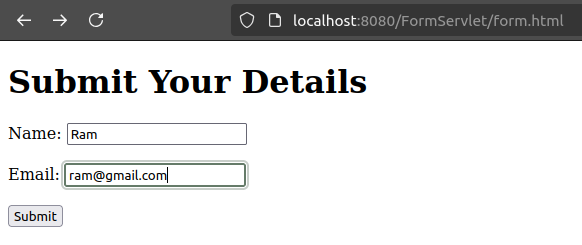
out.println("</body></html>");

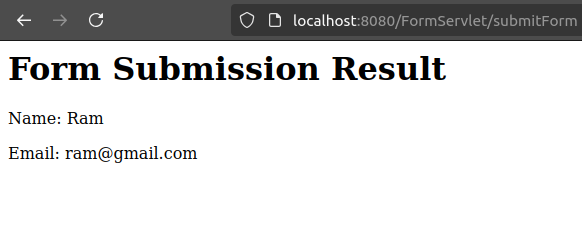
}

}

web.xml

<servlet>  
 <servlet-name>FormServlet</servlet-name>  
 <servlet-class>com.example.formservlet.FormServlet</servlet-class>  
</servlet>  
<servlet-mapping>  
 <servlet-name>FormServlet</servlet-name>  
 <url-pattern>/submitForm</url-pattern>  
</servlet-mapping>

And this gives the output as:  




1. Write a servlet that distinguishes between GET and POST requests and processes them accordingly.

The following Java servlet demonstrates how to handle both GET and POST requests. It overrides the doGet and doPost methods from the HttpServlet class to process requests differently based on the HTTP method used. This approach is useful for distinguishing between types of requests in a web application, such as fetching data versus submitting data.  
  
First, we define the RequestHandlerServlet as:

package com.example.requesthandlerservlet;

import jakarta.servlet.ServletException;

import jakarta.servlet.annotation.WebServlet;

import jakarta.servlet.http.HttpServlet;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

import java.io.IOException;

import java.io.PrintWriter;

@WebServlet("/RequestHandlerServlet")

public class RequestHandlerServlet extends HttpServlet {

@Override

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

response.setContentType("text/html");

PrintWriter out = response.getWriter();

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

out.println("<h1>GET Request Received</h1>");

out.println("<p>This response is generated for a GET request.</p>");

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

}

@Override

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

response.setContentType("text/html");

PrintWriter out = response.getWriter();

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

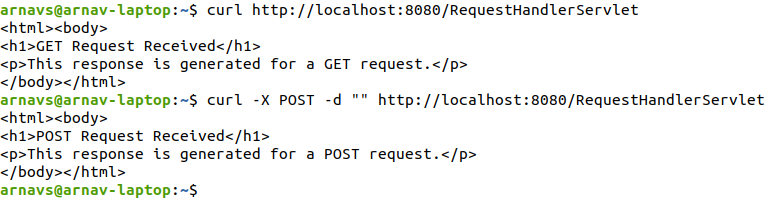
out.println("<h1>POST Request Received</h1>");

out.println("<p>This response is generated for a POST request.</p>");

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

}

}

And then use the default generated web.xml. And then, we check the output as:  


1. Implement a servlet to fetch and display user details from a database.

In this example, we will create a Java servlet that connects to a PostgreSQL database to fetch and display user details (such as student ID, name, and grade) from a previously defined Students table. The servlet will retrieve data from the database and display it in an HTML table on the browser. The connection to the database is made using JDBC, and the servlet will execute a SQL query to fetch student details.  
  
First, we define the StudentDetailsServlet class as:

package com.example.studentdetails;

import jakarta.servlet.\*;

import jakarta.servlet.http.\*;

import java.io.IOException;

import java.io.PrintWriter;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

public class StudentDetailsServlet extends HttpServlet {

// Database credentials

private static final String URL = "jdbc:postgresql://localhost:5432/java\_lab";

private static final String USER = "postgres";

private static final String PASSWORD = "password";

@Override

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

// Set the content type to HTML

response.setContentType("text/html");

PrintWriter out = response.getWriter();

try {

Class.forName("org.postgresql.Driver");

} catch (ClassNotFoundException e) {

throw new RuntimeException(e);

}

// Establish database connection

try (Connection connection = DriverManager.getConnection(URL, USER, PASSWORD)) {

Statement stmt = connection.createStatement();

ResultSet rs = stmt.executeQuery("SELECT \* FROM Students");

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

out.println("<h1>Student Details</h1>");

out.println("<table border='1'><tr><th>ID</th><th>Name</th><th>Grade</th></tr>");

// Loop through the ResultSet and display each student's details

while (rs.next()) {

out.println("<tr>");

out.println("<td>" + rs.getInt("id") + "</td>");

out.println("<td>" + rs.getString("name") + "</td>");

out.println("<td>" + rs.getString("grade") + "</td>");

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

}

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

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

} catch (Exception e) {

e.printStackTrace();

out.println("<p>Error fetching student details: " + e.getMessage() + "</p>");

}

}

}

And then web.xml as:  
<servlet>

<servlet-name>StudentDetailsServlet</servlet-name>

<servlet-class>com.example.studentdetails.StudentDetailsServlet</servlet-class>

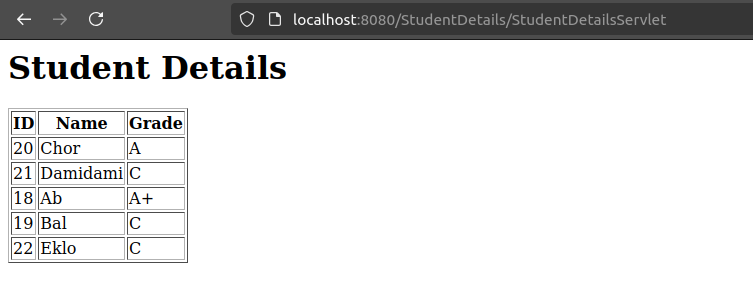
</servlet>

<servlet-mapping>

<servlet-name>StudentDetailsServlet</servlet-name>

<url-pattern>/StudentDetailsServlet</url-pattern>

</servlet-mapping>

And, we can view on the browser as:  


1. Create a servlet that uses cookies to store user preferences and a session to manage login information.

This servlet demonstrates the use of cookies to store user preferences and sessions to manage user login information. The implementation allows users to log in, set preferences, and view their preferences persistently using cookies.  
  
We define UserPreferencesServlet at first as:

package com.example.userpreferences;

import jakarta.servlet.*;*

*import jakarta.servlet.http.*;

import java.io.IOException;

import java.io.PrintWriter;

public class UserPreferencesServlet extends HttpServlet {

@Override  
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 // Get the session object to check if the user is logged in  
 HttpSession session = request.getSession(false);  
  
 // Set the content type to HTML  
 response.setContentType("text/html");  
 PrintWriter out = response.getWriter();  
  
 // Check if the user is logged in, display appropriate content  
 if (session != null && session.getAttribute("username") != null) {  
 String username = (String) session.getAttribute("username");  
 // Retrieve user preferences (theme) from cookies if available  
 Cookie[] cookies = request.getCookies();  
 String theme = "default";  
 if (cookies != null) {  
 for (Cookie cookie : cookies) {  
 if (cookie.getName().equals("theme")) {  
 theme = cookie.getValue();  
 break;  
 }  
 }  
 }  
  
 // Display the user preferences and a logout option  
 out.println("<html><body>");  
 out.println("<h1>Welcome, " + username + "!</h1>");  
 out.println("<p>Your preferred theme is: " + theme + "</p>");  
 out.println("<form action='' method='POST'>");  
 out.println("<label for='theme'>Choose Theme: </label>");  
 out.println("<select name='theme' id='theme'>");  
 out.println("<option value='light'>Light</option>");  
 out.println("<option value='dark'>Dark</option>");  
 out.println("</select><br>");  
 out.println("<input type='submit' value='Save Preferences'/>");  
 out.println("</form>");  
 out.println("<a href='logout'>Logout</a>");  
 out.println("</body></html>");  
 } else {  
 // If user is not logged in, show the login form  
 out.println("<html><body>");  
 out.println("<h1>Please login</h1>");  
 out.println("<form action='' method='POST'>");  
 out.println("<label for='username'>Username: </label><input type='text' name='username' required/><br>");  
 out.println("<label for='password'>Password: </label><input type='password' name='password' required/><br>");  
 out.println("<input type='submit' value='Login'/>");  
 out.println("</form>");  
 out.println("</body></html>");  
 }  
}  
  
@Override  
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 // Check if it's a login request or a preference update  
 String username = request.getParameter("username");  
 String password = request.getParameter("password");  
  
 // Handle login functionality  
 if (username != null && password != null && !username.isEmpty() && !password.isEmpty()) {  
 // Validate username and password (simple check for example purposes)  
 if (username.equals("admin") && password.equals("password")) {  
 // Create a session to store login info  
 HttpSession session = request.getSession();  
 session.setAttribute("username", username);  
  
 // Redirect to the main page  
 response.sendRedirect("UserPreferencesServlet");  
 return;  
 } else {  
 // Invalid credentials  
 response.getWriter().println("Invalid username or password.");  
 }  
 }  
  
 // Handle saving user preferences  
 String theme = request.getParameter("theme");  
 if (theme != null) {  
 // Store the selected theme in a cookie for future visits  
 Cookie themeCookie = new Cookie("theme", theme);  
 themeCookie.setMaxAge(60 \* 60 \* 24 \* 365); // Expiry time: 1 year  
 response.addCookie(themeCookie);  
  
 // Redirect to the preferences page  
 response.sendRedirect("UserPreferencesServlet");  
 }  
}  
}

And then define the web.xml as:  
 <servlet>

<servlet-name>UserPreferencesServlet</servlet-name>

<servlet-class>com.example.userpreferences.UserPreferencesServlet</servlet-class>

</servlet>

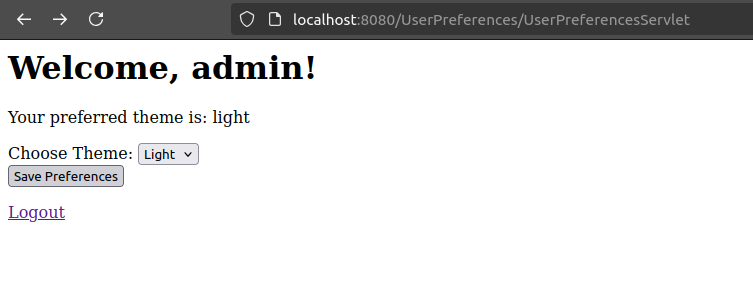
<servlet-mapping>

<servlet-name>UserPreferencesServlet</servlet-name>

<url-pattern>/UserPreferencesServlet</url-pattern>

</servlet-mapping>

And, we can view on the browser as:

And on every refresh, the preference persists.

1. Create a JSP page to display the current date and time using JSP declarations and expressions.

This JSP page demonstrates how to use JSP declarations to declare variables (in this case, the current date and time) and JSP expressions to display the values in the HTML response. The page will dynamically show the current date and time each time it's accessed.  
  
For this, we first define an index.jsp file as:

<%--

Created by IntelliJ IDEA.

User: arnavs

--%>

<%@ page contentType="text/html;charset=UTF-8" language="java" %>

<%@ page import="java.util.Date" %>

<html>

<head>

<title>Current Date and Time</title>

</head>

<body>

<h1>Current Date and Time</h1>

<!-- Using JSP Declaration to declare a variable -->

<%

Date currentDate = new Date();

%>

<p>The current date and time is: <%= currentDate.toString() %></p>

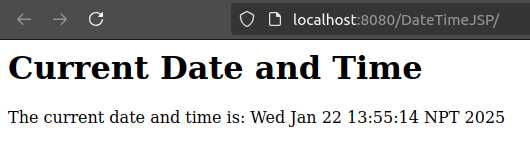
</body>

</html>

And then, define a web.xml as:  
<welcome-file-list>

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

</welcome-file-list>

And we can view on browser as:  


1. Write a JSP page that demonstrates the use of implicit objects like request, response, and session.

This JSP page demonstrates how to use implicit objects (request, response, and session) to handle user input, store session data, and display dynamic content. The session object allows us to persist the user's name across different requests, and the request and response objects provide access to client input and server output.  
  
First, we define index.jsp as:

<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>

<%@ page import="java.util.\*" %>

<html>

<head>

<title>JSP Implicit Objects Example</title>

</head>

<body>

<h1>JSP Implicit Objects Example</h1>

<form method="post">

Name: <input type="text" name="username" />

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

</form>

<%

// Check if the user has already submitted a name

String name = request.getParameter("username");

if (name != null && !name.isEmpty()) {

// Store the username in the session

session.setAttribute("username", name);

response.setContentType("text/html");

// Output a message using the implicit response object

out.println("<h3>Hello, " + name + "!</h3>");

out.println("<p>Your name has been stored in the session.</p>");

} else {

// If the name is not provided, retrieve it from the session

String storedName = (String) session.getAttribute("username");

if (storedName != null) {

out.println("<h3>Welcome back, " + storedName + "!</h3>");

} else {

out.println("<h3>Welcome! Please enter your name.</h3>");

}

}

%>

</body>

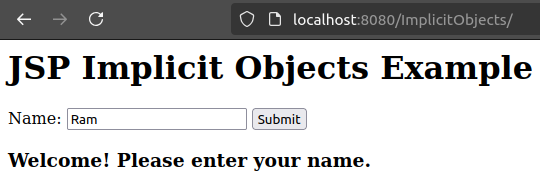
</html>

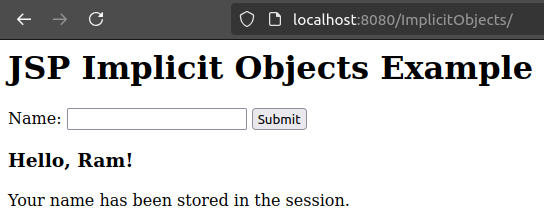
And then define web.xml as:  
 <welcome-file-list>

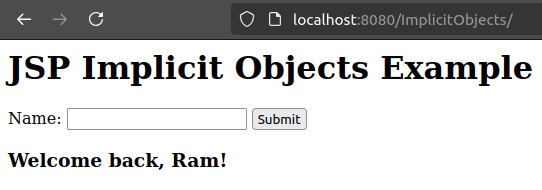
<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

And on browser, we can see the following:







1. Implement a JSP page to process a login form and display a welcome message for the user.

This JSP page demonstrates how to process a simple login form where the user can submit their username and password. The page will validate the credentials, and if the login is successful, it will display a personalized welcome message. For the sake of simplicity, we assume that the correct username is "admin" and the password is "password".

We have the index.jsp file as:

<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>

<%@ page import="java.util.\*" %>

<html>

<head>

<title>Login Page</title>

</head>

<body>

<h1>Login</h1>

<!-- Displaying a login form -->

<form method="post">

Username: <input type="text" name="username" required /><br><br>

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

<input type="submit" value="Login" />

</form>

<%

// Retrieve form parameters

String username = request.getParameter("username");

String password = request.getParameter("password");

// Check if the form has been submitted

if (username != null && password != null) {

// Validate username and password (Example with hardcoded values)

if ("admin".equals(username) && "password".equals(password)) {

// If valid, display the welcome message

session.setAttribute("username", username);

out.println("<h3>Welcome, " + username + "!</h3>");

} else {

// Invalid credentials

out.println("<p style='color: red;'>Invalid username or password. Please try again.</p>");

}

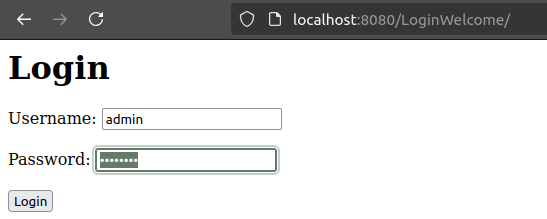
}

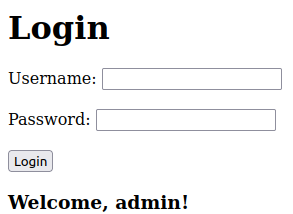
%>

</body>

</html>

And on the browser, we can view:





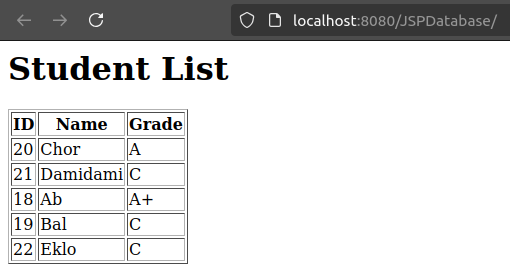
1. Create a JSP page that fetches data from a database and displays it in an HTML table.

This JSP page demonstrates how to fetch data from a PostgreSQL database and display it in an HTML table. In this example, we will query a students table that contains id, name, and grade columns and display the results in a table format.

We have the index.jsp file as:

<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>  
<%@ page import="java.sql.\*, jakarta.servlet.\*, jakarta.servlet.http.\*" %>  
<html>  
<head>  
 <title>Student List</title>  
</head>  
<body>  
<h1>Student List</h1>  
  
<table border="1">  
 <tr>  
 <th>ID</th>  
 <th>Name</th>  
 <th>Grade</th>  
 </tr>  
  
 <%  
 // Database connection details  
 String url = "jdbc:postgresql://localhost:5432/java\_lab"; // Replace with your actual database name  
 String user = "postgres";  
 String password = "password";  
 Connection conn = null;  
 Statement stmt = null;  
 ResultSet rs = null;  
  
 try {  
 // Establish connection to the database  
 Class.forName("org.postgresql.Driver");  
 conn = DriverManager.getConnection(url, user, password);  
  
 // Create a statement object to execute the SQL query  
 stmt = conn.createStatement();  
  
 // SQL query to fetch student data  
 String sql = "SELECT id, name, grade FROM students";  
 rs = stmt.executeQuery(sql);  
  
 // Loop through the result set and display data in table rows  
 while (rs.next()) {  
 int id = rs.getInt("id");  
 String name = rs.getString("name");  
 String grade = rs.getString("grade");  
  
 out.println("<tr>");  
 out.println("<td>" + id + "</td>");  
 out.println("<td>" + name + "</td>");  
 out.println("<td>" + grade + "</td>");  
 out.println("</tr>");  
 }  
 } catch (Exception e) {  
 out.println("<tr><td colspan='3'>Error: " + e.getMessage() + "</td></tr>");  
 } finally {  
 // Close the resources  
 try {  
 if (rs != null) rs.close();  
 if (stmt != null) stmt.close();  
 if (conn != null) conn.close();  
 } catch (SQLException se) {  
 se.printStackTrace();  
 }  
 }  
 %>  
</table>  
</body>  
</html>

And on the browser, we can see:



1. Create an RMI application where the client sends a number, and the server responds with its factorial.

This example demonstrates how to implement a Remote Method Invocation (RMI) application where the client sends a number to the server, and the server responds with its factorial. RMI allows a client to invoke methods on an object located on a remote server, which makes it a useful technology for distributed applications.

FactorialService.java

import java.rmi.Remote;

import java.rmi.RemoteException;

public interface FactorialService extends Remote {

long calculateFactorial(int number) throws RemoteException;

}

FactorialServer.java

import java.rmi.\*;

import java.rmi.server.\*;

public class FactorialServer extends UnicastRemoteObject implements FactorialService {

public FactorialServer() throws RemoteException {

super();

}

public long calculateFactorial(int number) throws RemoteException {

long result = 1;

for (int i = 1; i <= number; i++) {

result \*= i;

}

return result;

}

public static void main(String[] args) {

try {

FactorialServer server = new FactorialServer();

// Bind the server object to the RMI registry

Naming.rebind("rmi://localhost/FactorialService", server);

System.out.println("FactorialServer is ready.");

} catch (Exception e) {

System.out.println("Error: " + e.getMessage());

e.printStackTrace();

}

}

}

FactorialClient.java

import java.rmi.\*;

public class FactorialClient {

public static void main(String[] args) {

try {

// Lookup the RMI registry for the remote service

FactorialService service = (FactorialService) Naming.lookup("rmi://localhost/FactorialService");

// Get the number from the user

int number = 5;

// Call the remote method to calculate factorial

long factorial = service.calculateFactorial(number);

// Display the result

System.out.println("Factorial of " + number + " is " + factorial);

} catch (Exception e) {

System.out.println("Error: " + e.getMessage());

e.printStackTrace();

}

}

}

And, we can view the output by first compiling the application, then starting the RMIRegistry, and finally running the application as:  
