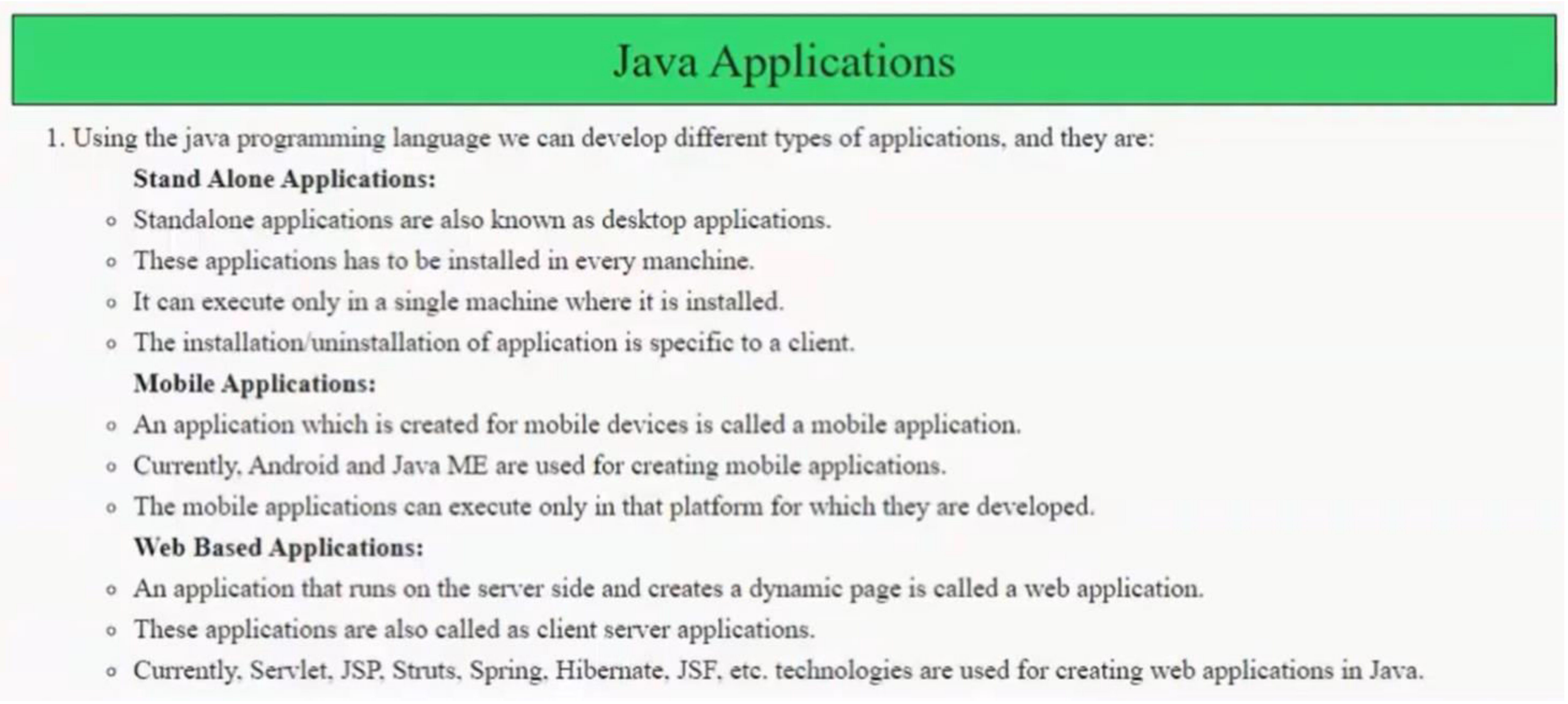
Servlets & JSP

Dr K Chaitanya
Assistant Professor
Department of CSE
ANU College of Engineering &Technology
Acharya Nagarjuna University



Java Applications

1. Using the java programming language we can develop different types of applications, and they are:

Stand Alone Applications:

- Standalone applications are also known as desktop applications.
- These applications has to be installed in every manchine.
- It can execute only in a single machine where it is installed.
- The installation/uninstallation of application is specific to a client.

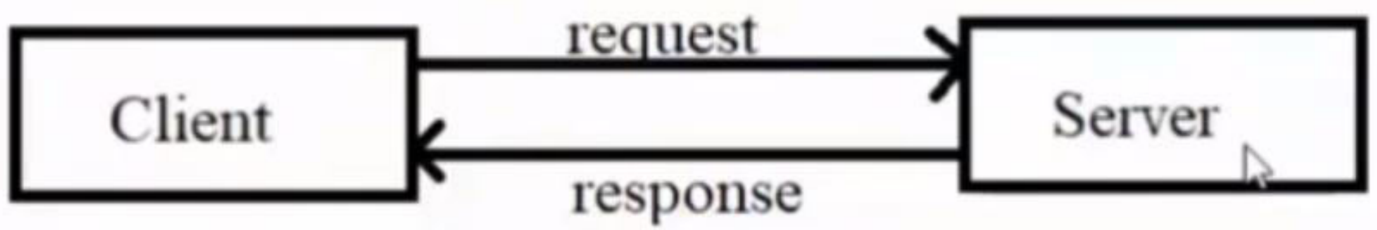
Mobile Applications:

- An application which is created for mobile devices is called a mobile application.
- Currently, Android and Java ME are used for creating mobile applications.
- The mobile applications can execute only in that platform for which they are developed.

Web Based Applications:

- An application that runs on the server side and creates a dynamic page is called a web application.
- These applications are also called as client server applications.
- Currently, Servlet, JSP, Struts, Spring, Hibernate, JSF, etc. technologies are used for creating web applications in Java.

Core Java: J2SE
Advanced Java:J2EE,J2ME
J2EE: For Developing web based applications
J2ME: For Developing mobile applications



2. Every web based application require 2 types of softwares, and they are: 1) Client Software, 2) Server Software.
3. **Client Software:** It can be any browser that we use in our machine like internet explorer, chrome, firefox etc. Every operating system will provide one default browser.
4. **Server Software:** These servers are available in the market like tomcat, weblogic, glassfish, websphere etc.
5. The user will send the request to the server by using the client software, the server software will receive the request and process the request and then send the response to the client.
6. A web based application consists of web resources (html, images, css, js files etc) and web components (servlets and jsp).
7. To make the communication between client and server in a application we use a protocol called **HTTP**.
8. The web based application that we develop are classified into Static web application and Dynamic web application.
9. **Static:** The Application which won't be changed from person to person and time to time , such type of response is called as Static. Ex: Loginpage, Registrationpage etc. [html]
10. **Dynamic:** The Application which is varied from person to person and time to time, such type of response is called as Dynamic. Ex: Inbox, BankBalance etc. [servlet, jsp etc...]

Software's needed

- JDK ---- servlets in java
- Eclipse --- editor
- Apache tomcat -- server
- Oracle s/w -- database

Servlet

- **Servlet** technology is used to create a web application (resides at server side and generates a dynamic web page).
- Before Servlet, CGI (Common Gateway Interface) scripting language was common as a server-side programming language.
- Disadvantages of CGI

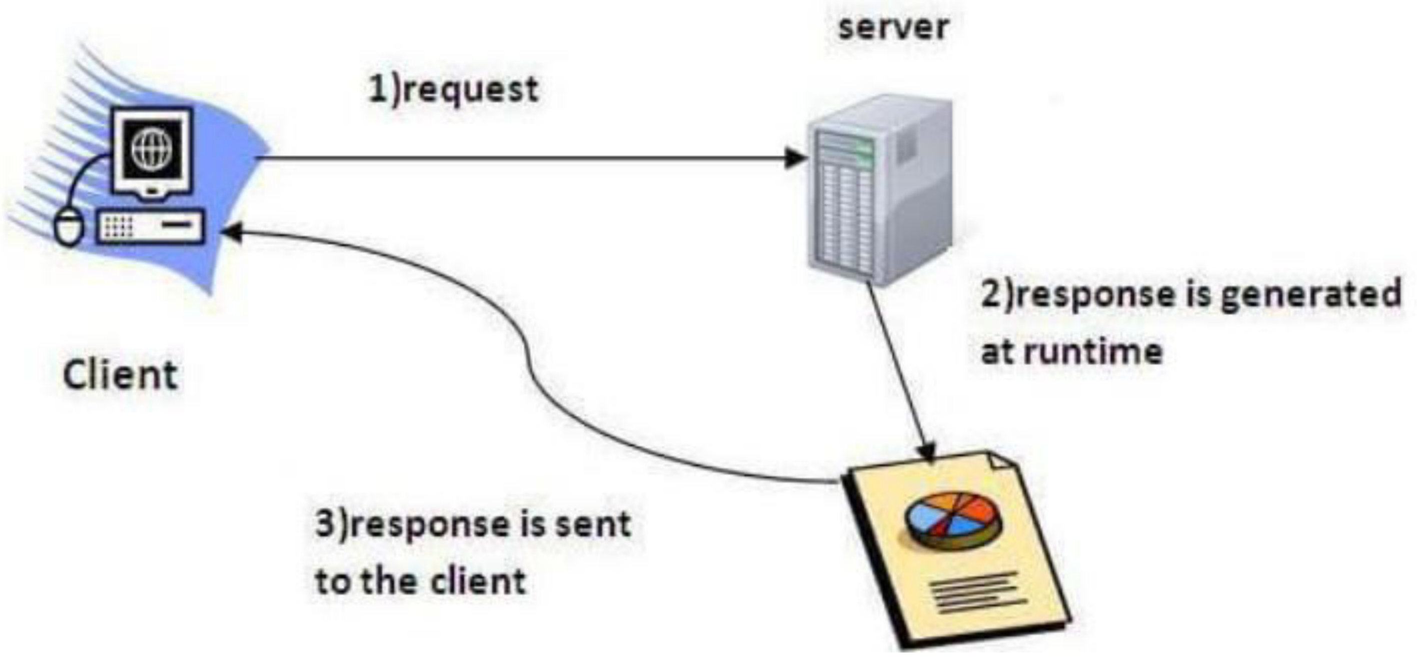
  There are many problems in CGI technology:

  1. If the number of clients increases, it takes more time for sending the response.
  2. For each request, it starts a process, and the web server is limited to start processes.
  3. It uses platform dependent language e.g. C, C++, perl.
- Advantages of Servlet

  Threads have many benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. The advantages of Servlet are as follows:

  1. **Better performance:** because it creates a thread for each request, not process.
  2. **Portability:** because it uses Java language.
  3. **Robust:** JVM manages Servlets, so we don't need to worry about the memory leak, garbage collection, etc.
  4. **Secure:** because it uses java language.
- A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Filter, etc. and other elements such as HTML, CSS, and JavaScript. The web components typically execute in Web Server and respond to the HTTP request.

- Servlet is a technology which is used to create a web application.
- Servlet is an API that provides many interfaces and classes including documentation.
- Servlet is an interface that must be implemented for creating any Servlet.
- Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.
- Servlet is a web component that is deployed on the server to create a dynamic web page.


server
1)request
Client
2)response is generated at runtime
3)response is sent to the client

Life cycle of a servlet

The servlet is in new state if servlet instance is created. After invoking the init() method, Servlet comes in the ready state. In the ready state, servlet performs all the tasks. When the web container invokes the destroy() method, it shifts to the end state.

1) Servlet class is loaded

The classloader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.

2) Servlet instance is created

The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.

3) init method is invoked

The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet. It is the life cycle method of the javax.servlet.Servlet interface.

Syntax of the init method: **public void** init(ServletConfig config) **throws** ServletException

4) service method is invoked

The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service method. If servlet is initialized, it calls the service method. Notice that servlet is initialized only once.

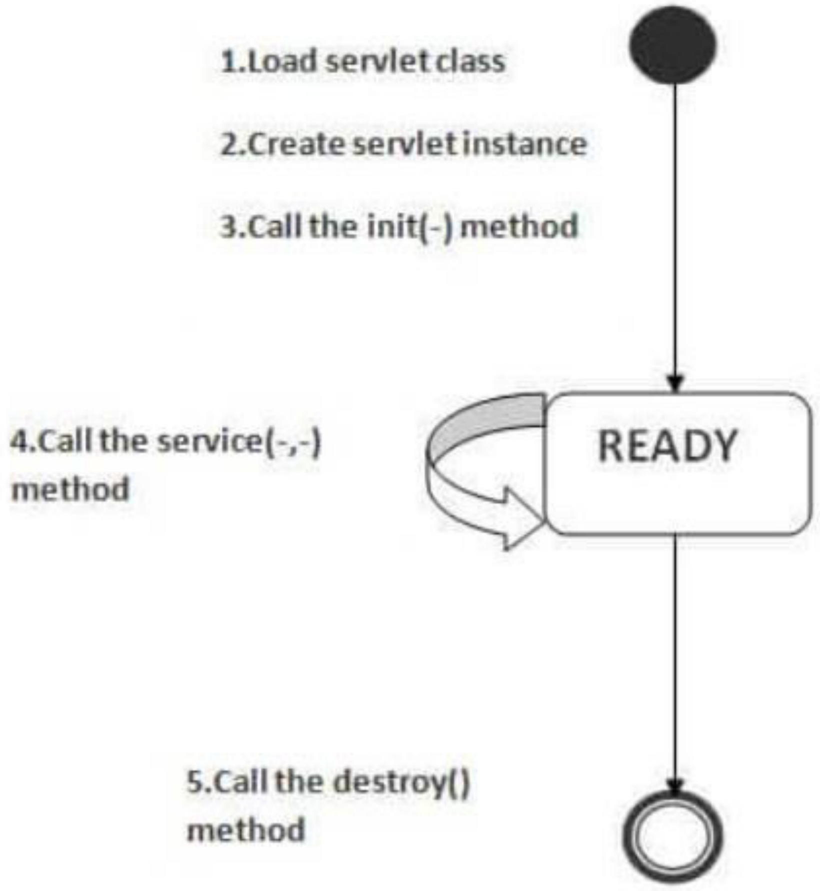
The syntax of the service method: **public void** service(ServletRequest request, ServletResponse response) **throws** ServletException, IOException

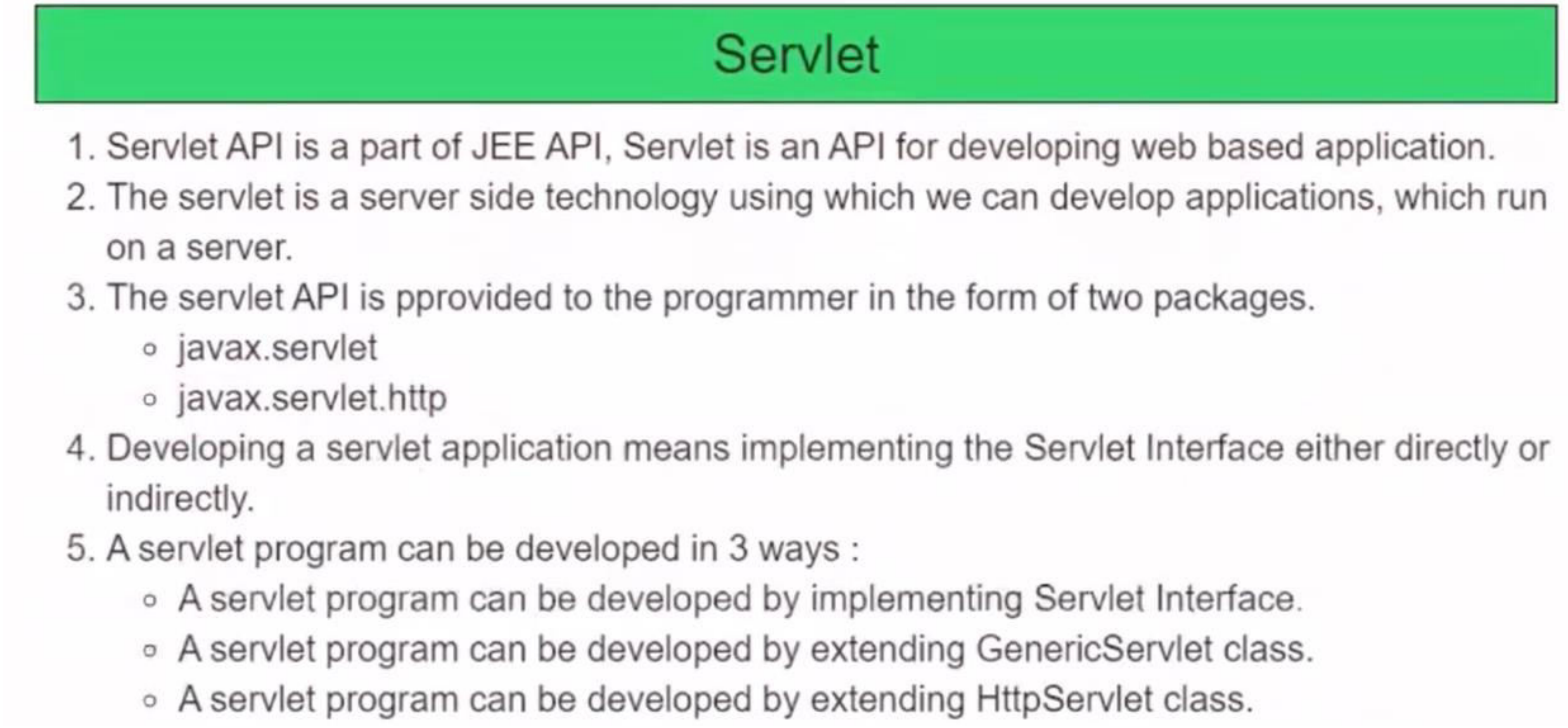
5) destroy method is invoked

The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc.

The syntax of the destroy method of the Servlet interface: **public void** destroy()

1. Servlet class is loaded.
2. Servlet instance is created.
3. init method is invoked.
4. service method is invoked.
5. destroy method is invoked.


1.Load servlet class
2.Create servlet instance
3.Call the init(-) method
READY
4.Call the service(-,-) method
5.Call the destroy() method

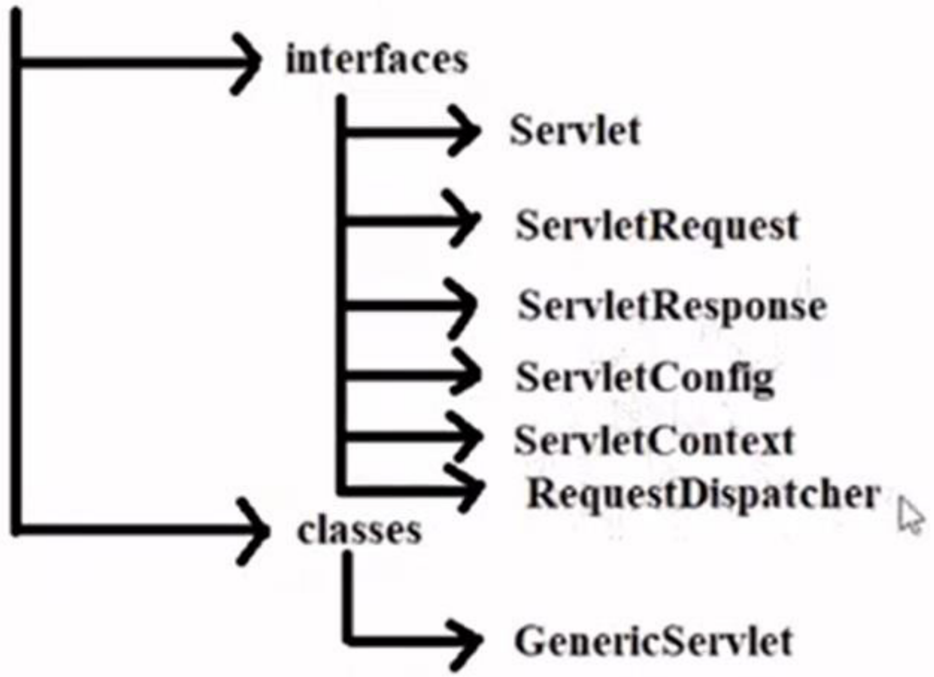


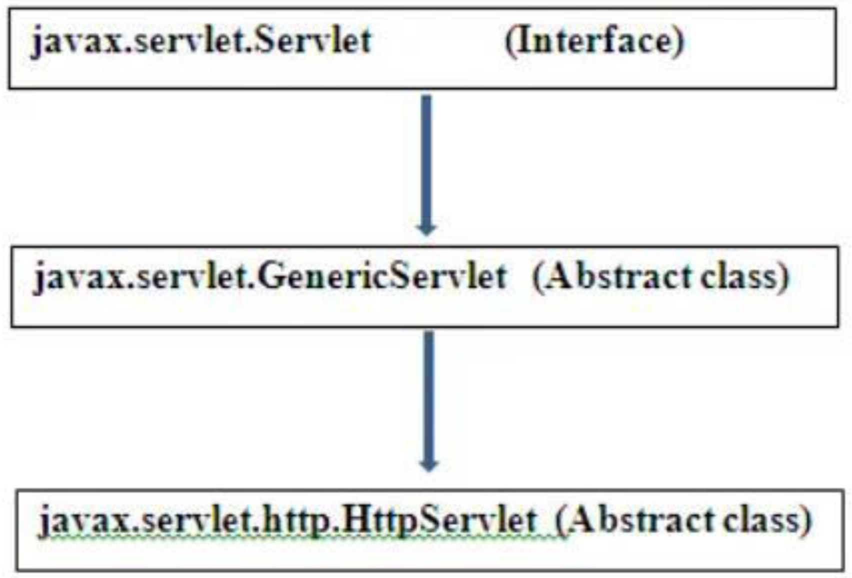
Servlet

1. Servlet API is a part of JEE API, Servlet is an API for developing web based application.
2. The servlet is a server side technology using which we can develop applications, which run on a server.
3. The servlet API is pprovided to the programmer in the form of two packages.
   - javax.servlet
   - javax.servlet.http
4. Developing a servlet application means implementing the Servlet Interface either directly or indirectly.
5. A servlet program can be developed in 3 ways :
   - A servlet program can be developed by implementing Servlet Interface.
   - A servlet program can be developed by extending GenericServlet class.
   - A servlet program can be developed by extending HttpServlet class.

If we implement Servlet Interface we need to implement all the 5 methods.
If we extends GenericServlet class we need to implement 1 method.
Generally for developing web based applications we extends HttpServlet class.

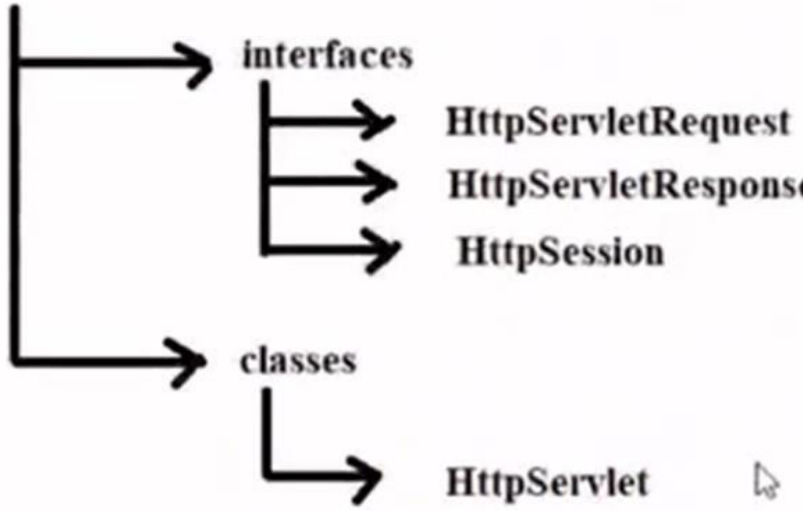
6. **javax.servlet package :** Used to develop protocol independent servlets [Generic].

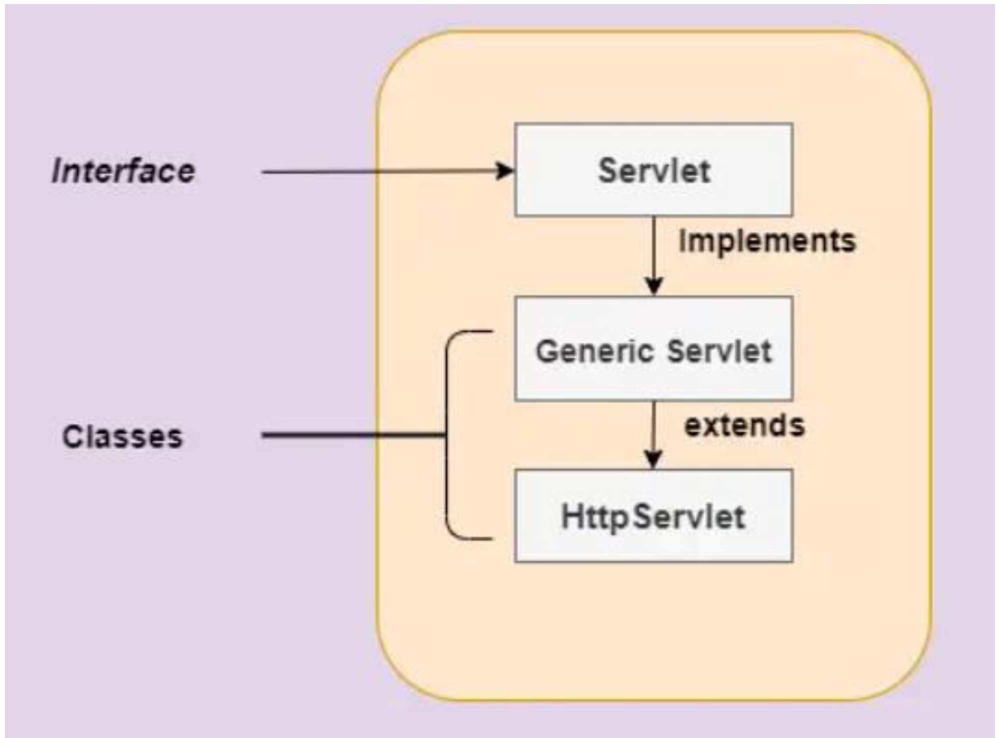




javax.servlet.Servlet (Interface)
javax.servlet.GenericServlet (Abstract class)
javax.servlet.http.HttpServlet (Abstract class)

7. **javax.servlet.http package :** Used to develop Http protocol based servlets.



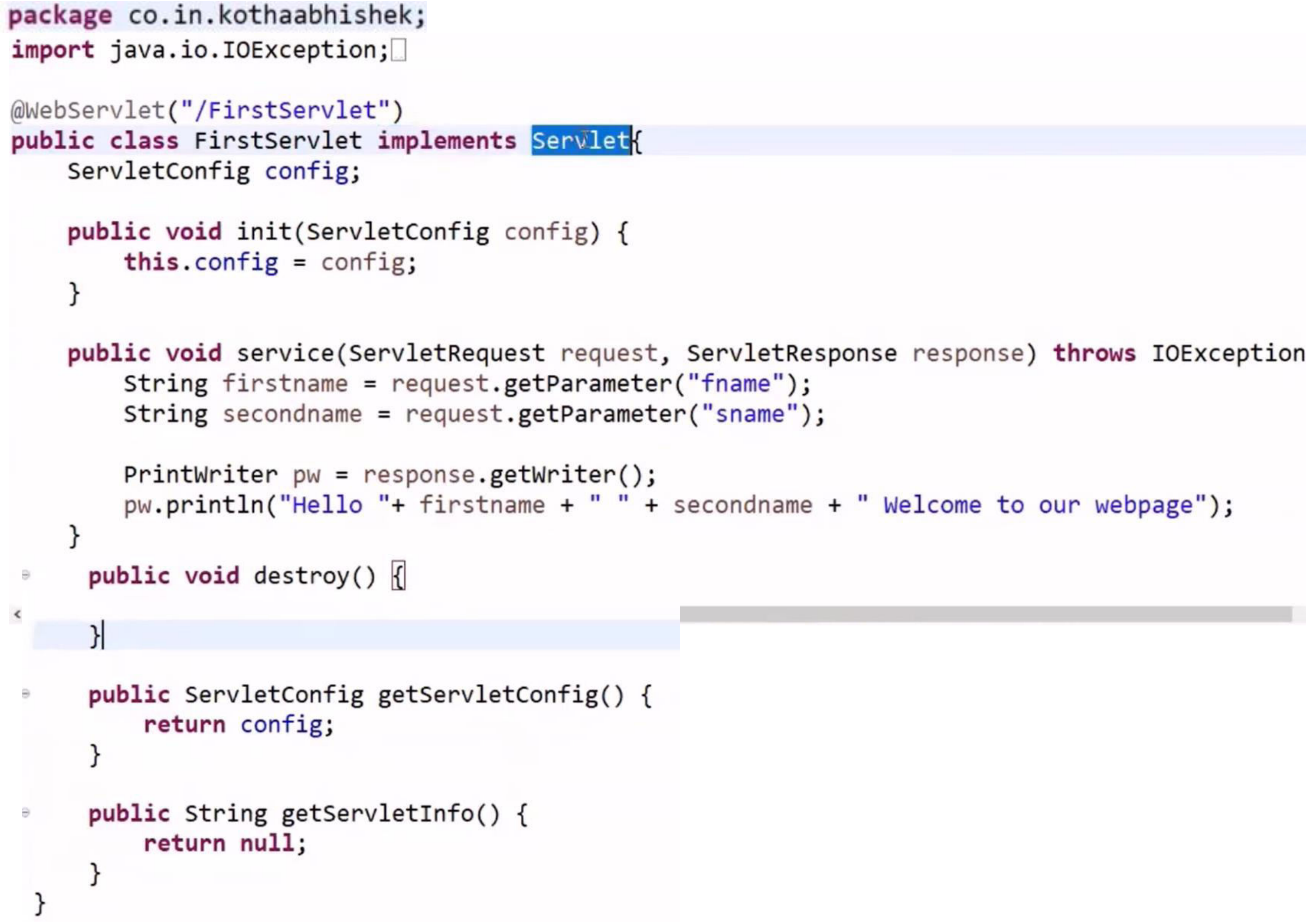


Servlet Interface

1. The Servlet interface belongs to javax.servlet package and it contains the following 5 methods which has to be implemented.

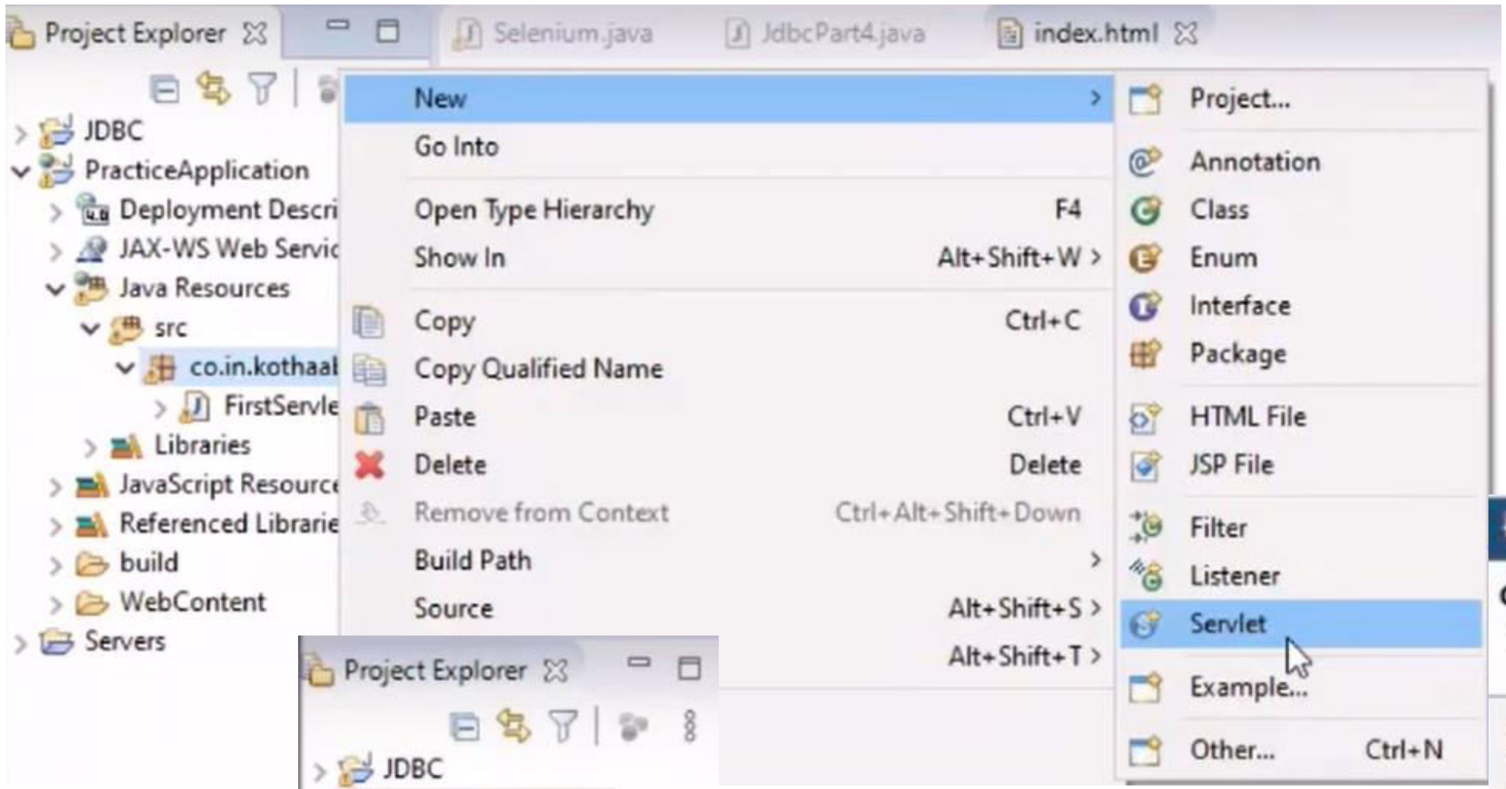
   Methods of Servlet interface :

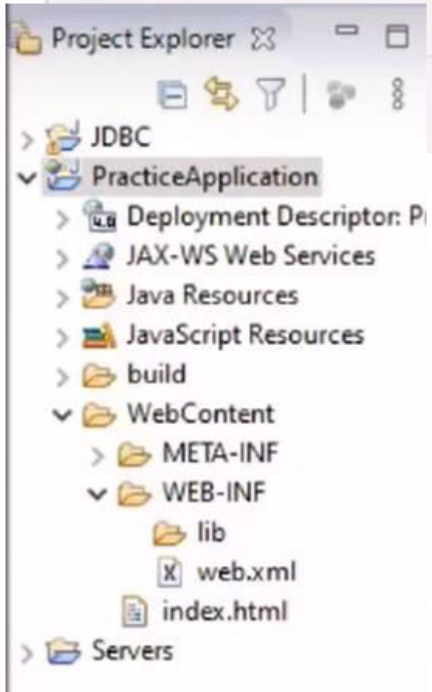
   - **public void init(ServletConfig)** : This method will be executed only one time, when the servlet object is created or when the first request is sent to the servlet.
   - **public void service(ServletRequest, ServletResponse)** : This method is the main method to perform the actual task i.e it will contain the business logic. The servlet container calls the service() method to handle requests coming from the client and to send the response back to the client. The service method checks the HTTP request type (GET, POST, PUT, DELETE, etc) and calls doGet, doPost, doPut, doDelete, etc which is appropriate.
   - **public void destroy()** : This method is executed only one time, when the servlet is destroyed.
   - **public ServletConfig getServletConfig()** : This method returns an object of ServletConfig.
   - **public String getServletInfo()** : This method returns the description about the object.

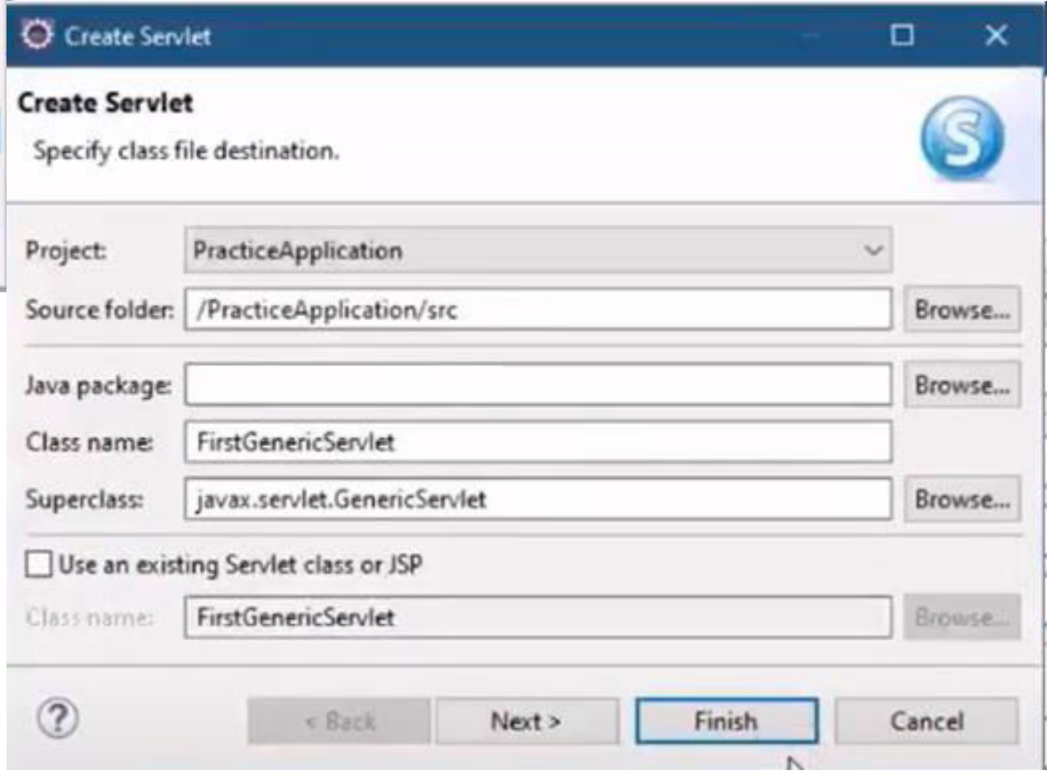


```
package co.in.kothaabhishek;
import java.io.IOException;

@WebServlet("/FirstServlet")
public class FirstServlet implements Servlet{
    ServletConfig config;

    public void init(ServletConfig config) {
        this.config = config;
    }

    public void service(ServletRequest request, ServletResponse response) throws IOException
        String firstname = request.getParameter("fname");
        String secondname = request.getParameter("sname");

        PrintWriter pw = response.getWriter();
        pw.println("Hello "+ firstname + " " + secondname + " Welcome to our webpage");
    }
    public void destroy() {

    }

    public ServletConfig getServletConfig() {
        return config;
    }

    public String getServletInfo() {
        return null;
    }
}
```

GenericServlet






Create Servlet

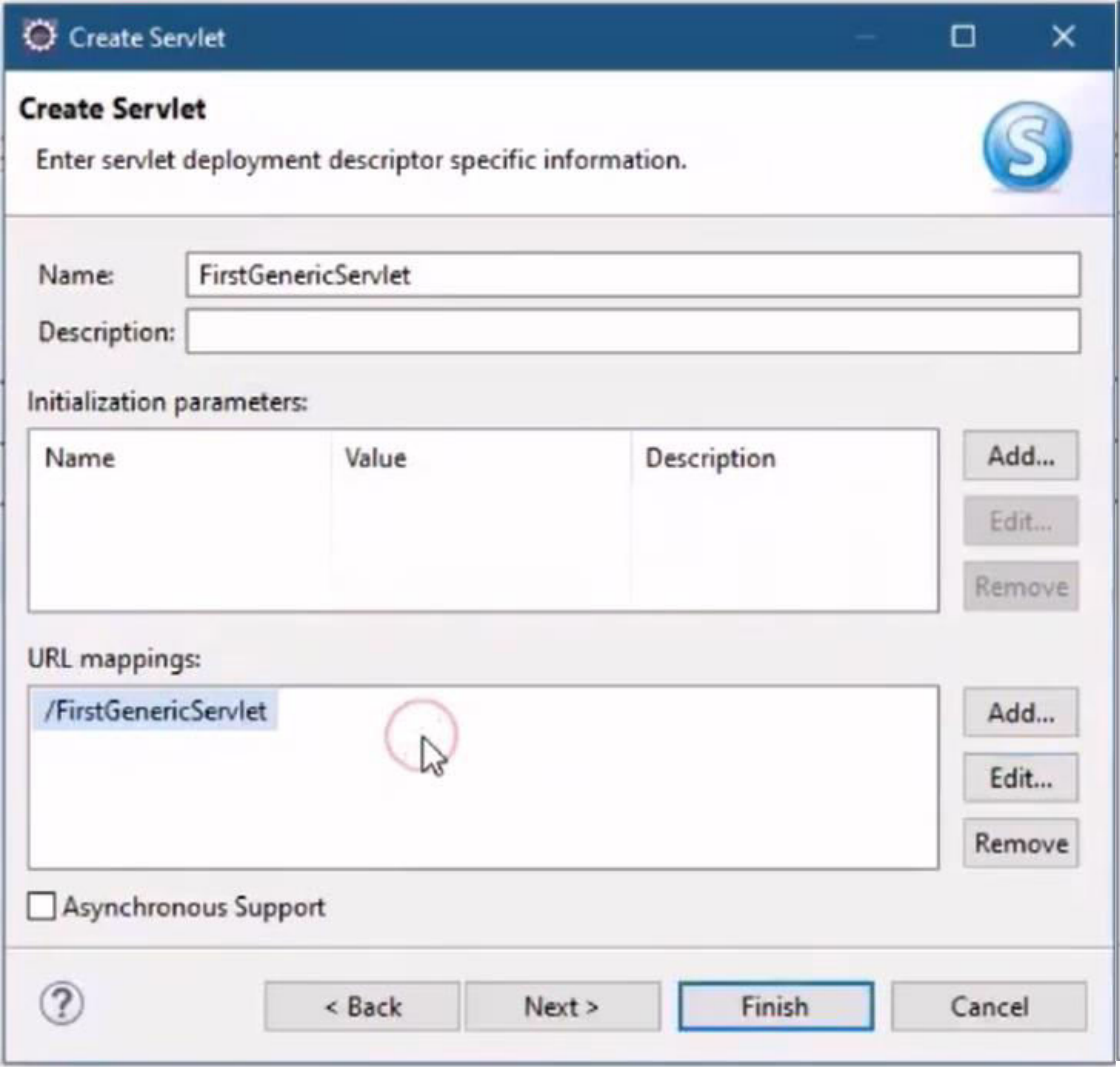
Create Servlet

Specify class file destination.

Project:	PracticeApplication
Source folder:	/PracticeApplication/src
Java package:	
Class name:	FirstGenericServlet
Superclass:	javax.servlet.GenericServlet

☐ Use an existing Servlet class or JSP

Class name: FirstGenericServlet


Create Servlet

Create Servlet

Enter servlet deployment descriptor specific information.

Name: FirstGenericServlet

Description:

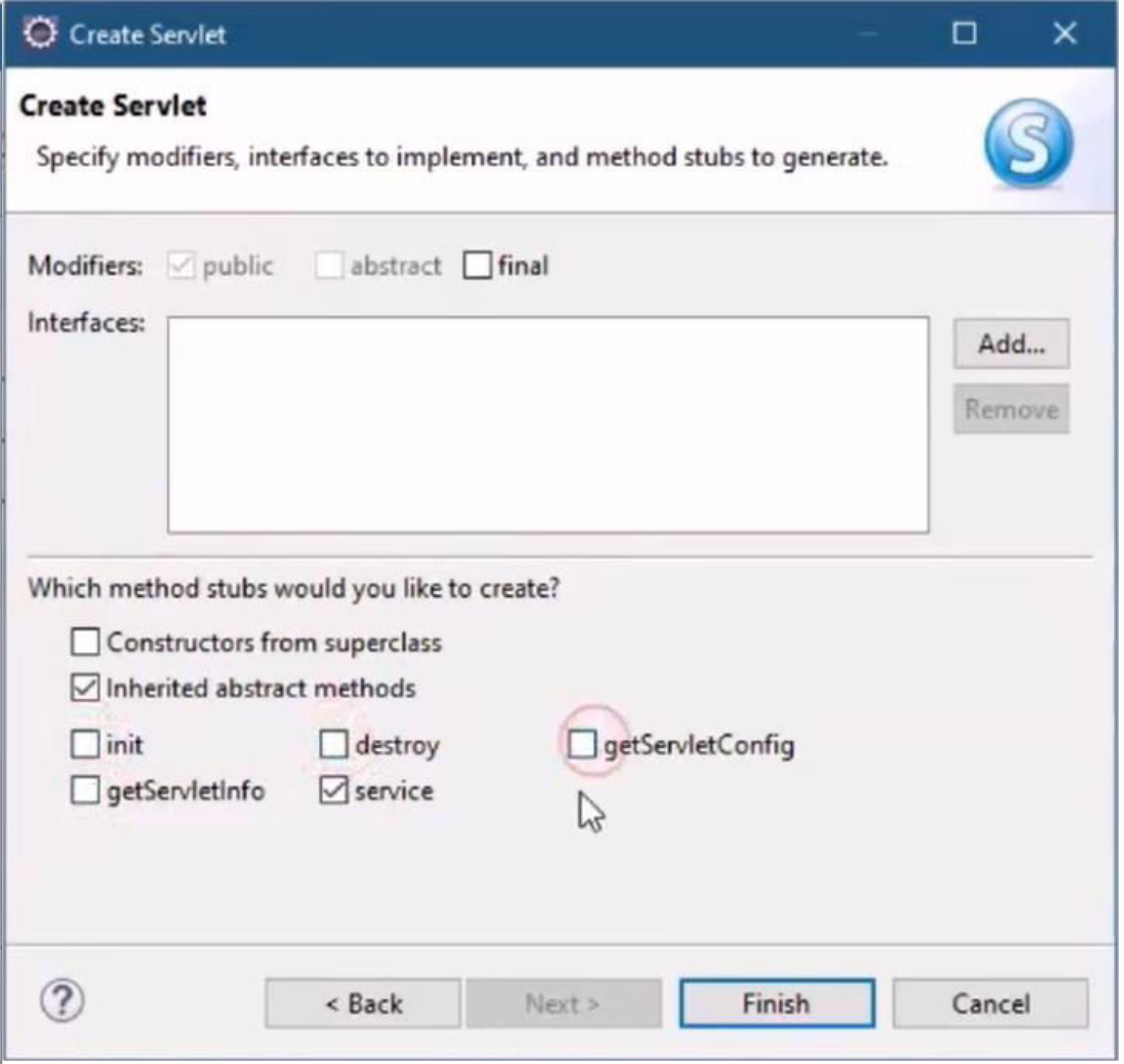
Initialization parameters:

Name	Value	Description

URL mappings:

/FirstGenericServlet

[ ] Asynchronous Support


Create Servlet

Create Servlet

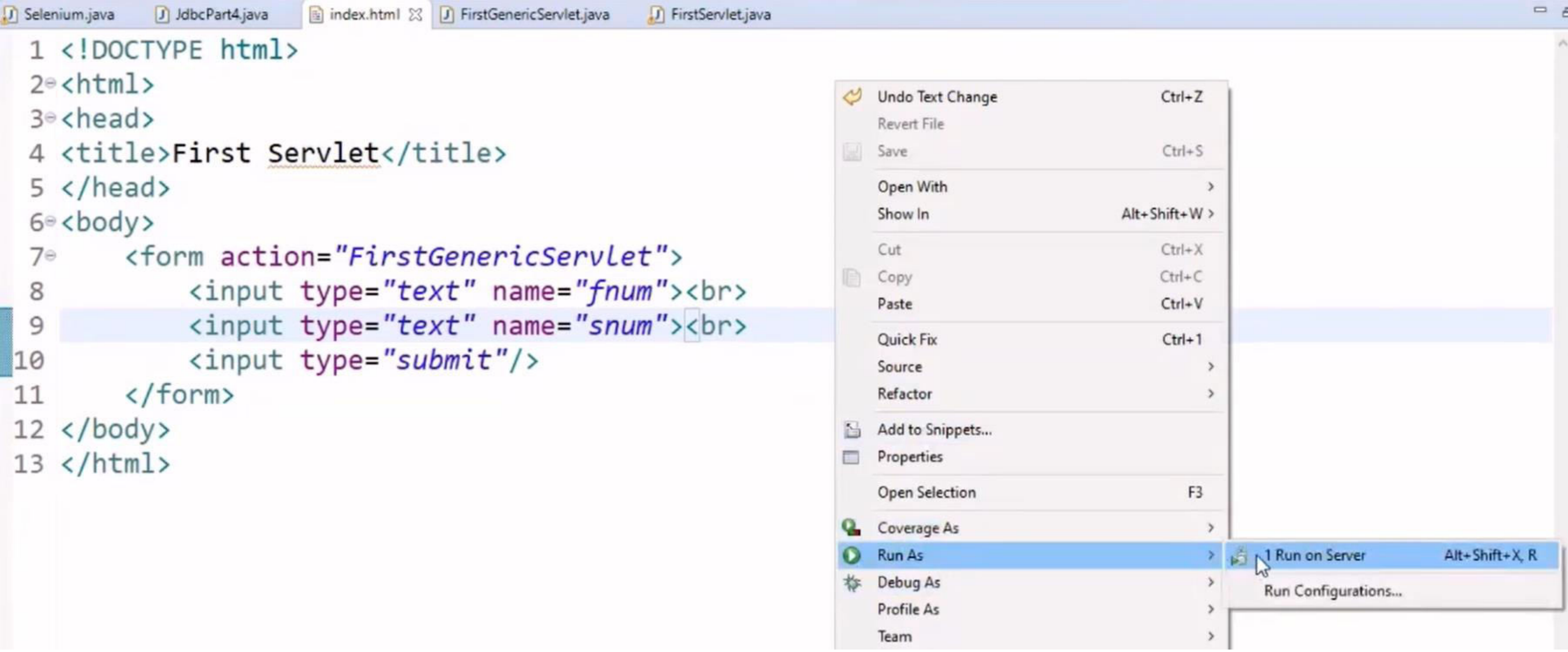
Specify modifiers, interfaces to implement, and method stubs to generate.

Modifiers: [✓ public] [ ] abstract [ ] final

Interfaces:

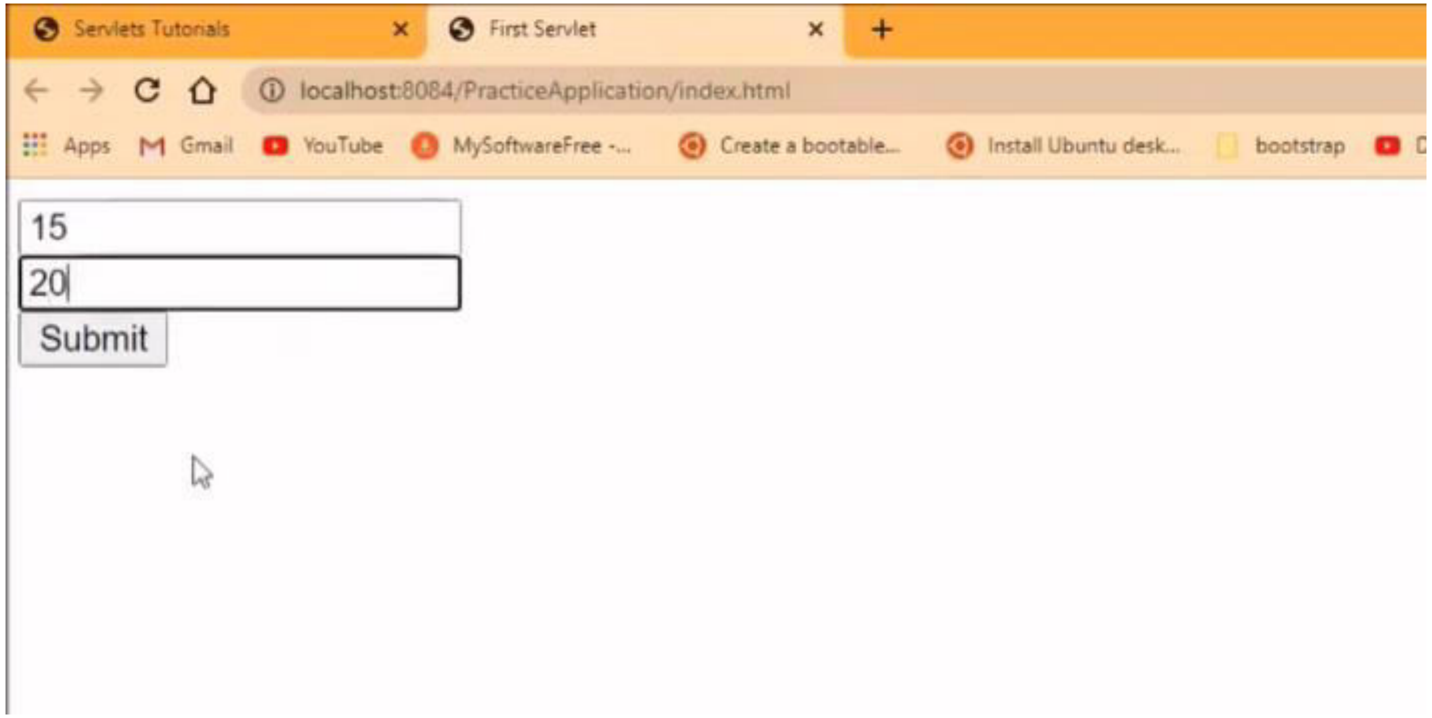
Which method stubs would you like to create?

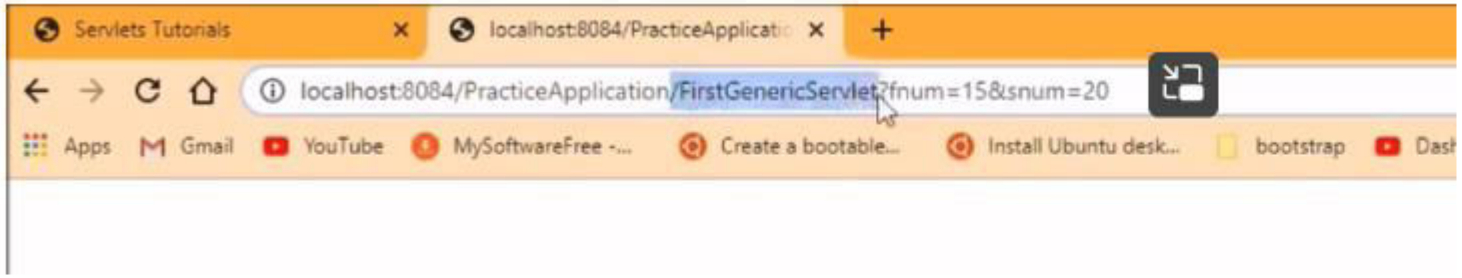
- [ ] Constructors from superclass
- [✓] Inherited abstract methods
- [ ] init
- [ ] destroy
- [ ] getServletConfig
- [ ] getServletInfo
- [✓] service

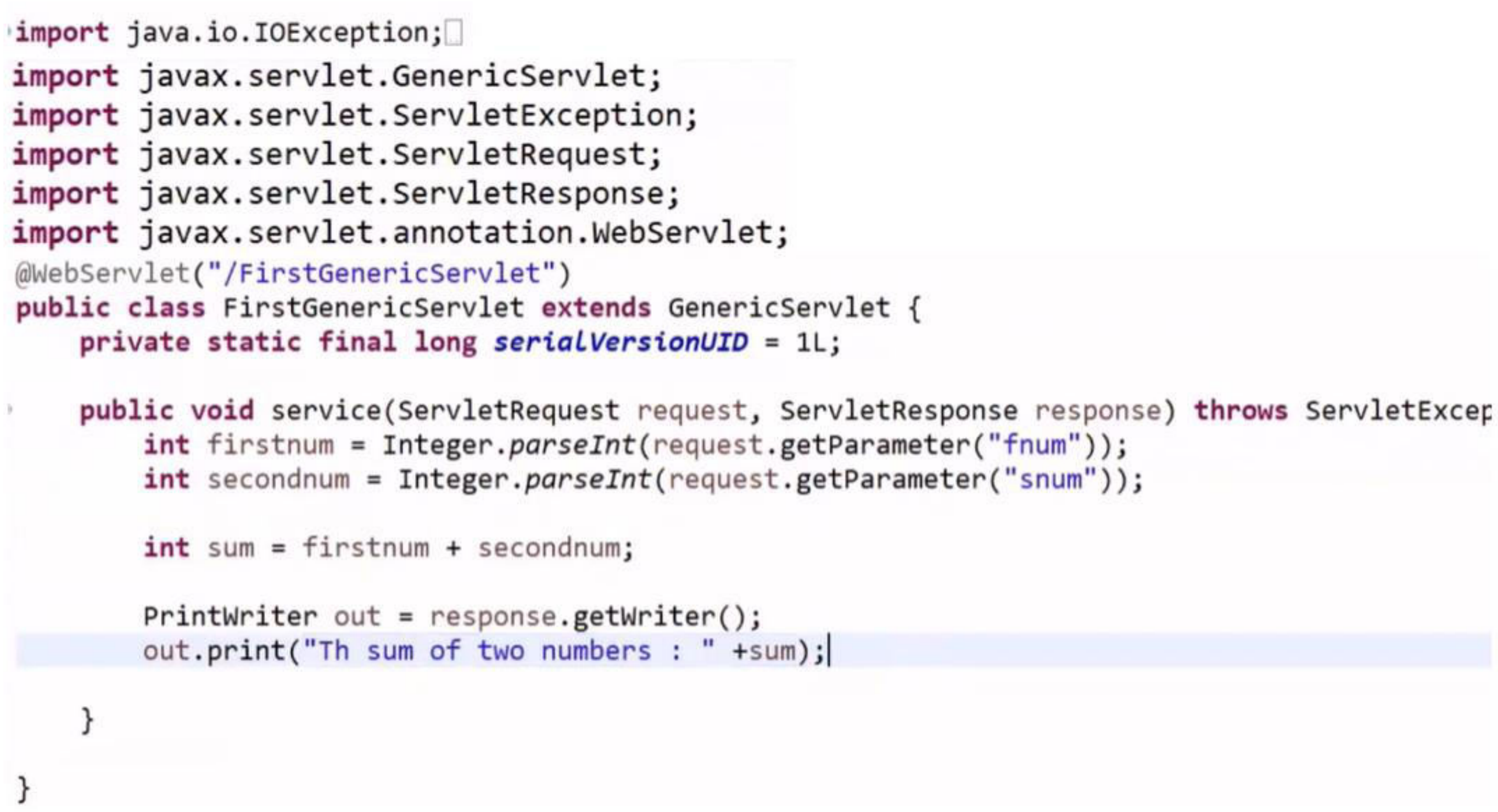


Selenium.java | JdbcPart4.java | index.html | FirstGenericServlet.java | FirstServlet.java

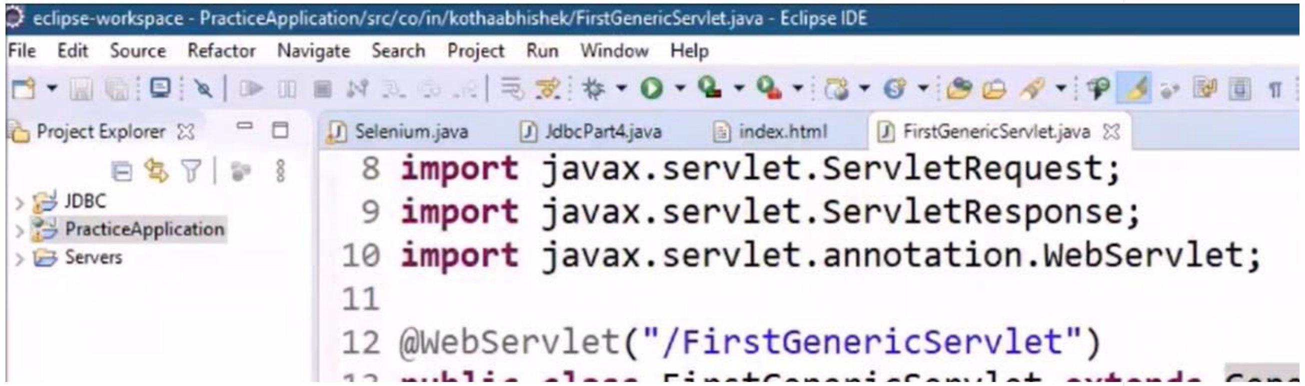
```html
<!DOCTYPE html>
<html>
<head>
<title>First Servlet</title>
</head>
<body>
    <form action="FirstGenericServlet">
        <input type="text" name="fnum"><br>
        <input type="text" name="snum"><br>
        <input type="submit"/>
    </form>
</body>
</html>
```


Servlets Tutorials
First Servlet
localhost:8084/PracticeApplication/index.html
Apps
Gmail
YouTube
MySoftwareFree -...
Create a bootable...
Install Ubuntu desk...
bootstrap
15
20
Submit

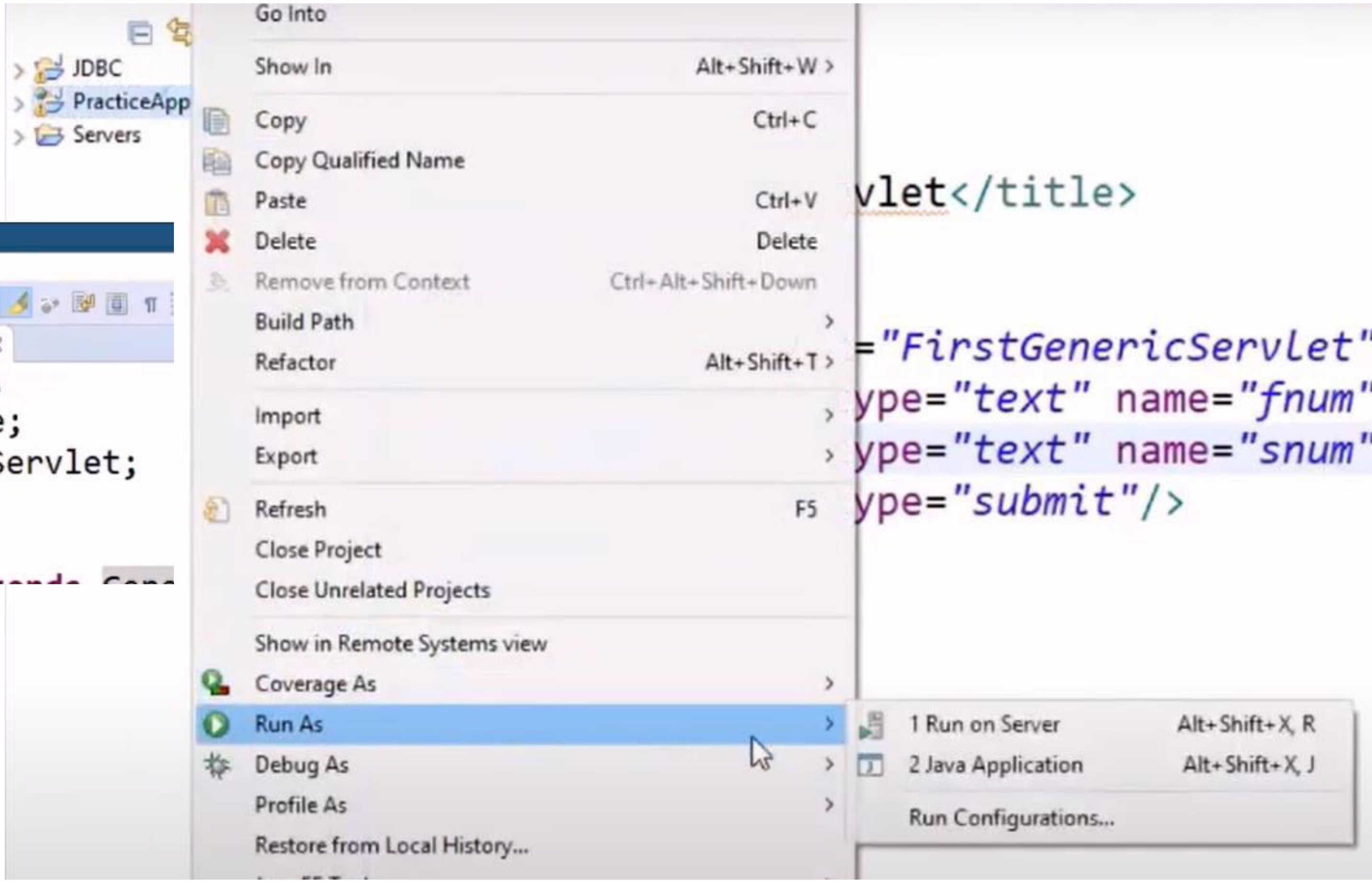

Servlets Tutorials
localhost:8084/PracticeApplicati
localhost:8084/PracticeApplication/FirstGenericServlet?fnum=15&snum=20
Apps
Gmail
YouTube
MySoftwareFree -...
Create a bootable...
Install Ubuntu desk...
bootstrap
Dash



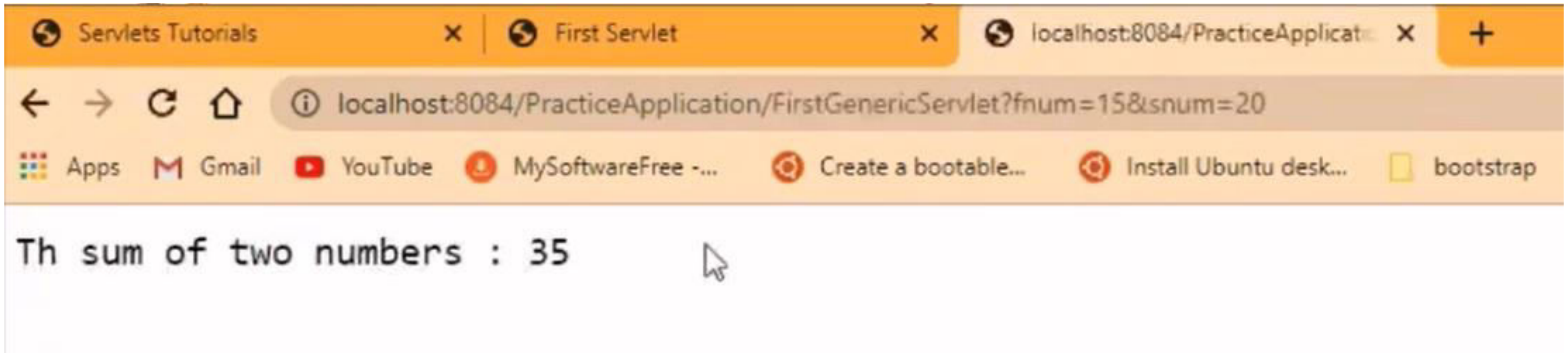
```
import java.io.IOException;
import javax.servlet.GenericServlet;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebServlet;
@WebServlet("/FirstGenericServlet")
public class FirstGenericServlet extends GenericServlet {
    private static final long serialVersionUID = 1L;

    public void service(ServletRequest request, ServletResponse response) throws ServletExcep
        int firstnum = Integer.parseInt(request.getParameter("fnum"));
        int secondnum = Integer.parseInt(request.getParameter("snum"));

        int sum = firstnum + secondnum;

        PrintWriter out = response.getWriter();
        out.print("Th sum of two numbers : " +sum);

    }

}
```


eclipse-workspace - PracticeApplication/src/co/in/kothaabhishek/FirstGenericServlet.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
JDBC
PracticeApplication
Servers
Selenium.java
JdbcPart4.java
index.html
FirstGenericServlet.java

```
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebServlet;

@WebServlet("/FirstGenericServlet")
```

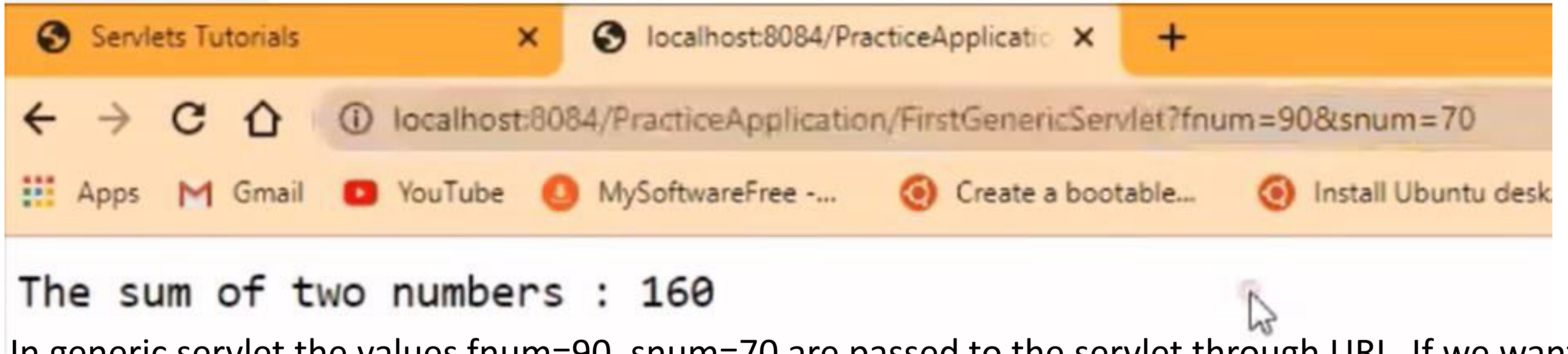


```
vlet</title>
="FirstGenericServlet"
ype="text" name="fnum"
ype="text" name="snum"
ype="submit"/>
```



```
Th sum of two numbers : 35
```

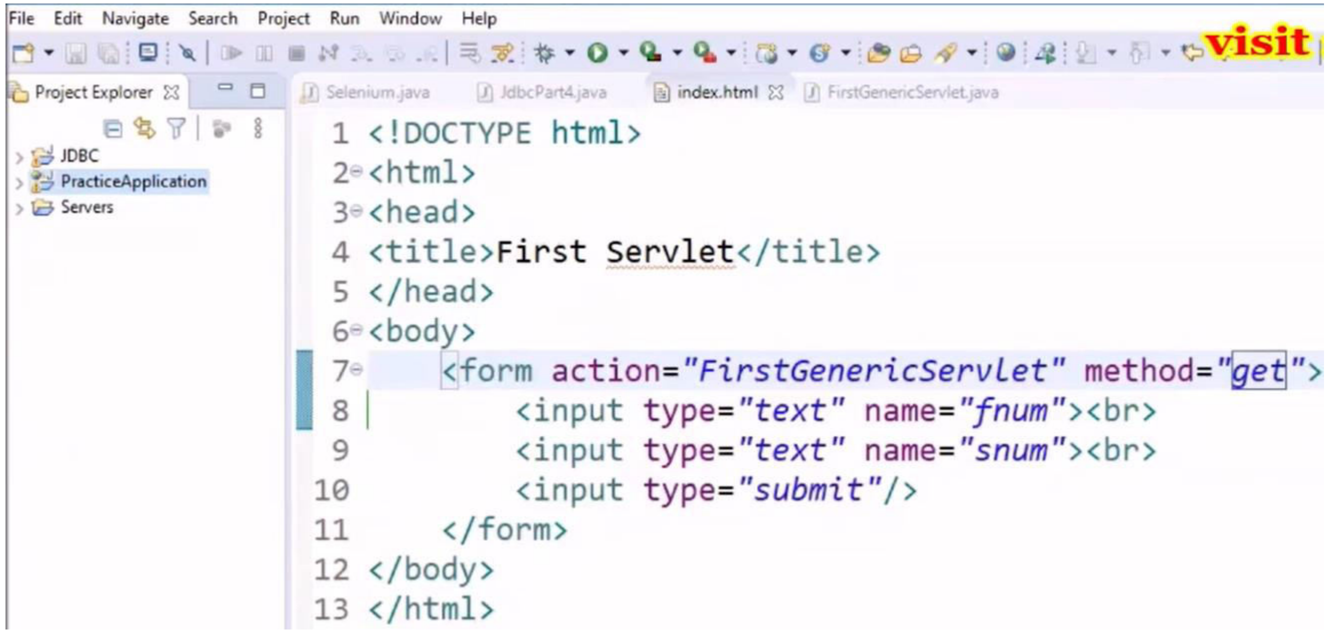
When we use get and post methods



```
localhost:8084/PracticeApplication/FirstGenericServlet?fnum=90&snum=70
```

```
The sum of two numbers : 160
```

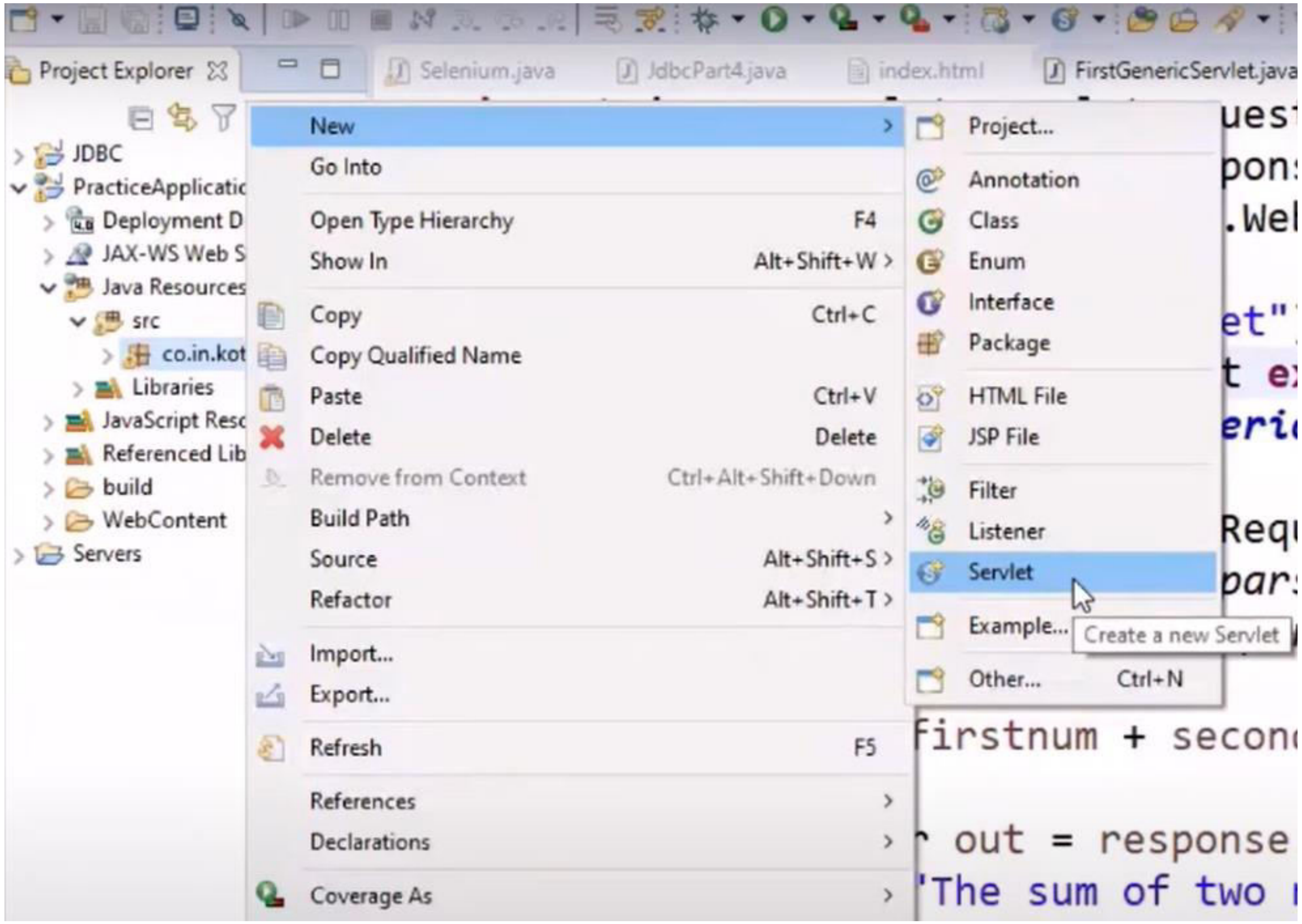
In generic servlet the values fnum=90, snum=70 are passed to the servlet through URL. If we want to read data from server to browser we use get method. If we want to read data from browser to server we use post method. If we use get method we can pass limited data. If we use post method we can pass unlimited data. If we use post method the data is passes through body so then can transmit unlimited data.



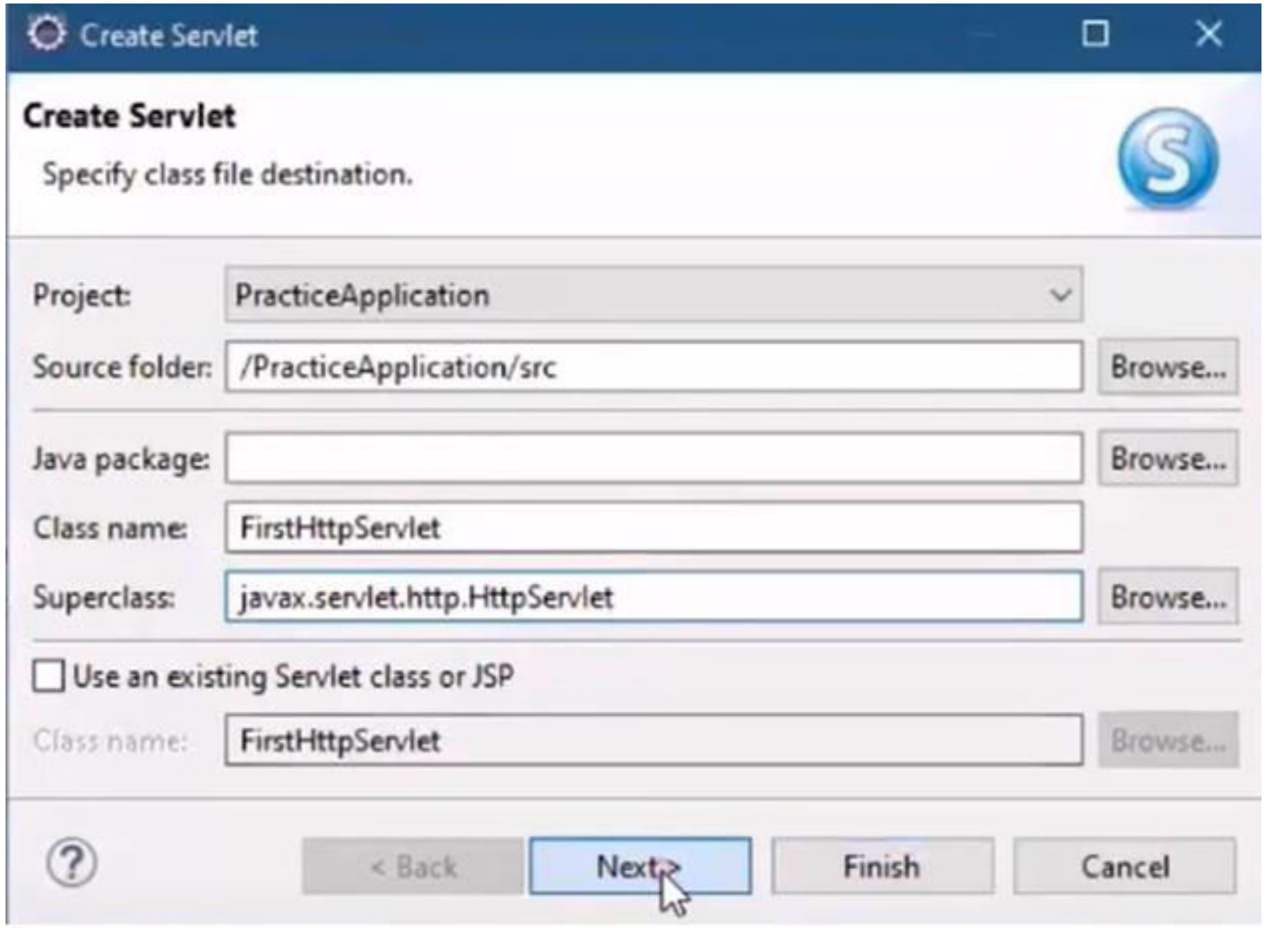
```
<!DOCTYPE html>
<html>
<head>
<title>First Servlet</title>
</head>
<body>
    <form action="FirstGenericServlet" method="get">
        <input type="text" name="fnum"><br>
        <input type="text" name="snum"><br>
        <input type="submit"/>
    </form>
</body>
</html>
```

By default method="get"

Http Servlet



```
firstnum + secon
 out = response
"The sum of two
```


Create Servlet

Create Servlet

Specify class file destination.

Project: PracticeApplication

Source folder: /PracticeApplication/src

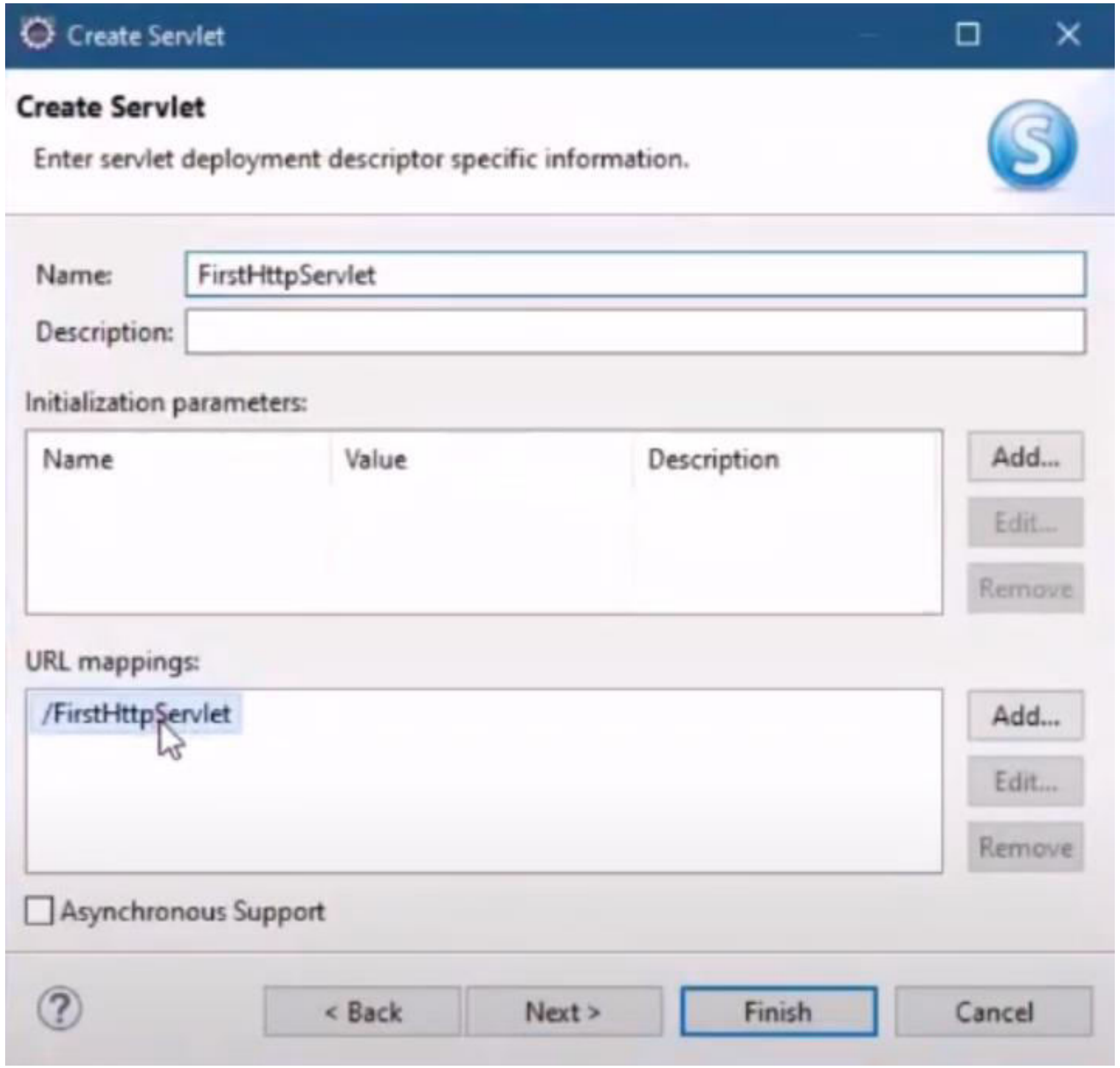
Java package:

Class name: FirstHttpServlet

Superclass: javax.servlet.http.HttpServlet

☐ Use an existing Servlet class or JSP

Class name: FirstHttpServlet


Create Servlet

Create Servlet

Enter servlet deployment descriptor specific information.

Name: FirstHttpServlet

Description:

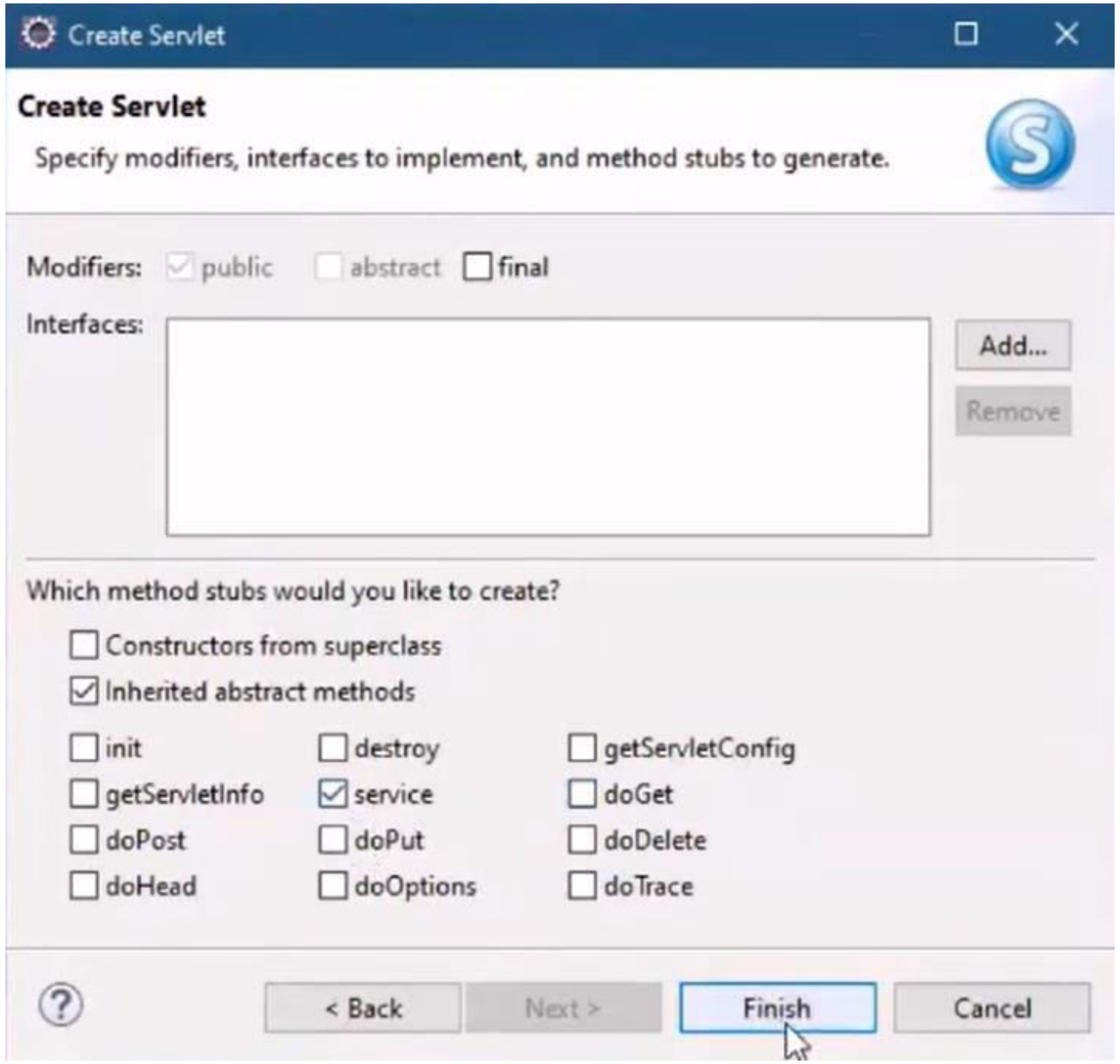
Initialization parameters:

Name	Value	Description

URL mappings:

/FirstHttpServlet

☐ Asynchronous Support


Create Servlet

Create Servlet

Specify modifiers, interfaces to implement, and method stubs to generate.

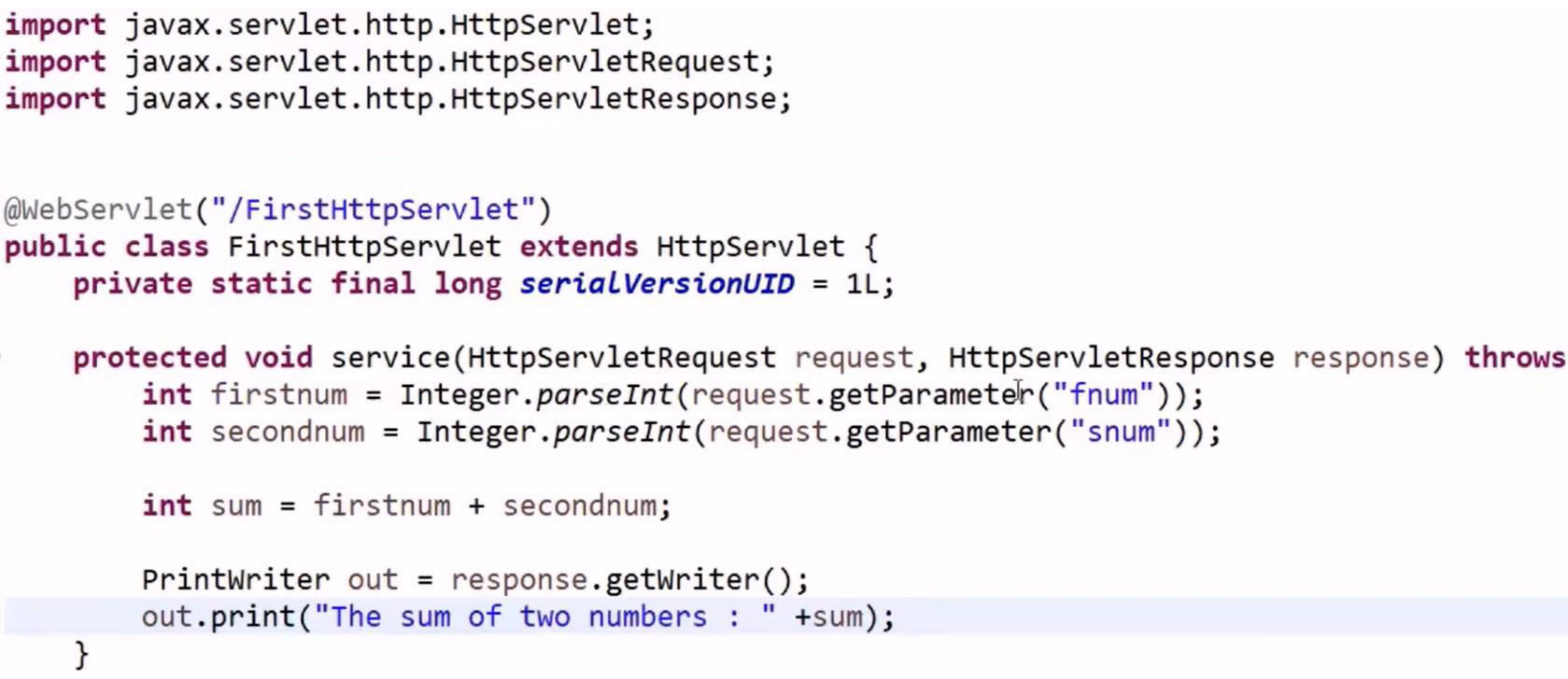
Modifiers: [✓ public] abstract final

Interfaces:

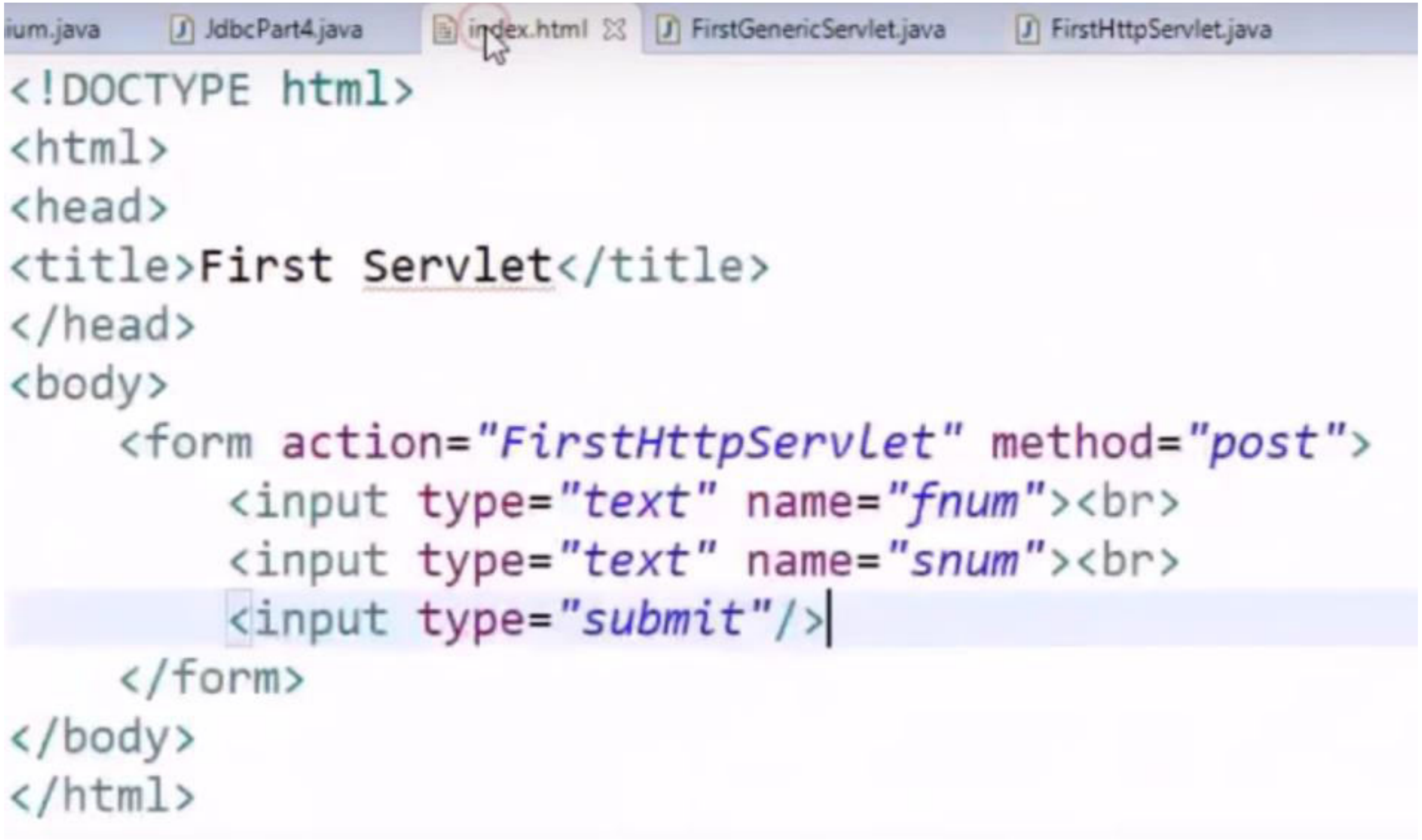
Which method stubs would you like to create?

Constructors from superclass
[✓ Inherited abstract methods]

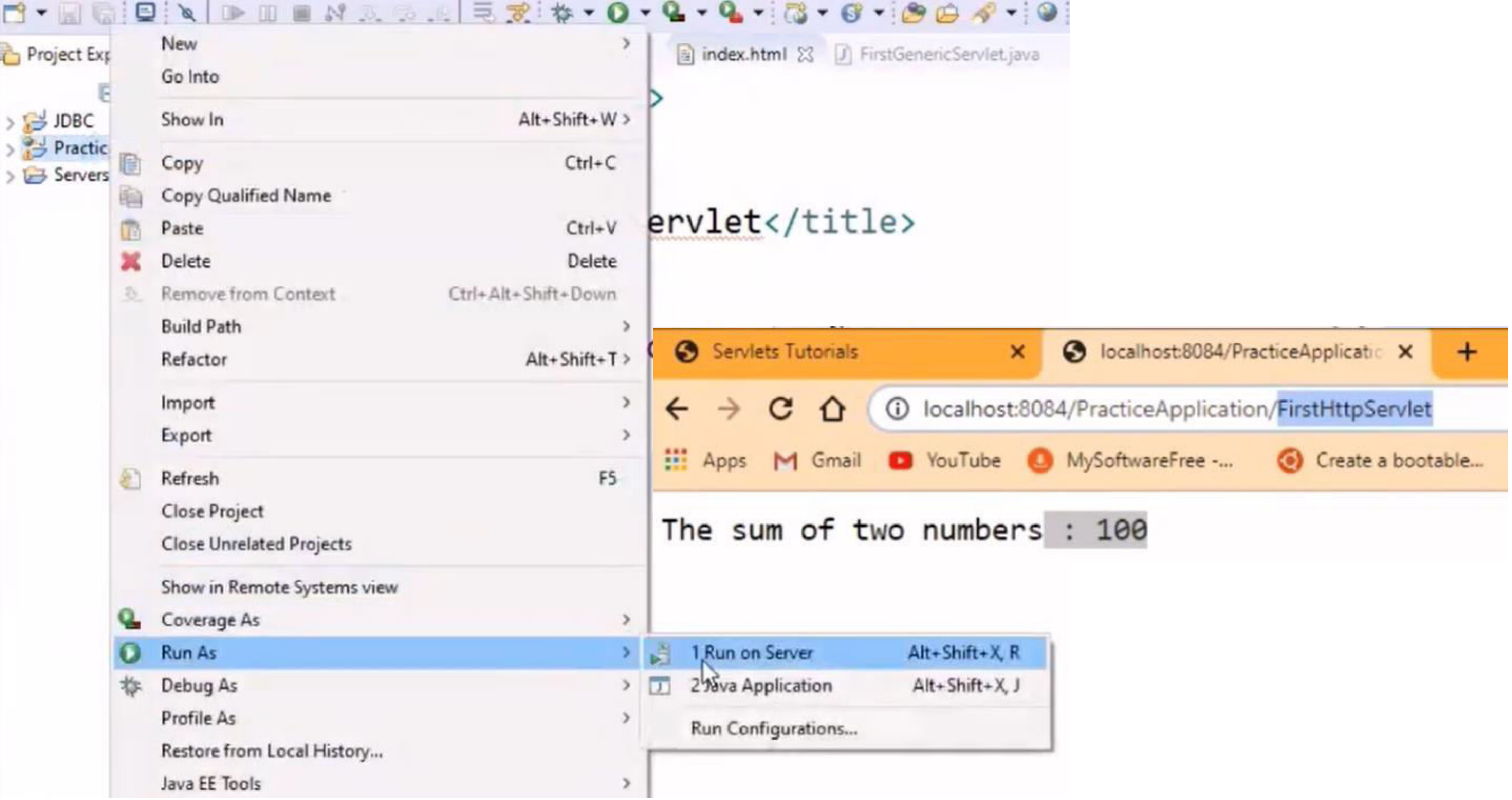
init	destroy	getServletConfig
getServletInfo	[✓ service]	doGet
doPost	doPut	doDelete
doHead	doOptions	doTrace



```java
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


@WebServlet("/FirstHttpServlet")
public class FirstHttpServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void service(HttpServletRequest request, HttpServletResponse response) throws
        int firstnum = Integer.parseInt(request.getParameter("fnum"));
        int secondnum = Integer.parseInt(request.getParameter("snum"));

        int sum = firstnum + secondnum;

        PrintWriter out = response.getWriter();
        out.print("The sum of two numbers : " +sum);
    }
```



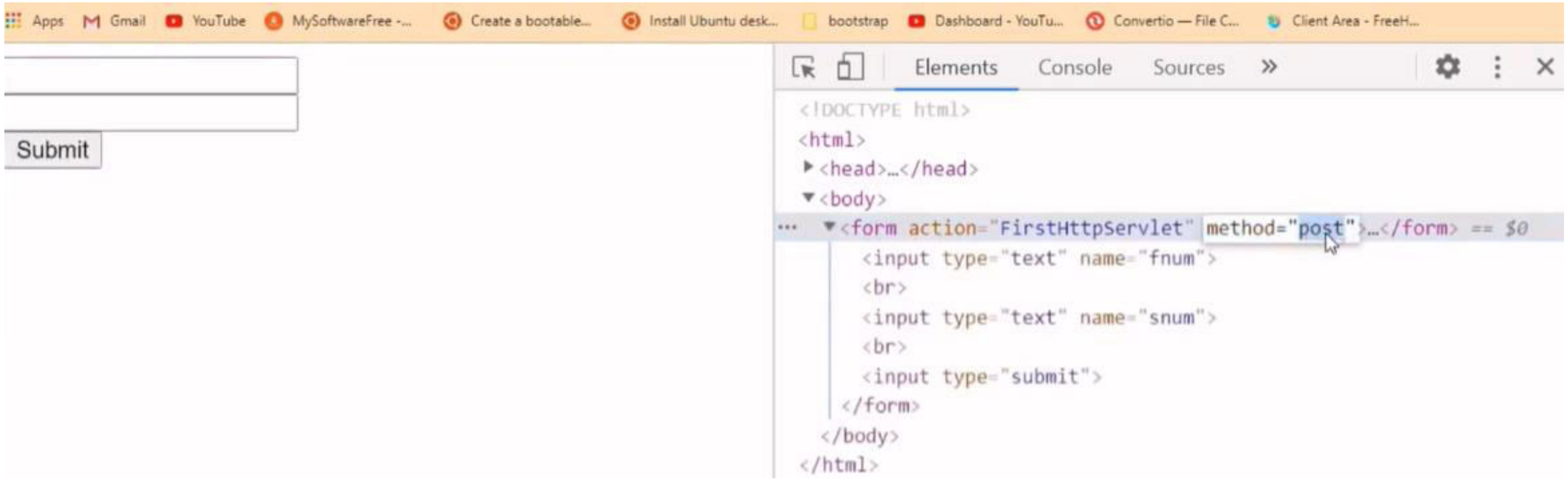
```html
<!DOCTYPE html>
<html>
<head>
<title>First Servlet</title>
</head>
<body>
    <form action="FirstHttpServlet" method="post">
        <input type="text" name="fnum"><br>
        <input type="text" name="snum"><br>
        <input type="submit"/>
    </form>
</body>
</html>
```



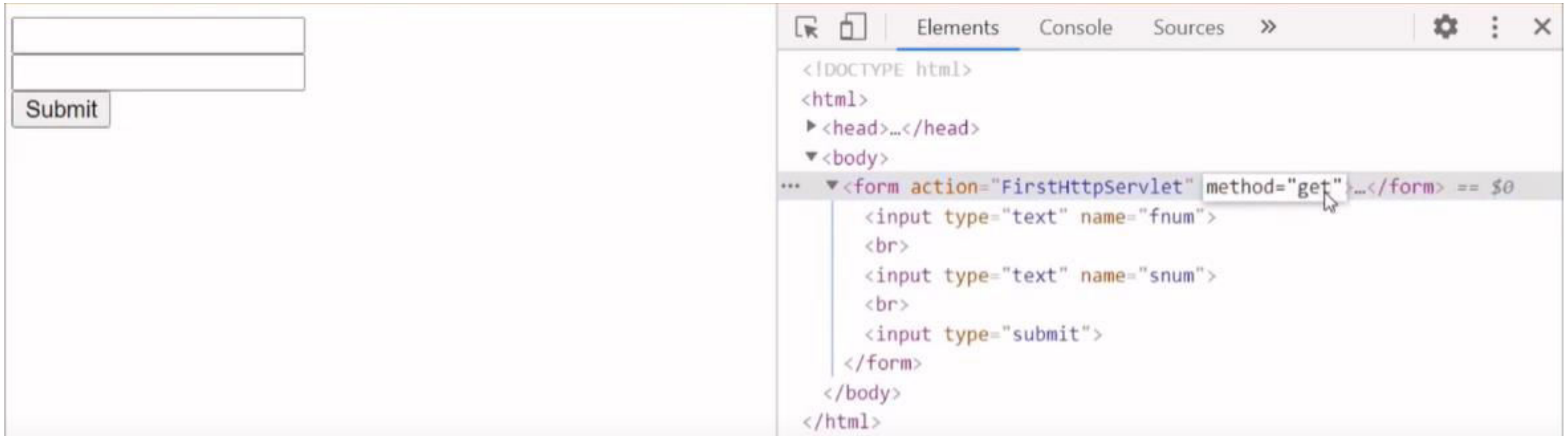
```
ervlet</title>
```

localhost:8084/PracticeApplication/FirstHttpServlet

```
The sum of two numbers : 100
```

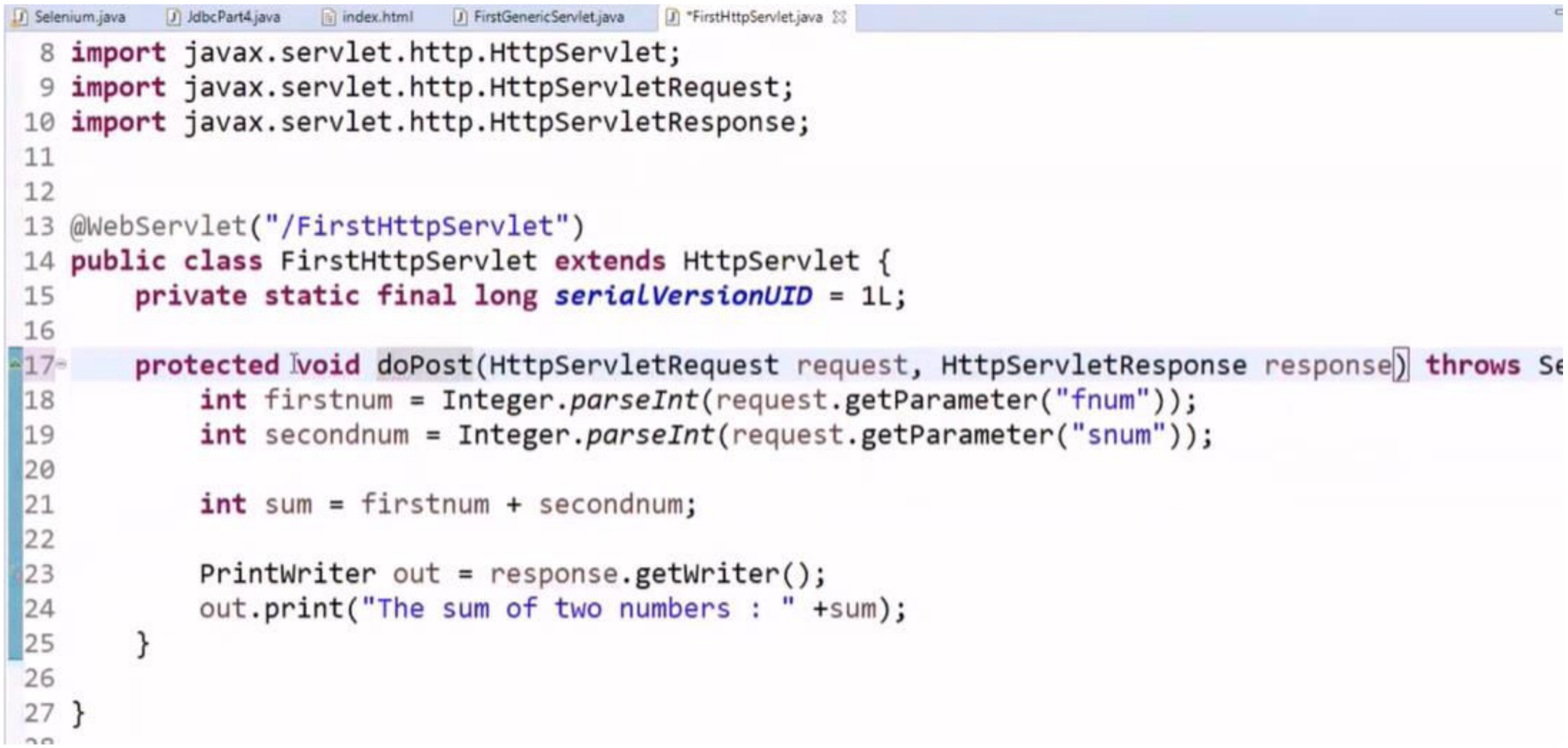


```
<!DOCTYPE html>
<html>
▸<head>…</head>
▾<body>
  ▾<form action="FirstHttpServlet" method="post">…</form> == $0
      <input type="text" name="fnum">
      <br>
      <input type="text" name="snum">
      <br>
      <input type="submit">
    </form>
  </body>
</html>
```

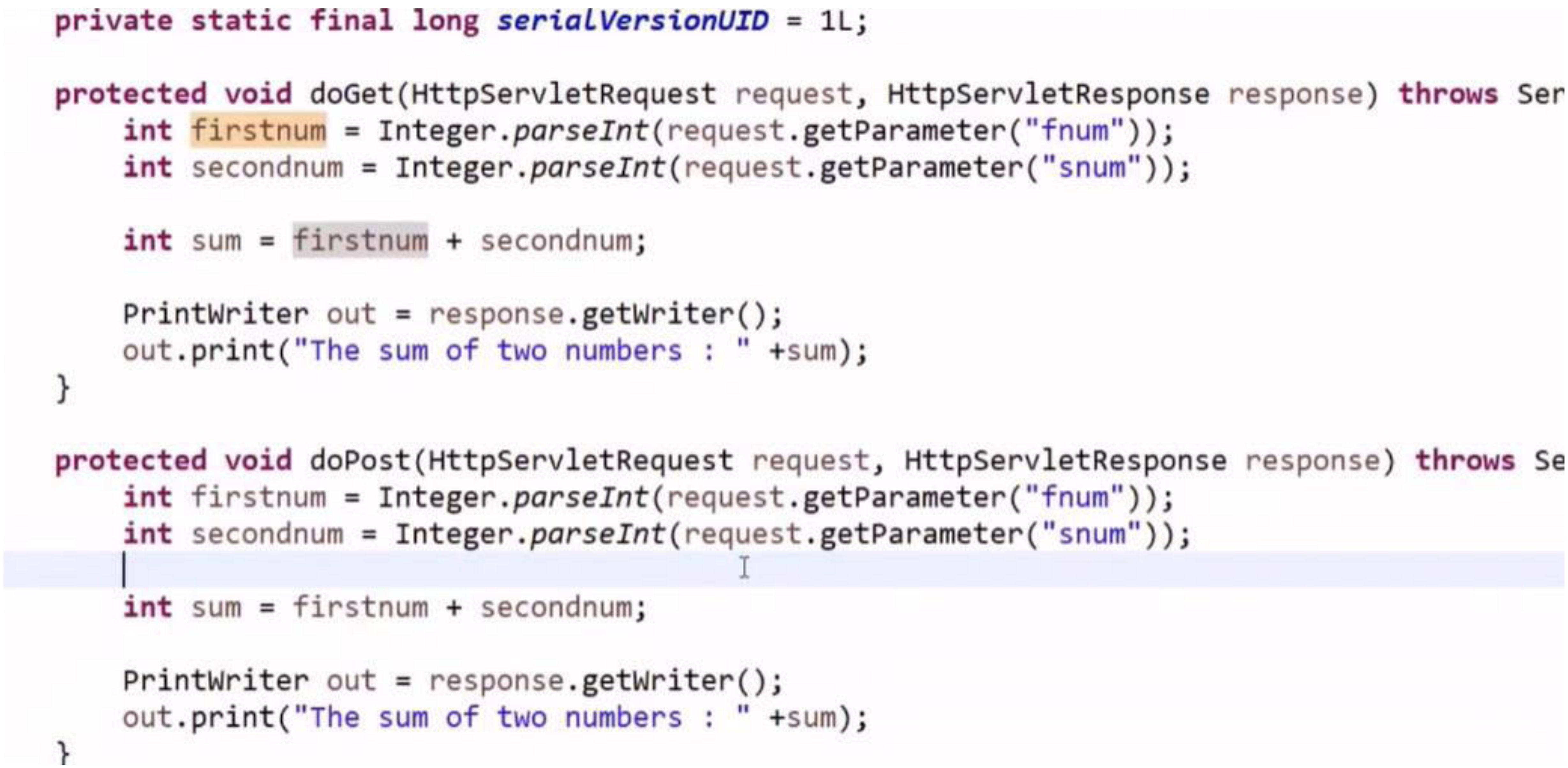


```
<!DOCTYPE html>
<html>
▸<head>…</head>
▾<body>
  ▾<form action="FirstHttpServlet" method="get">…</form> == $0
      <input type="text" name="fnum">
      <br>
      <input type="text" name="snum">
      <br>
      <input type="submit">
    </form>
  </body>
</html>
```

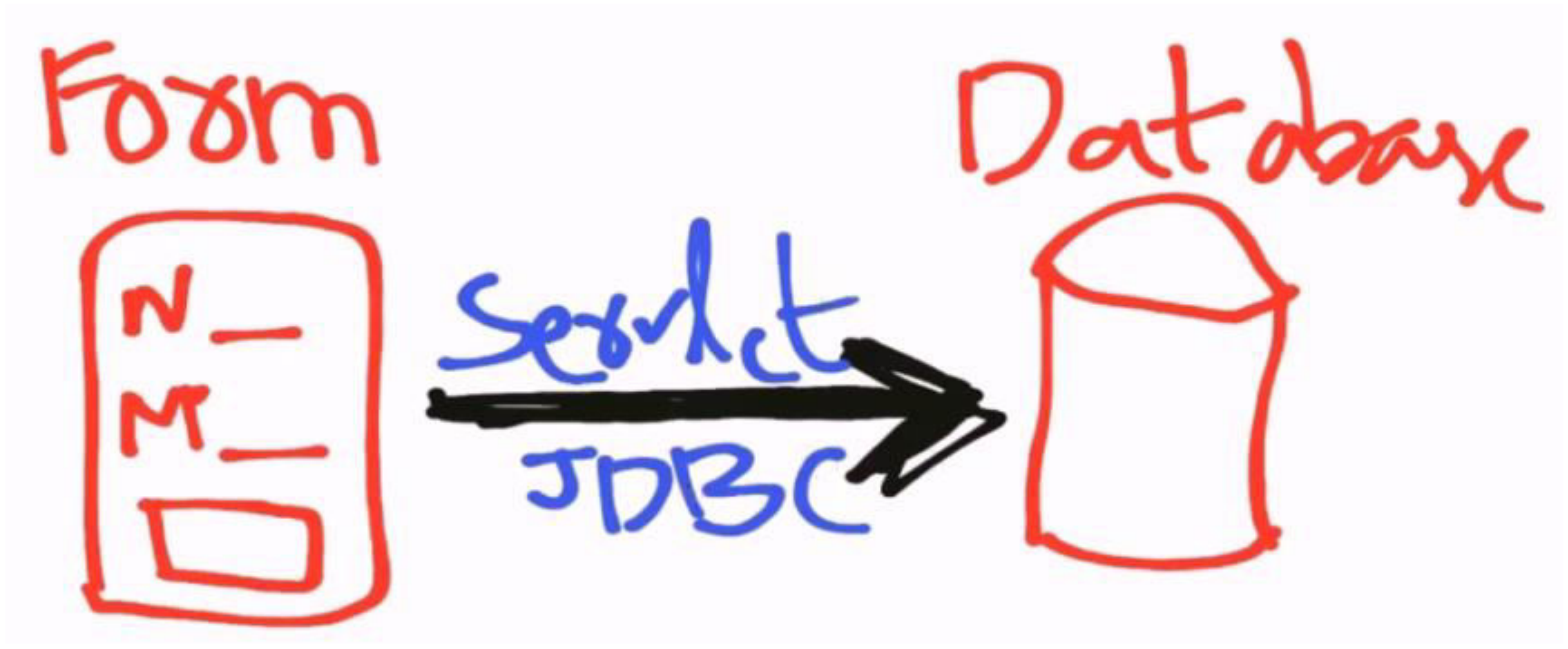
Eventhough we use post in .html file, if we change the method=get (while running right click->inspect->change method=get) once again we get values in the URL. To avoid this problem, instead of using service() method we use doPost() method.



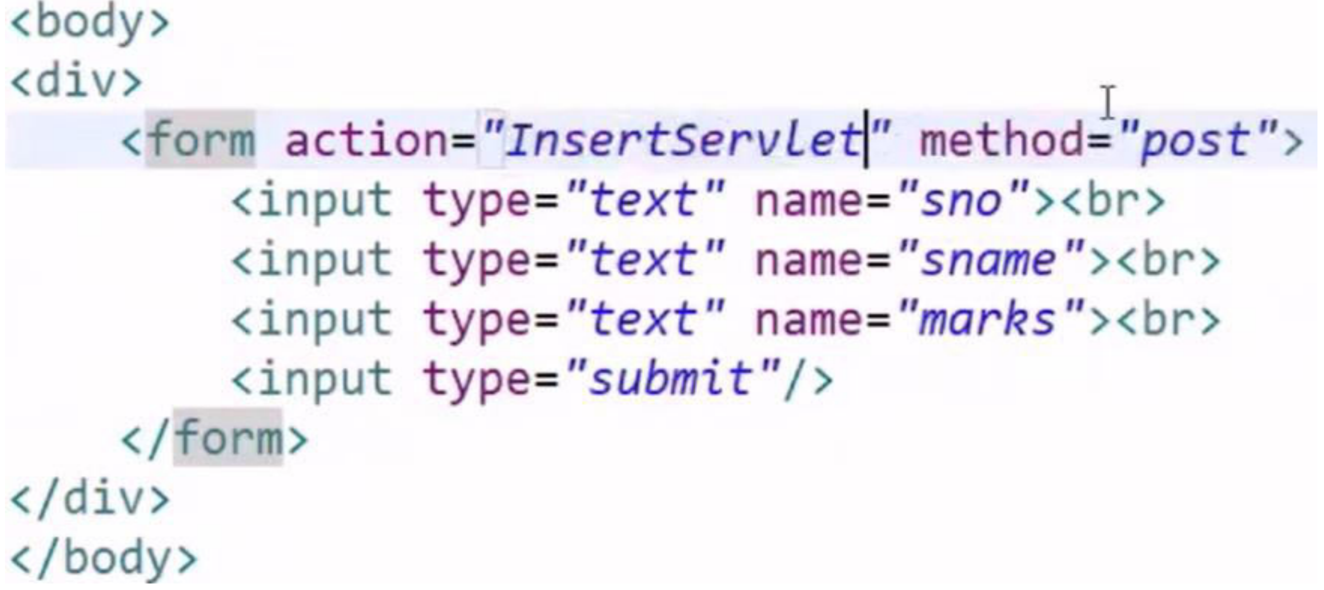
```java
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


@WebServlet("/FirstHttpServlet")
public class FirstHttpServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws Se
        int firstnum = Integer.parseInt(request.getParameter("fnum"));
        int secondnum = Integer.parseInt(request.getParameter("snum"));

        int sum = firstnum + secondnum;

        PrintWriter out = response.getWriter();
        out.print("The sum of two numbers : " +sum);
    }

}
```



```
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws Ser
        int firstnum = Integer.parseInt(request.getParameter("fnum"));
        int secondnum = Integer.parseInt(request.getParameter("snum"));

        int sum = firstnum + secondnum;

        PrintWriter out = response.getWriter();
        out.print("The sum of two numbers : " +sum);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws Se
        int firstnum = Integer.parseInt(request.getParameter("fnum"));
        int secondnum = Integer.parseInt(request.getParameter("snum"));

        int sum = firstnum + secondnum;

        PrintWriter out = response.getWriter();
        out.print("The sum of two numbers : " +sum);
    }
```

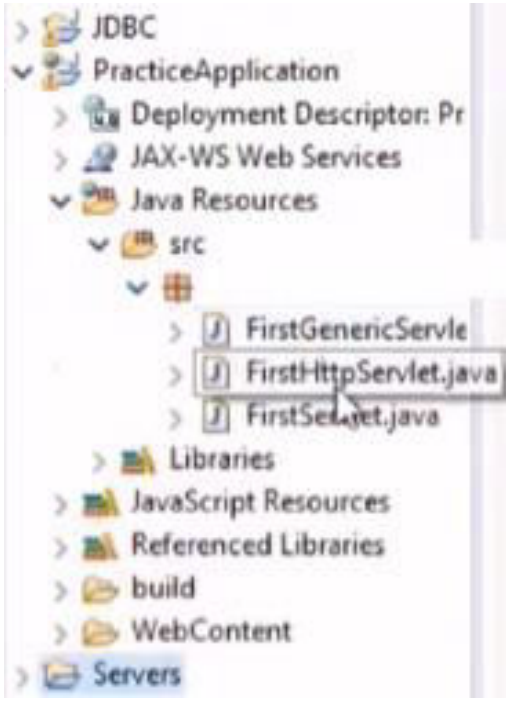


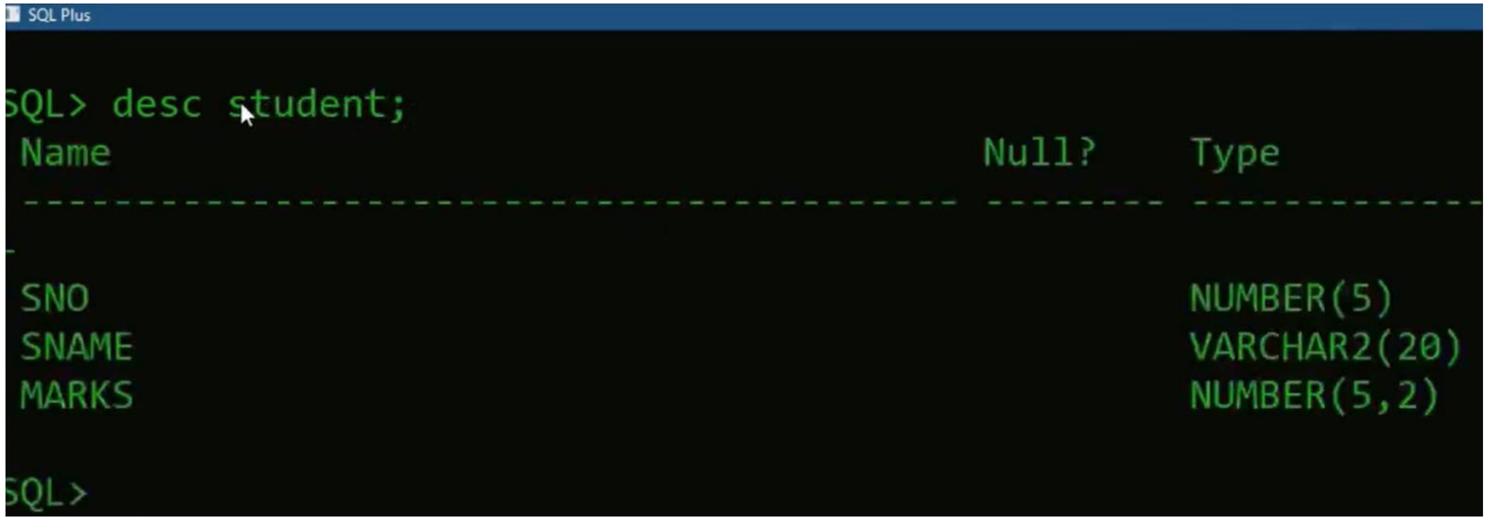
In index.html we create a form. Using servlets we access the data from the html file. By using JDBC we store the accessed data into the database. Through sql commands we can insert data otherwise we can also insert data through forms and servlets.



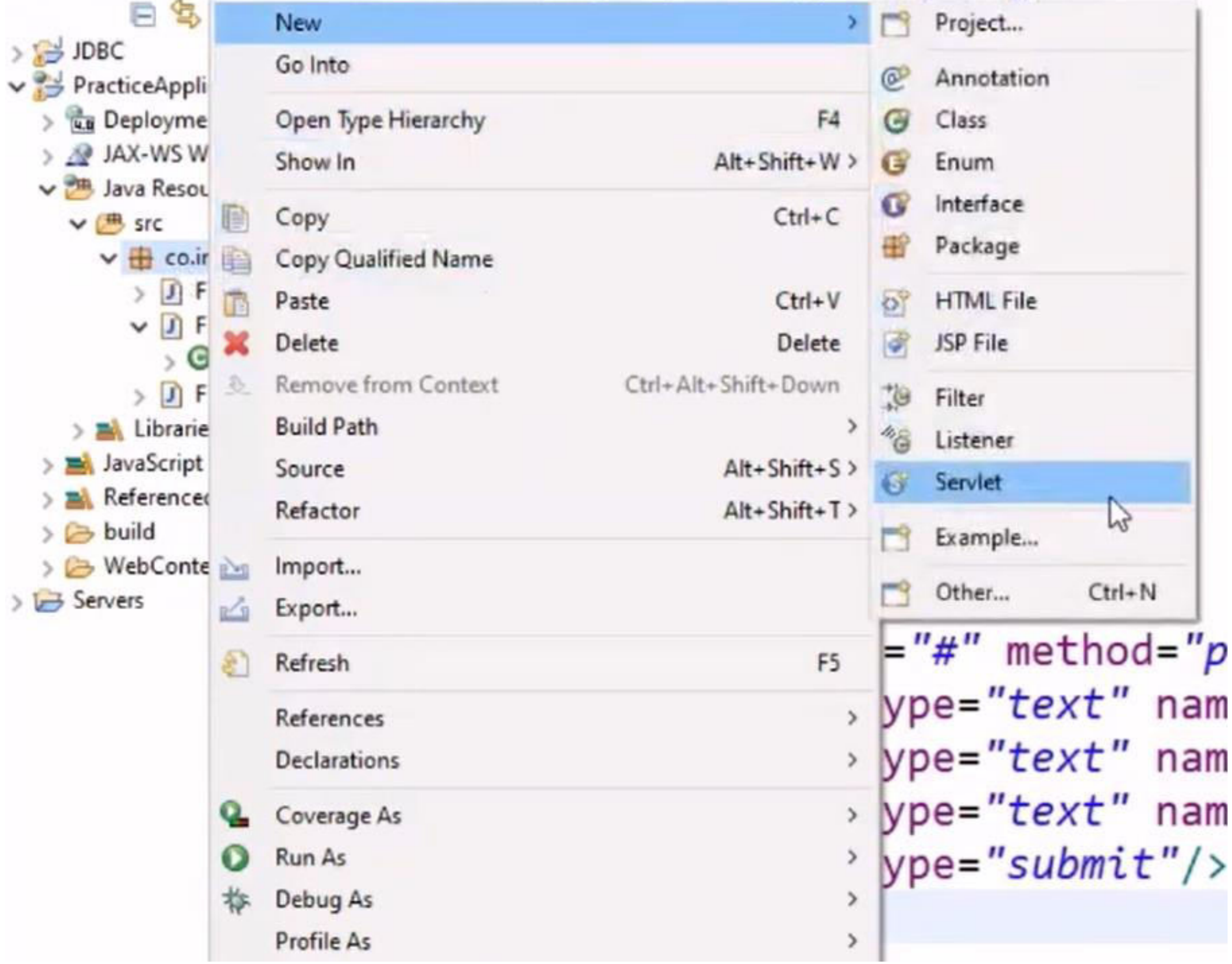
```html
<body>
<div>
    <form action="InsertServlet" method="post">
        <input type="text" name="sno"><br>
        <input type="text" name="sname"><br>
        <input type="text" name="marks"><br>
        <input type="submit"/>
    </form>
</div>
</body>
```

Index.html

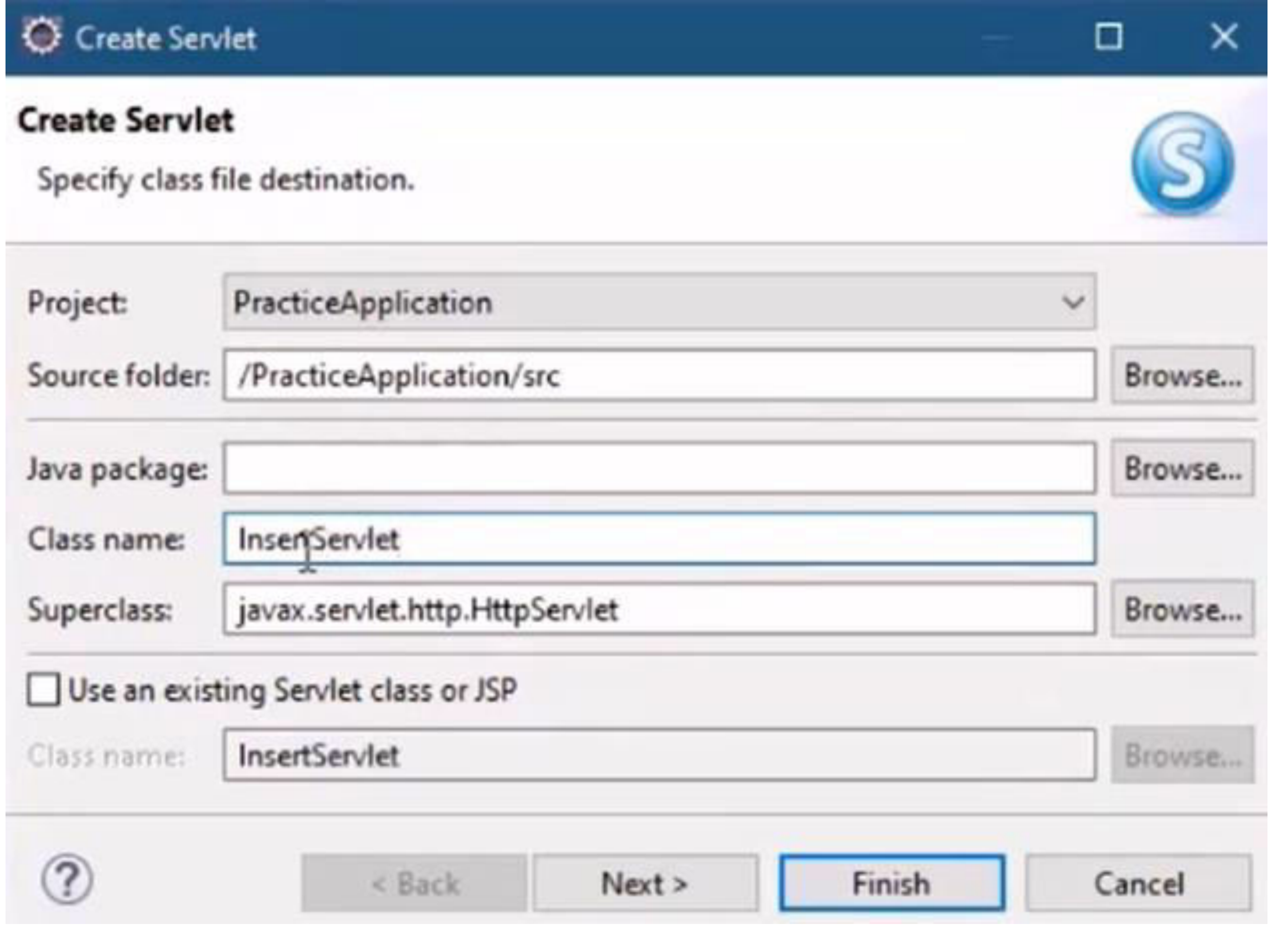




```
SQL> desc student;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
-

 SNO                                                NUMBER(5)
 SNAME                                              VARCHAR2(20)
 MARKS                                              NUMBER(5,2)

SQL>
```



```
="#" method="p
ype="text" nam
ype="text" nam
ype="text" nam
ype="submit"/>
```



Create Servlet

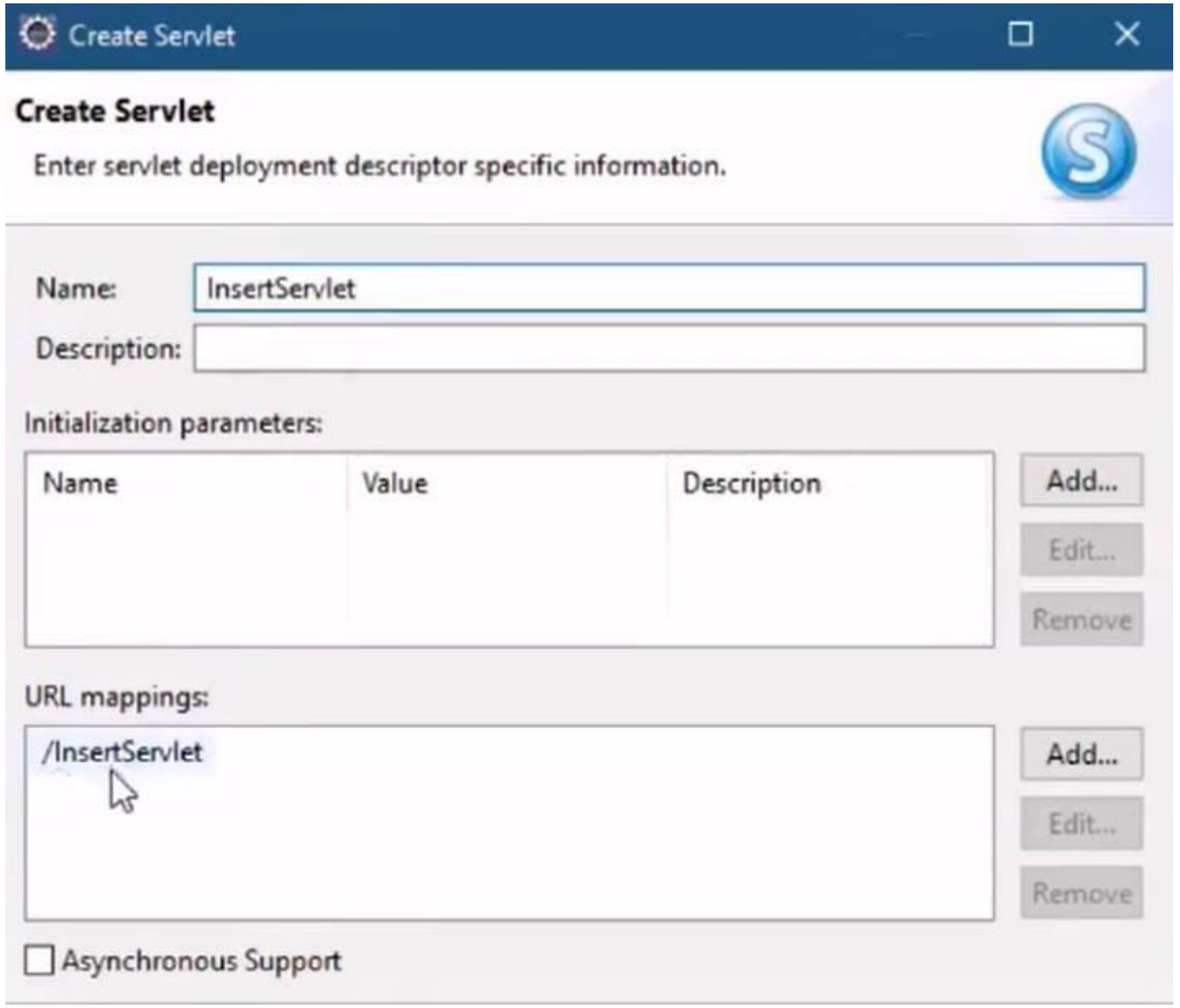
Create Servlet

Specify class file destination.

Project:	PracticeApplication	
Source folder:	/PracticeApplication/src	Browse...
Java package:		Browse...
Class name:	InsertServlet	
Superclass:	javax.servlet.http.HttpServlet	Browse...

☐ Use an existing Servlet class or JSP

Class name: InsertServlet



Create Servlet

Create Servlet

Enter servlet deployment descriptor specific information.

Name: InsertServlet

Description:

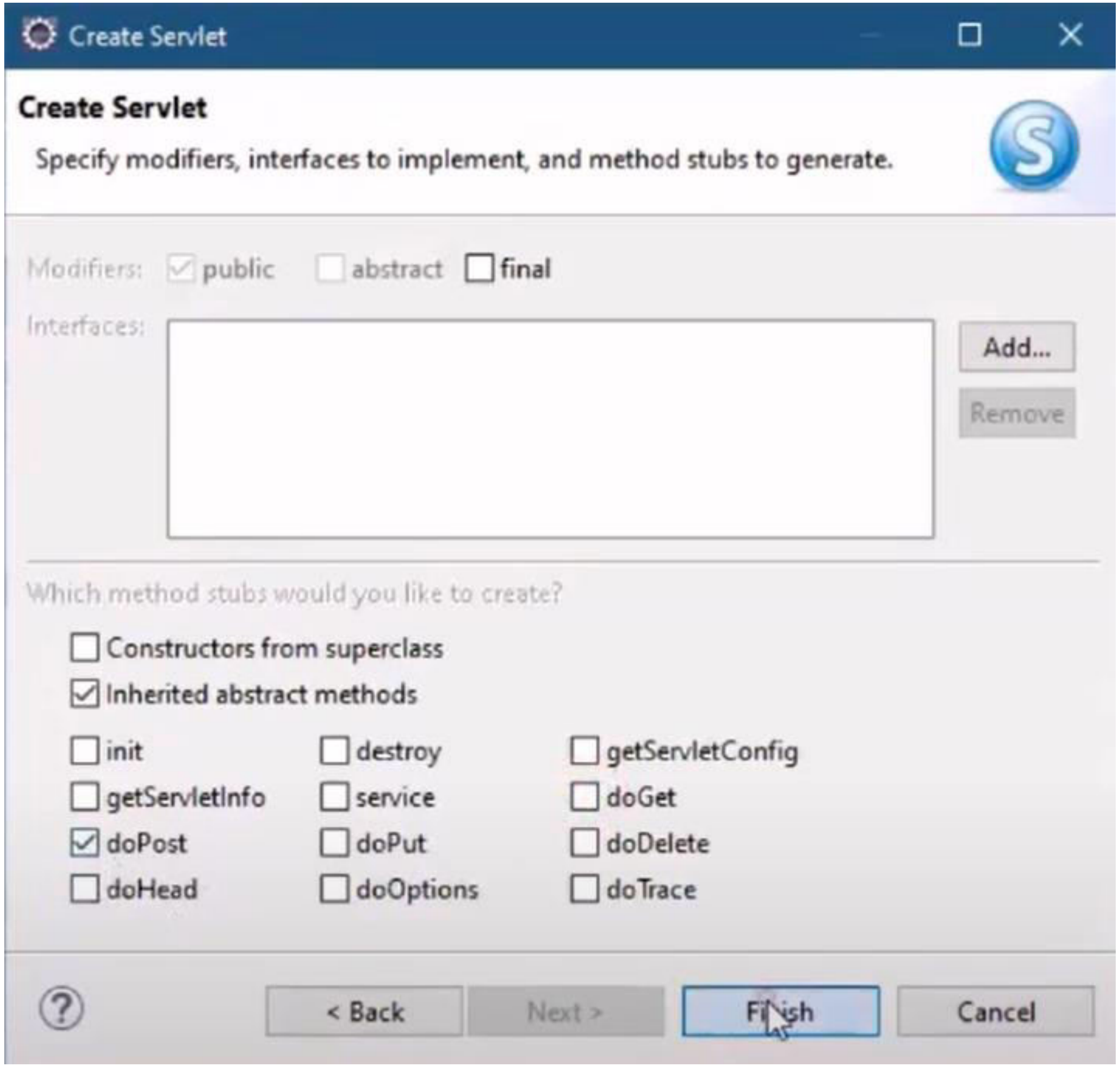
Initialization parameters:

Name	Value	Description

URL mappings:

/InsertServlet

☐ Asynchronous Support


Create Servlet

Create Servlet

Specify modifiers, interfaces to implement, and method stubs to generate.

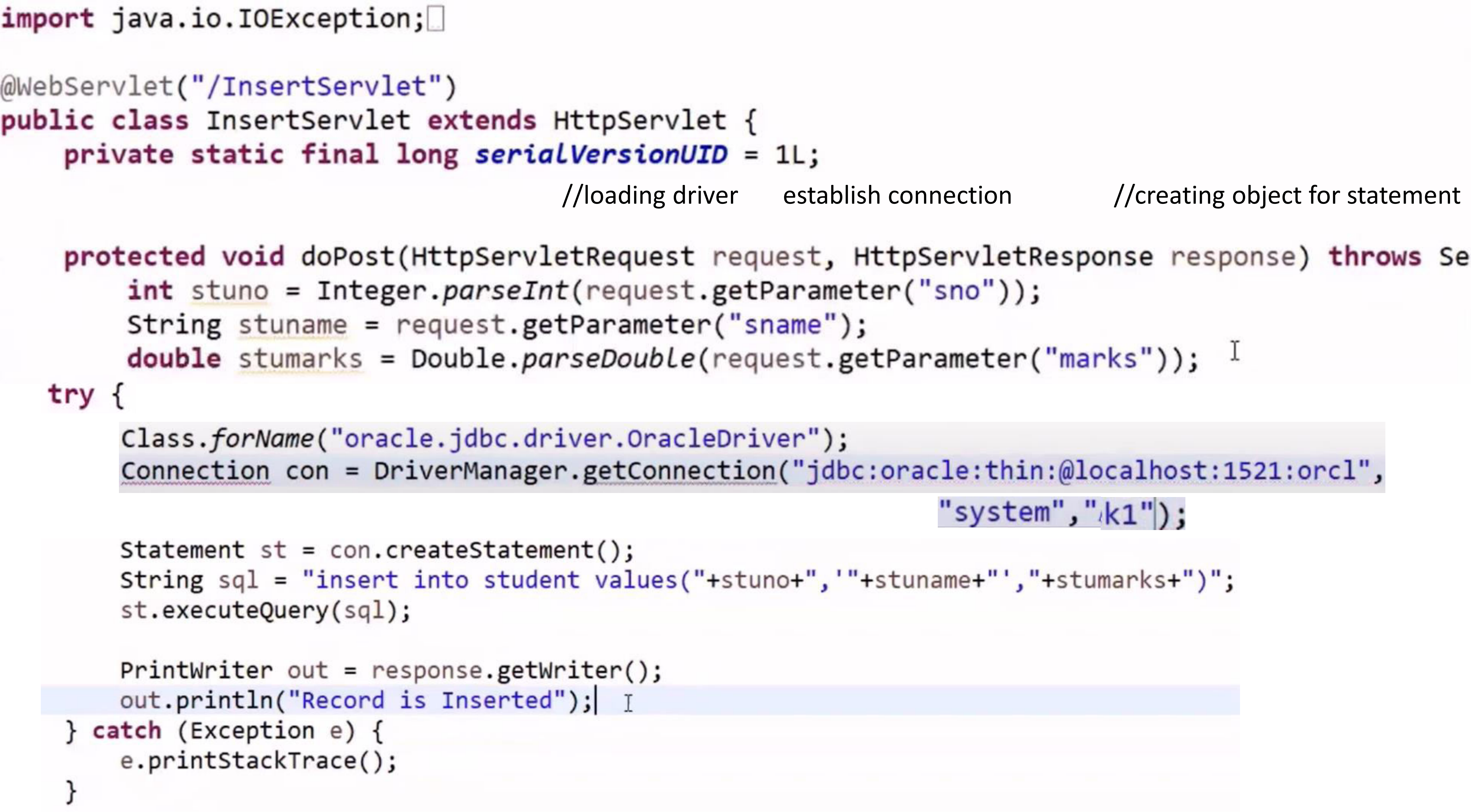
Modifiers: [✓ public] abstract final

Interfaces:

Which method stubs would you like to create?

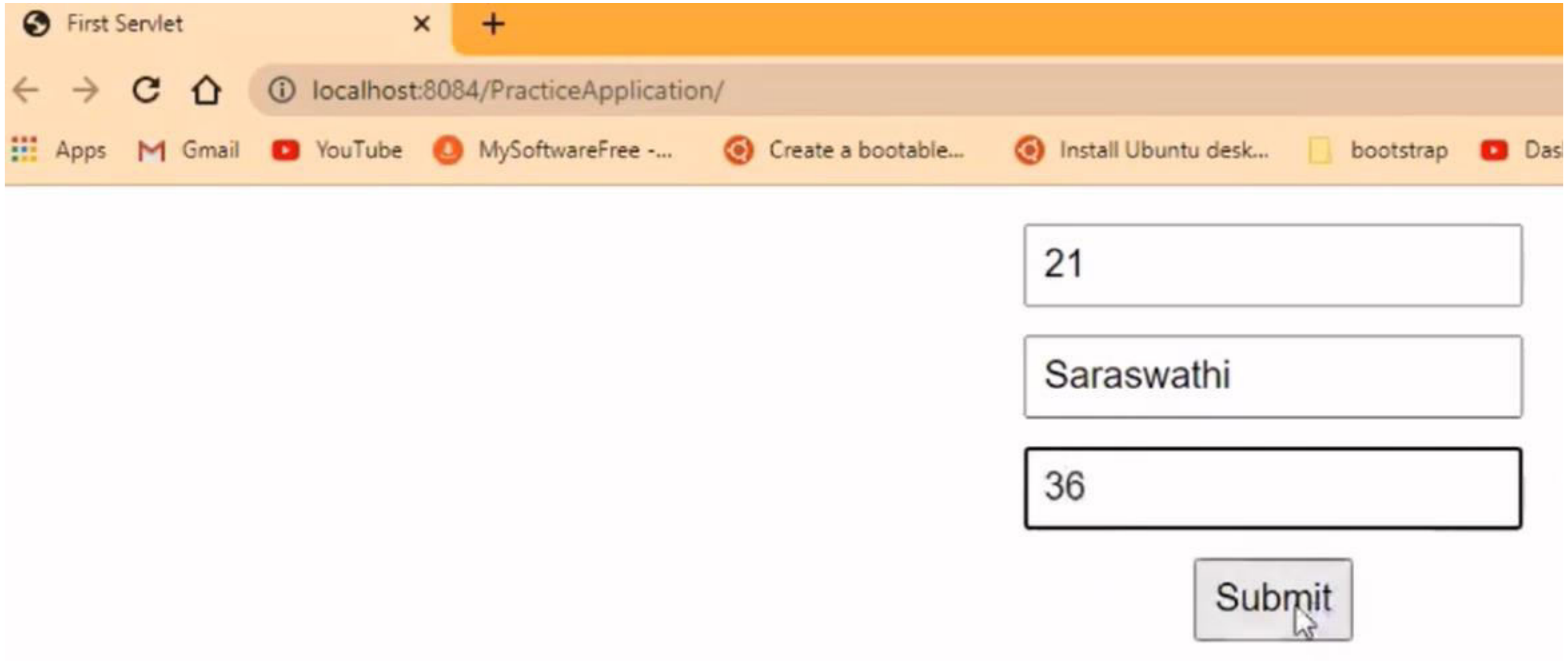
- [ ] Constructors from superclass
- [✓] Inherited abstract methods

[ ] init	[ ] destroy	[ ] getServletConfig
[ ] getServletInfo	[ ] service	[ ] doGet
[✓] doPost	[ ] doPut	[ ] doDelete
[ ] doHead	[ ] doOptions	[ ] doTrace



//loading driver establish connection //creating object for statement

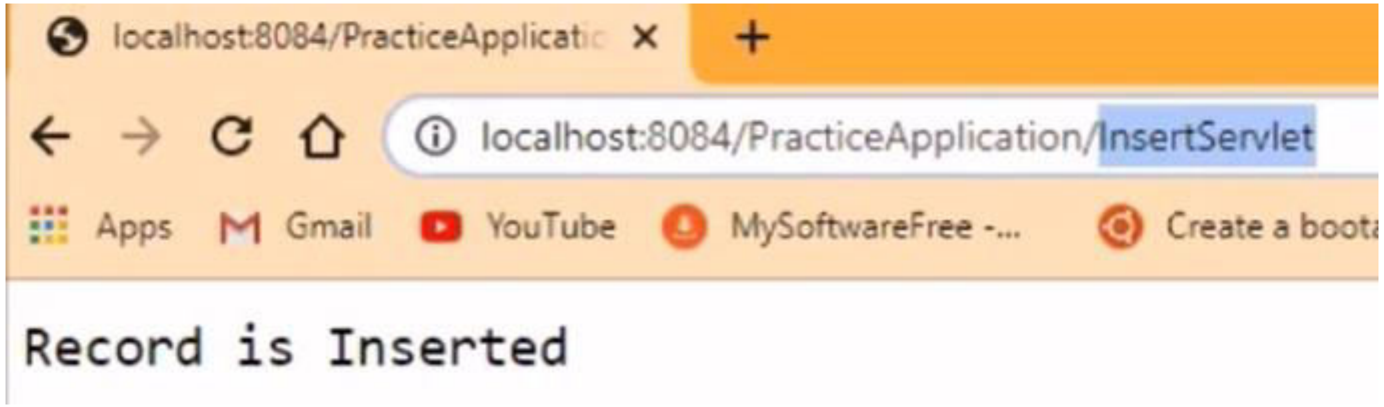
```
import java.io.IOException;

@WebServlet("/InsertServlet")
public class InsertServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws Se
        int stuno = Integer.parseInt(request.getParameter("sno"));
        String stuname = request.getParameter("sname");
        double stumarks = Double.parseDouble(request.getParameter("marks"));
   try {
        Class.forName("oracle.jdbc.driver.OracleDriver");
        Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl",
                                                                      "system","k1");

        Statement st = con.createStatement();
        String sql = "insert into student values("+stuno+",'"+stuname+"',"+stumarks+")";
        st.executeQuery(sql);

        PrintWriter out = response.getWriter();
        out.println("Record is Inserted");
    } catch (Exception e) {
        e.printStackTrace();
    }
```



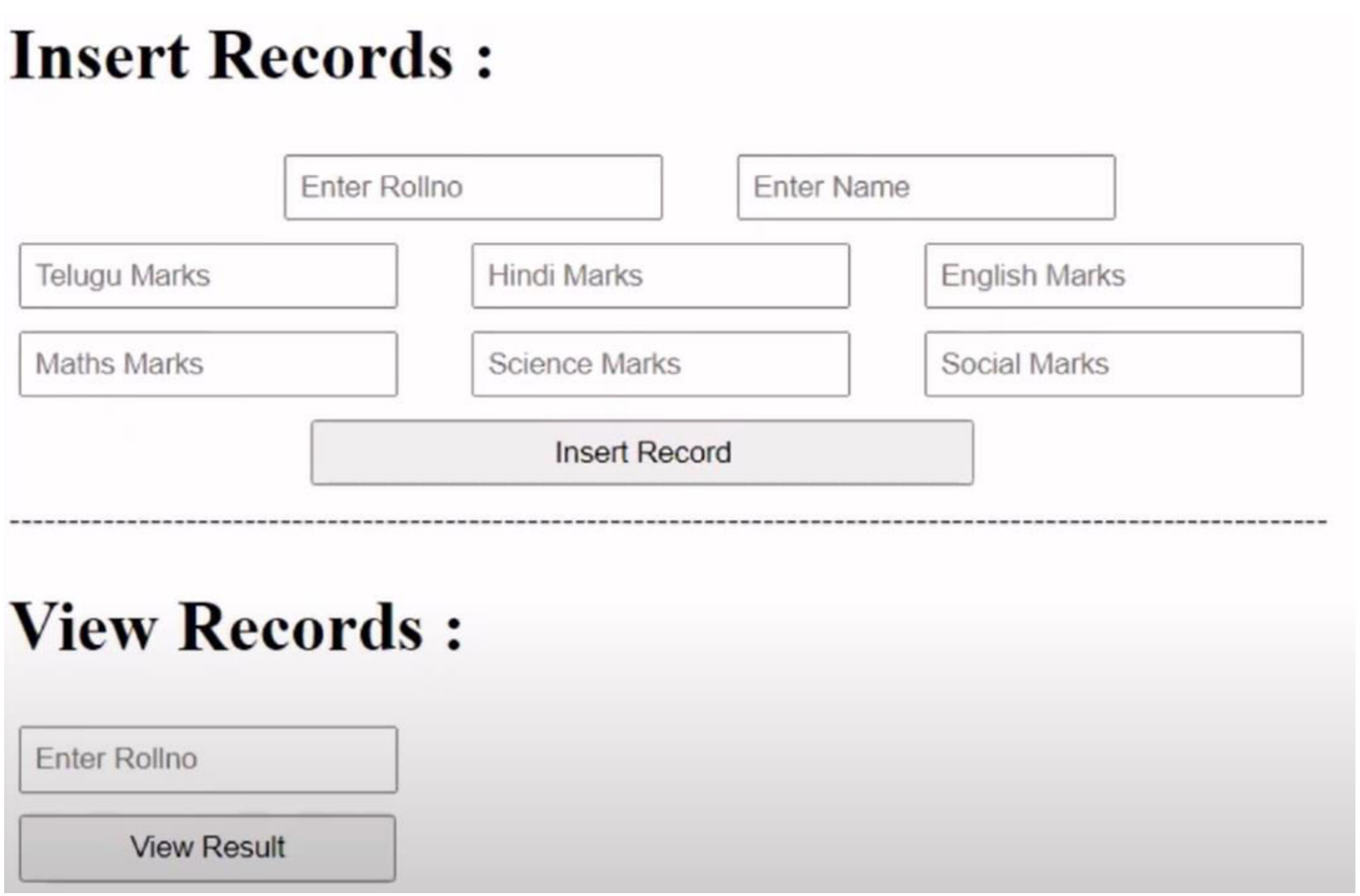
21

Saraswathi

36

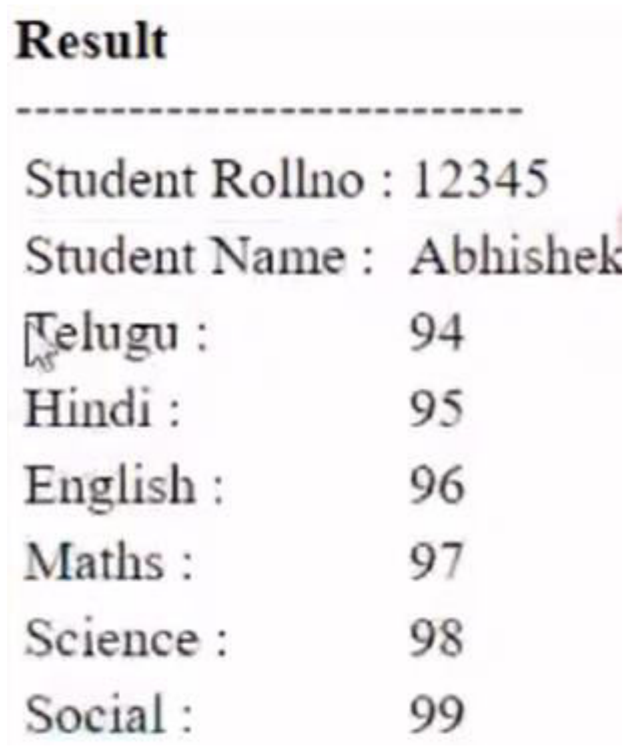

localhost:8084/PracticeApplicatio
localhost:8084/PracticeApplication/InsertServlet
Apps
Gmail
YouTube
MySoftwareFree -...

```
Record is Inserted
```



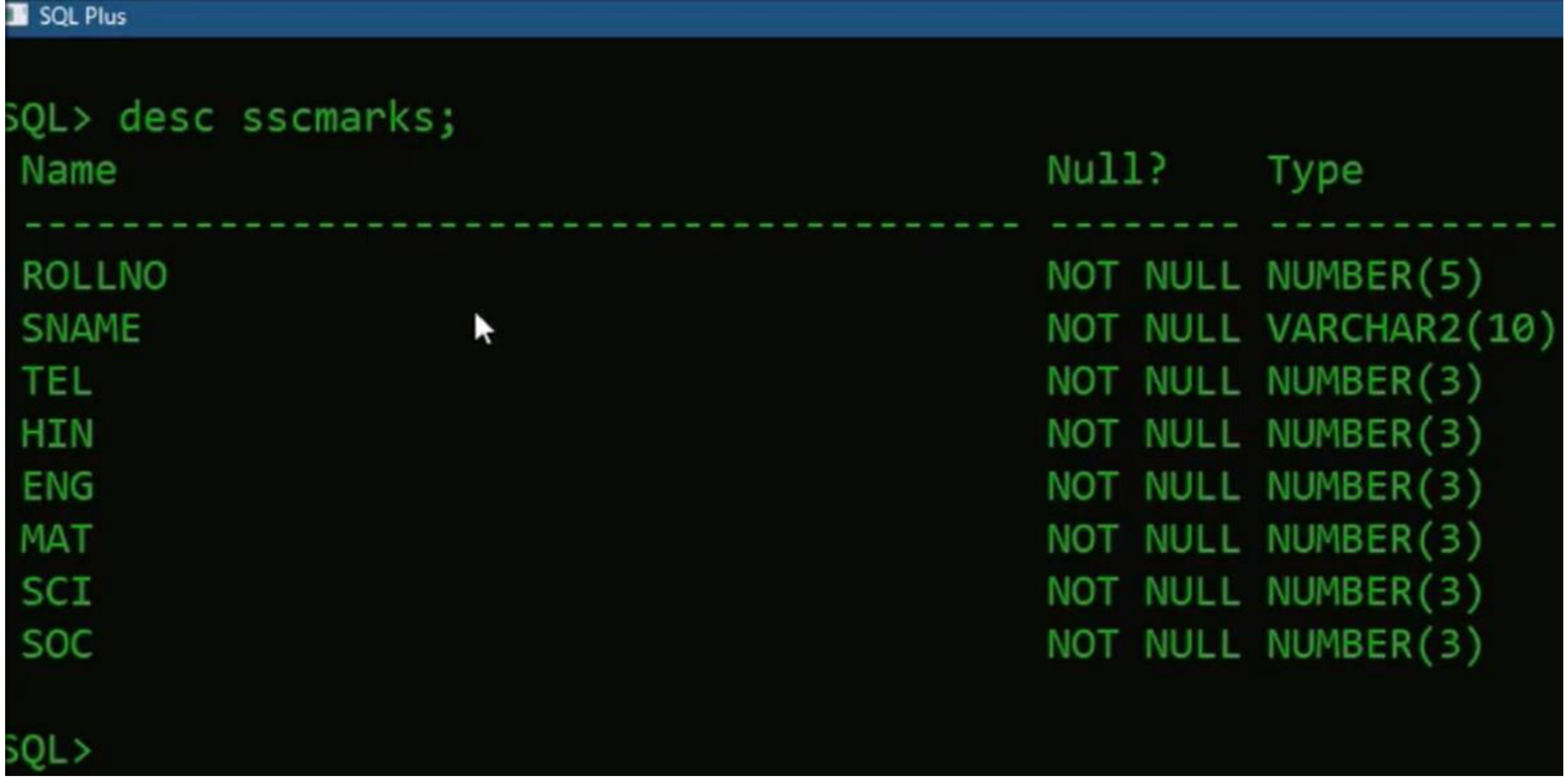
Insert Records :

View Records :

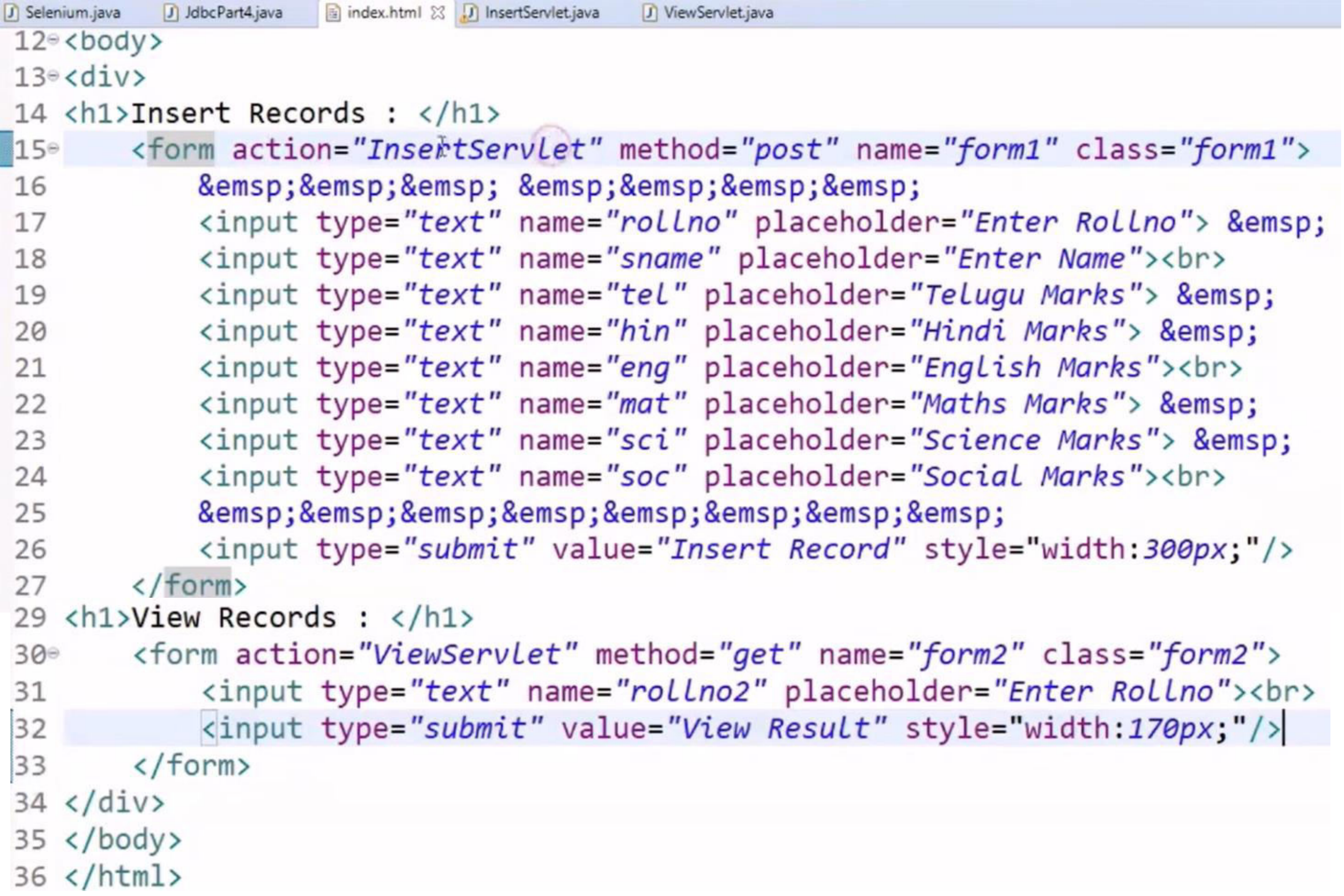


Result

```
-------------------------
Student Rollno : 12345
Student Name :  Abhishek
Telugu :        94
Hindi :         95
English :       96
Maths :         97
Science :       98
Social :        99
```


SQL Plus

```
QL> desc sscmarks;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 ROLLNO                                    NOT NULL NUMBER(5)
 SNAME                                     NOT NULL VARCHAR2(10)
 TEL                                       NOT NULL NUMBER(3)
 HIN                                       NOT NULL NUMBER(3)
 ENG                                       NOT NULL NUMBER(3)
 MAT                                       NOT NULL NUMBER(3)
 SCI                                       NOT NULL NUMBER(3)
 SOC                                       NOT NULL NUMBER(3)

QL>
```



```
<body>
<div>
<h1>Insert Records : </h1>
    <form action="InsertServlet" method="post" name="form1" class="form1">
        &emsp;&emsp;&emsp; &emsp;&emsp;&emsp;&emsp;
        <input type="text" name="rollno" placeholder="Enter Rollno"> &emsp;
        <input type="text" name="sname" placeholder="Enter Name"><br>
        <input type="text" name="tel" placeholder="Telugu Marks"> &emsp;
        <input type="text" name="hin" placeholder="Hindi Marks"> &emsp;
        <input type="text" name="eng" placeholder="English Marks"><br>
        <input type="text" name="mat" placeholder="Maths Marks"> &emsp;
        <input type="text" name="sci" placeholder="Science Marks"> &emsp;
        <input type="text" name="soc" placeholder="Social Marks"><br>
        &emsp;&emsp;&emsp;&emsp;&emsp;&emsp;&emsp;&emsp;
        <input type="submit" value="Insert Record" style="width:300px;"/>
    </form>
<h1>View Records : </h1>
    <form action="ViewServlet" method="get" name="form2" class="form2">
        <input type="text" name="rollno2" placeholder="Enter Rollno"><br>
        <input type="submit" value="View Result" style="width:170px;"/>
    </form>
</div>
</body>
</html>
```

View Servlet :

```
 import java.io.IOException;
import java.io.PrintWriter;
 import java.sql.Connection;
 import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
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("/ViewServlet")
public class ViewServlet extends HttpServlet
{
 private static final long serialVersionUID = 1L;
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
{
 int rollno = Integer.parseInt(request.getParameter("rollno2"));
try {
 Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","system","k1");
Statement st = con.createStatement();

String query = "select * from sscmarks where rollno="+rollno;
ResultSet rs = st.executeQuery(query); rs.next();
response.setContentType("text/html");

PrintWriter out = response.getWriter();
out.println(" Student Rollno : "+rs.getString(1)+"<br>");
out.println("Student Name : "+rs.getString(2)+"<br>");
out.println("Telugu : "+rs.getInt(3)+"<br>");
out.println("Hindi : "+rs.getInt(4)+"<br>");
out.println("English : "+rs.getInt(5)+"<br>");
out.println("Maths : "+rs.getInt(6)+"<br>");
out.println("Science : "+rs.getInt(7)+"<br>");
out.println("Social : "+rs.getInt(8)+"<br>");
int total = rs.getInt(3)+rs.getInt(4)+rs.getInt(5)+rs.getInt(6)+rs.getInt(7)+rs.getInt(8);
out.println("Total : "+total);
}
catch (Exception e)
 { e.printStackTrace(); }
 } }
```

getParameterNames() method

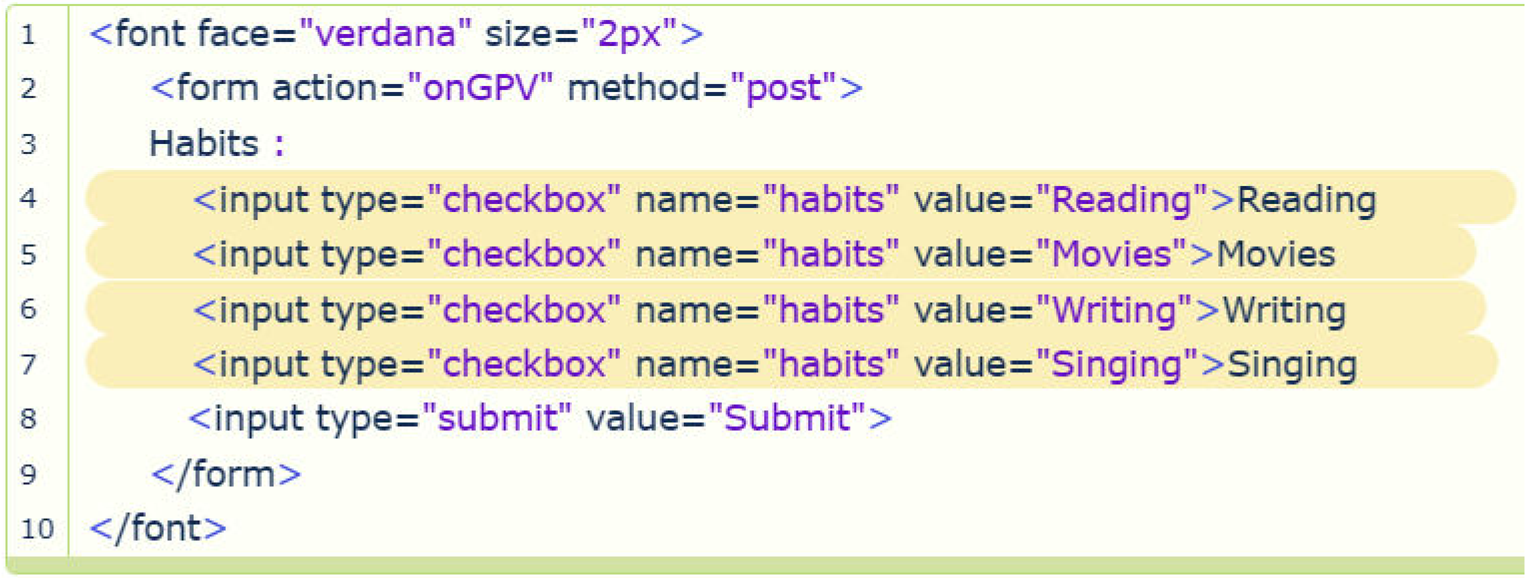
- With getParameter() we are able to find parameter values by passing parameter name.
- Parametername–n1
  Parameter value – val
- If a client send the data to the servlet, that data will be available in the object of HttpServletRequest interface.
- In case of getParameter() method we have to pass input parameter name and it will give the value.
- If you are not aware of input parameter name ? or if you have 50+ input values its really tedious to use getParameter() method.

```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Enumeration;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class OngetParameterNames extends HttpServlet
{
    protected void doPost(HttpServletRequest req,HttpServletResponse res)throws ServletException,IOException
        {
        PrintWriter pw=res.getWriter();
                res.setContentType("text/html");

                Enumeration en=req.getParameterNames();

                while(en.hasMoreElements())
                {
                        Object objOri=en.nextElement();
                        String param=(String)objOri;
                        String value=req.getParameter(param);
                        pw.println("Parameter Name is '"+param+"' and Parameter Value is '"+value+"'");
                }
                        pw.close();
        }
}
```

getParameterValues() method

- The method getParameterValues() will generally came into picture if there is a chance of getting multiple values for any input parameter, this method will retrieve all of it values and store as string array.
- String[] values = getParameterValues("Input Parameter");

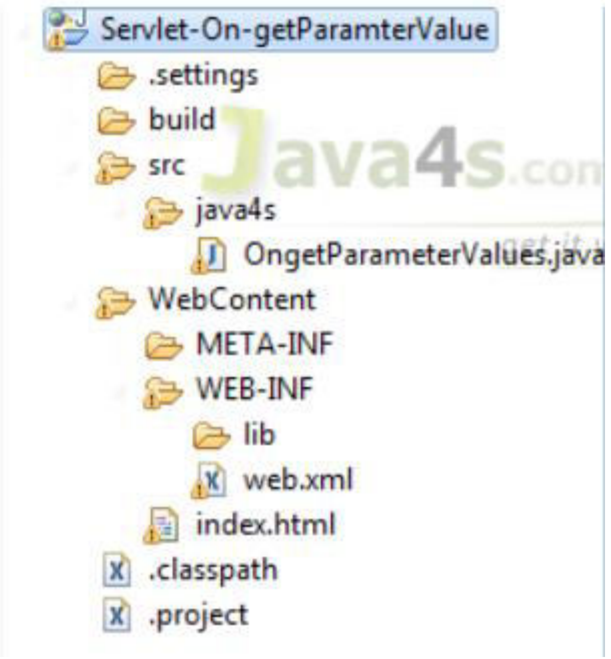
index.html



```
<font face="verdana" size="2px">
    <form action="onGPV" method="post">
    Habits :
       <input type="checkbox" name="habits" value="Reading">Reading
       <input type="checkbox" name="habits" value="Movies">Movies
       <input type="checkbox" name="habits" value="Writing">Writing
       <input type="checkbox" name="habits" value="Singing">Singing
       <input type="submit" value="Submit">
    </form>
</font>
```

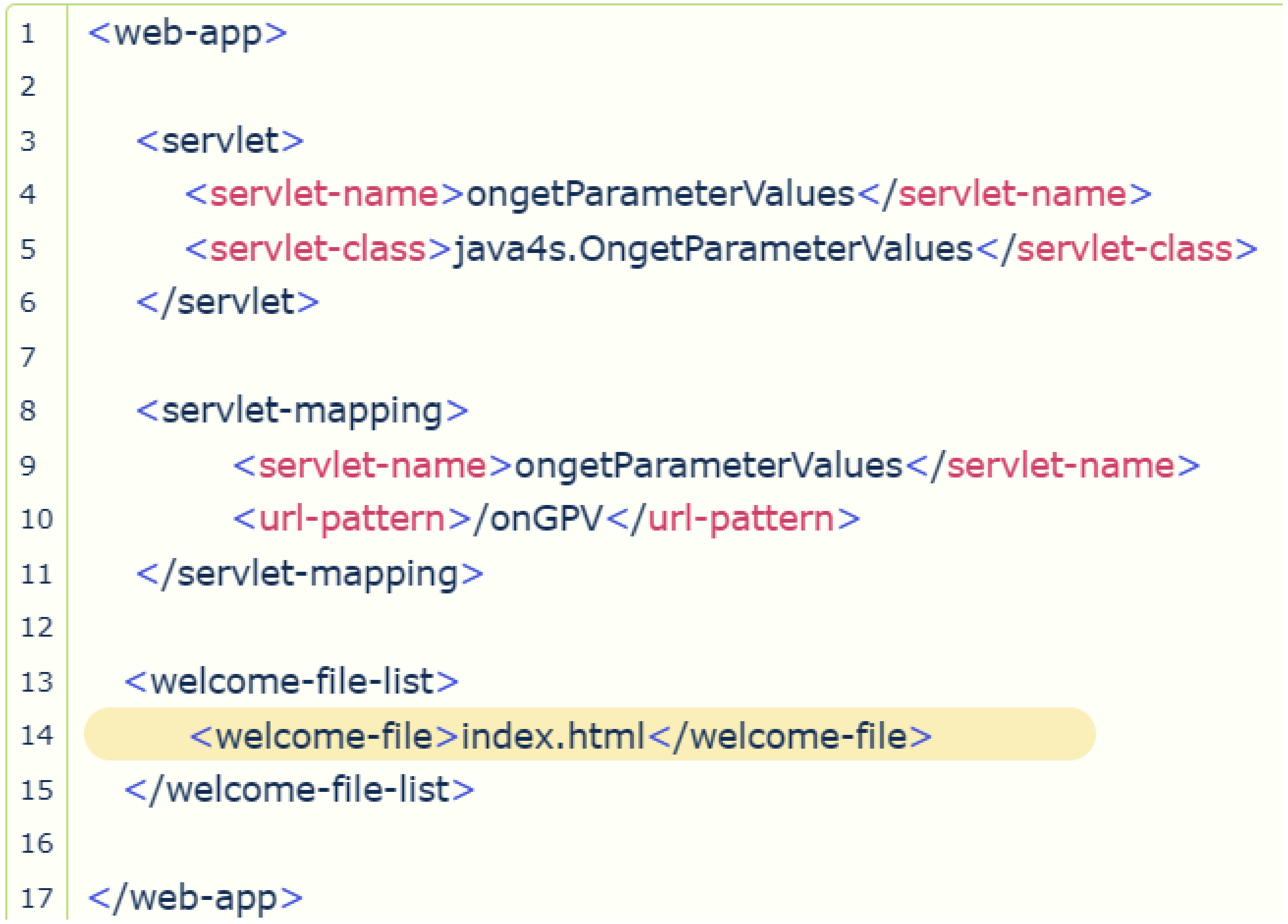
```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class OngetParameterValues extends HttpServlet
{
    protected void doPost(HttpServletRequest req,HttpServletResponse res)throws ServletException,IOExcep
    {
        PrintWriter pw=res.getWriter();
        res.setContentType("text/html");

        String[] values=req.getParameterValues("habits");
        pw.println("Selected Values...");
        for(int i=0;i<values.length;i++)
        {
           pw.println("<li>"+values[i]+"</li>");
        }
       pw.close();
    }
}
```

Directory Structure



```
Servlet-On-getParamterValue
    .settings
    build
    src
        java4s
            OngetParameterValues.java
    WebContent
        META-INF
        WEB-INF
            lib
            web.xml
        index.html
    .classpath
    .project
```

web.xml



```xml
<web-app>

  <servlet>
    <servlet-name>ongetParameterValues</servlet-name>
    <servlet-class>java4s.OngetParameterValues</servlet-class>
  </servlet>

  <servlet-mapping>
      <servlet-name>ongetParameterValues</servlet-name>
      <url-pattern>/onGPV</url-pattern>
  </servlet-mapping>

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

</web-app>
```

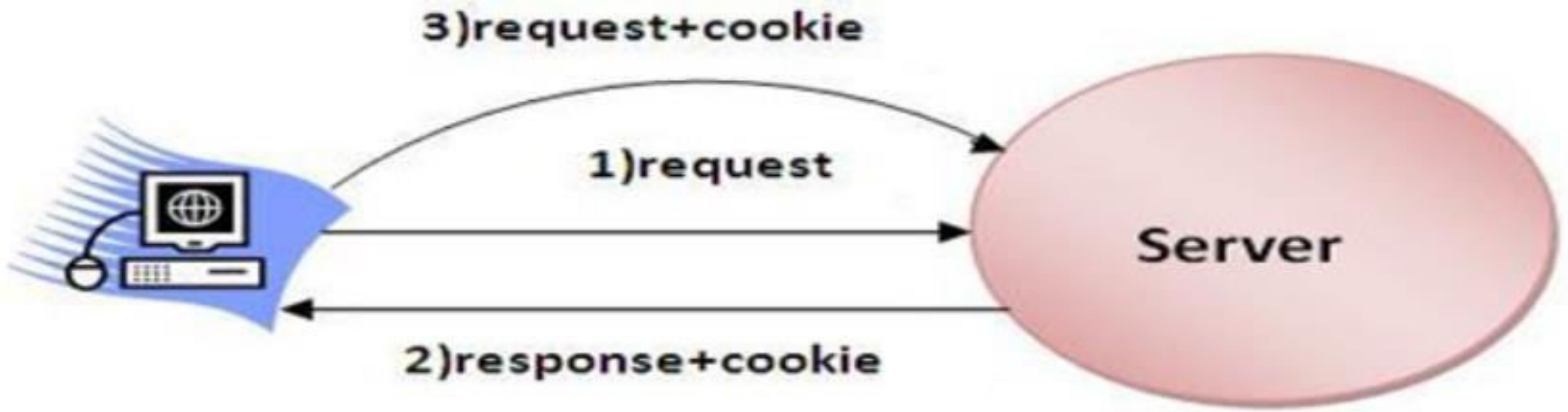
Cookies in Servlet

A **cookie** is a small piece of information that is persisted between the multiple client requests.

A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

How Cookie works

we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default.



Types of Cookie

- There are 2 types of cookies in servlets.
- Non-persistent cookie
- Persistent cookie

Non-persistent cookie

- It is **valid for single session** only. It is removed each time when user closes the browser.

Persistent cookie

- It is **valid for multiple session** . It is not removed each time when user closes the browser. It is removed only if user logout or sign-out.

Advantage of Cookies

- Simplest technique of maintaining the state.
- Cookies are maintained at client side.

Disadvantage of Cookies

- It will not work if cookie is disabled from the browser.
- Only textual information can be set in Cookie object.

- ## Cookie class

javax.servlet.http.Cookie class provides the functionality of using cookies. It provides a lot of useful methods for cookies.

Constructor of Cookie class

Constructor	**Description**
Cookie()	constructs a cookie.
Cookie(String name, String value)	constructs a cookie with a specified name and value.

Useful Methods of Cookie class

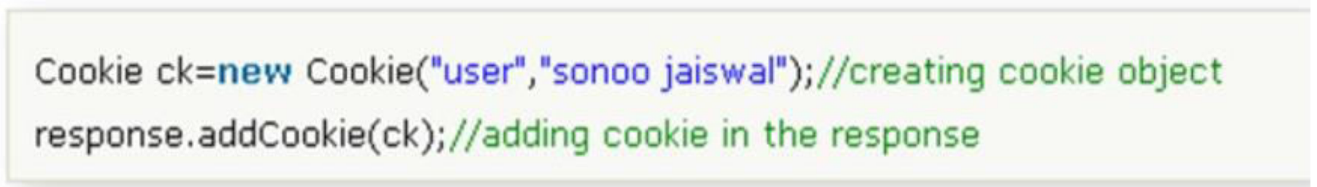
Method	**Description**
public void setMaxAge(int expiry)	Sets the maximum age of the cookie in seconds.
public String getName()	Returns the name of the cookie. The name cannot be changed after creation.
public String getValue()	Returns the value of the cookie.
public void setName(String name)	changes the name of the cookie.
public void setValue(String value)	changes the value of the cookie.

Other methods required for using Cookies

For adding cookie or getting the value from the cookie, we need some methods provided by other interfaces. They are:

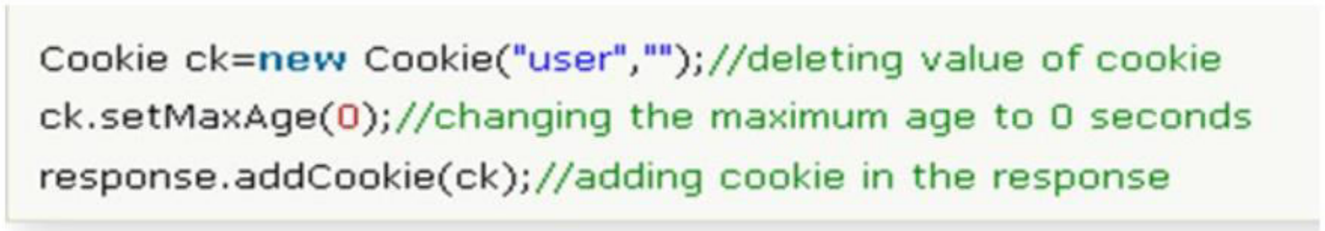
- **public void addCookie(Cookie ck):** method of HttpServletResponse interface is used to add cookie in response object.
- **public Cookie[] getCookies():** method of HttpServletRequest interface is used to return all the cookies from the browser.

- ## How to create Cookie?



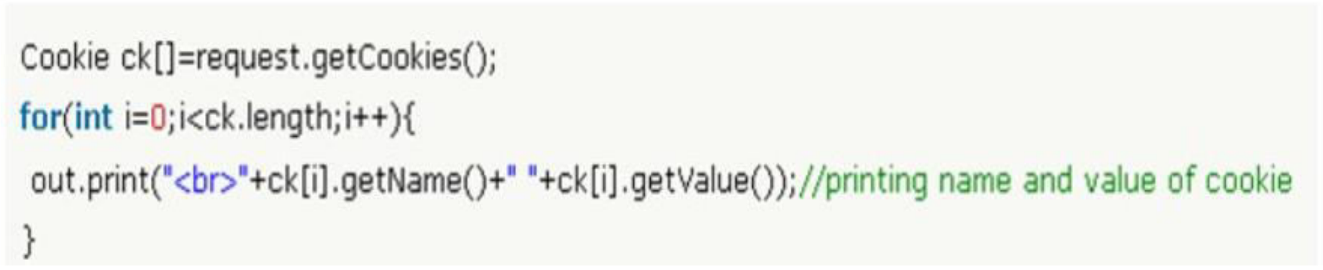
```
Cookie ck=new Cookie("user","sonoo jaiswal");//creating cookie object
response.addCookie(ck);//adding cookie in the response
```

- ## How to delete Cookie?

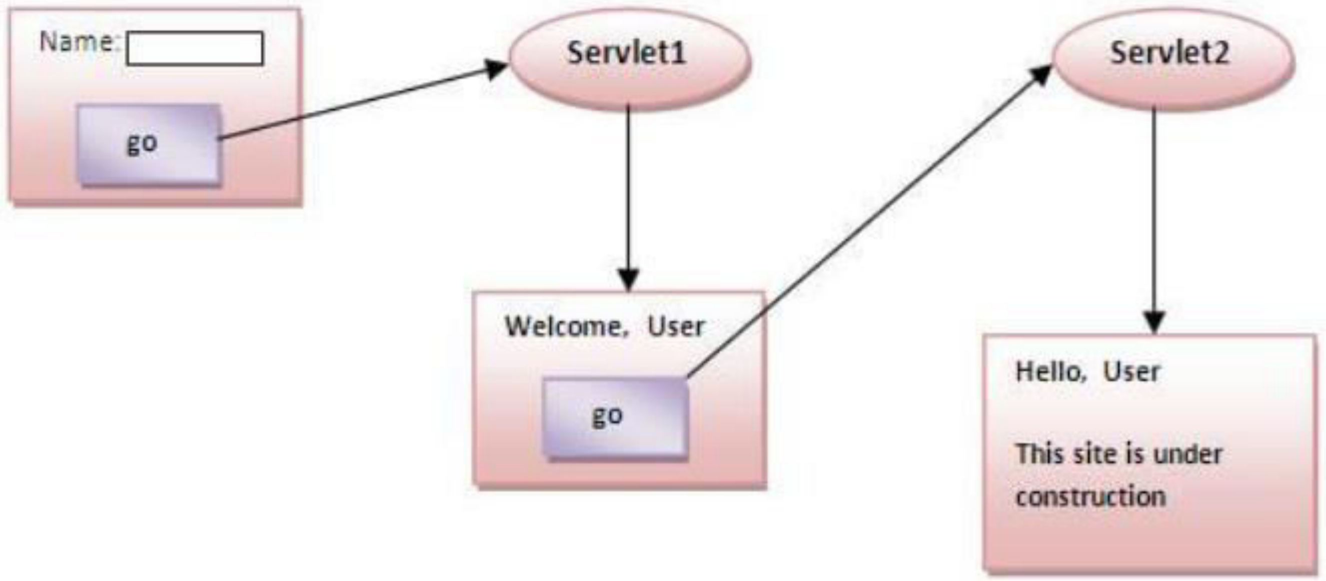


```
Cookie ck=new Cookie("user","");//deleting value of cookie
ck.setMaxAge(0);//changing the maximum age to 0 seconds
response.addCookie(ck);//adding cookie in the response
```

- ## How to get Cookies?



```
Cookie ck[]=request.getCookies();
for(int i=0;i<ck.length;i++){
 out.print("<br>"+ck[i].getName()+" "+ck[i].getValue());//printing name and value of cookie
}
```



index.html

```
<form action="servlet1" method="post">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class FirstServlet extends HttpServlet {

 public void doPost(HttpServletRequest request, HttpServletResponse response){
  try{

  response.setContentType("text/html");
  PrintWriter out = response.getWriter();

  String n=request.getParameter("userName");
  out.print("Welcome "+n);

  Cookie ck=new Cookie("uname",n);//creating cookie object
  response.addCookie(ck);//adding cookie in the response

  //creating submit button
  out.print("<form action='servlet2'>");
  out.print("<input type='submit' value='go'>");
  out.print("</form>");
```

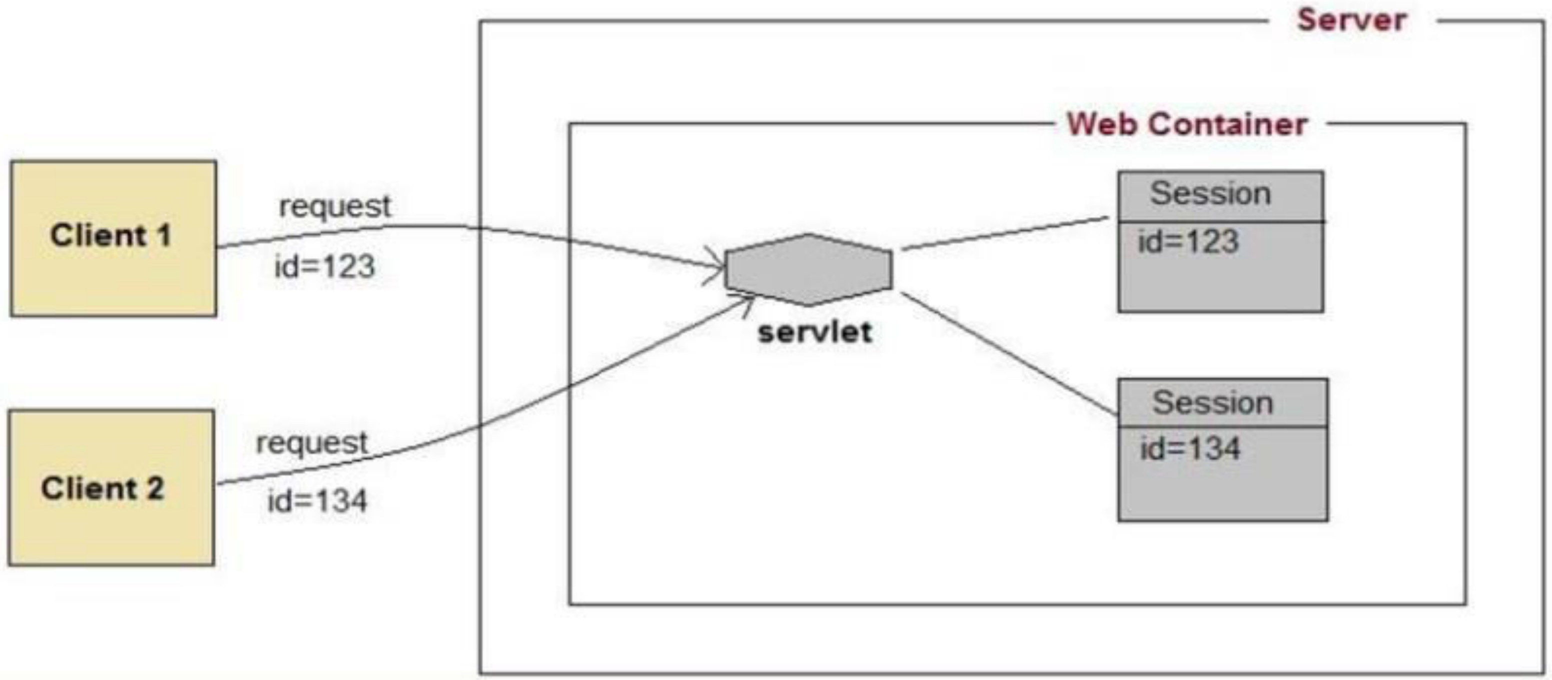
SecondServlet.java

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

public class SecondServlet extends HttpServlet {

public void doPost(HttpServletRequest request, HttpServletResponse response){
   try{

   response.setContentType("text/html");
   PrintWriter out = response.getWriter();

   Cookie ck[]=request.getCookies();
   out.print("Hello "+ck[0].getValue());

   out.close();

      }catch(Exception e){System.out.println(e);}
   }
```

```
<web-app>

<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>
```

Session Tracking in Servlets

- **Session** simply means a particular interval of time.
- **Session Tracking** is a way to maintain state (data) of an user. It is also known as **session management** in servlet.
- Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.
- HTTP is stateless that means each request is considered as the new request.



- whenever a user starts using our application, we can save a unique identification information about him, in an object which is available throughout the application, until its destroyed.
- So wherever the user goes, we will always have his information and we can always manage which user is doing what. Whenever a user wants to exit from your application, destroy the object with his information.

Creating a new session

getSession() method returns a session. If the session already exist, it return the esisting session else create a new sesion

```
HttpSession session = request.getSession();
```

```
HttpSession session = request.getSession(true);
```

getSession(true) always return a new session

Getting a pre-existing session

```
HttpSession session = request.getSession(false);
```

return a pre-existing session

Destroying a session

```
session.invalidate();
```

destroy a session

```html
<form method="post"
action="Validate">
 User: <input type="text"
name="user" /><br/>
 Password: <input type="text"
name="pass" ><br/>
 <input type="submit"
value="submit">
</form>
```

```xml
<web-app..>
   <servlet>
      <servlet-name>Validate</servlet-name>
      <servlet-class>Validate</servlet-class>
   </servlet>
   <servlet>
      <servlet-name>Welcome</servlet-name>
      <servlet-class>Welcome</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>Validate</servlet-name>
      <url-pattern>/Validate</url-pattern>
   </servlet-mapping>
   <servlet-mapping>
      <servlet-name>Welcome</servlet-name>
      <url-pattern>/Welcome</url-pattern>
   </servlet-mapping>
   <welcome-file-list>
      <welcome-file>index.html</welcome-file>
   </welcome-file-list>
</web-app>
```

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

public class Validate extends HttpServlet {

   protected void doPost(HttpServletRequest request, HttpServletResponse
response)
         throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");

      String name = request.getParameter("user");
      String pass = request.getParameter("pass");
      if(pass.equals("1234"))
      {
         //creating a session
         HttpSession session = request.getSession();
         session.setAttribute("user", name);
         response.sendRedirect("Welcome");
      }
   }
}
```

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

public class Welcome extends HttpServlet {

   protected void doGet(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        HttpSession session = request.getSession();
        String user = (String)session.getAttribute("user");
        out.println("Hello "+user);
    }
}
```

File: index.html

```html
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Servlet Login Example</title>
</head>
<body>

<h1>Login App using HttpSession</h1>
<a href="login.html">Login</a>|
<a href="LogoutServlet">Logout</a>|
<a href="ProfileServlet">Profile</a>

</body>
</html>
```

File: login.html

```html
<form action="LoginServlet" method="post">
Name:<input type="text" name="name"><br>
Password:<input type="password" name="password"><br>
<input type="submit" value="login">
</form>
```

File: link.html

```html
<a href="login.html">Login</a> |
<a href="LogoutServlet">Logout</a> |
<a href="ProfileServlet">Profile</a>
<hr>
```

File: LoginServlet.java

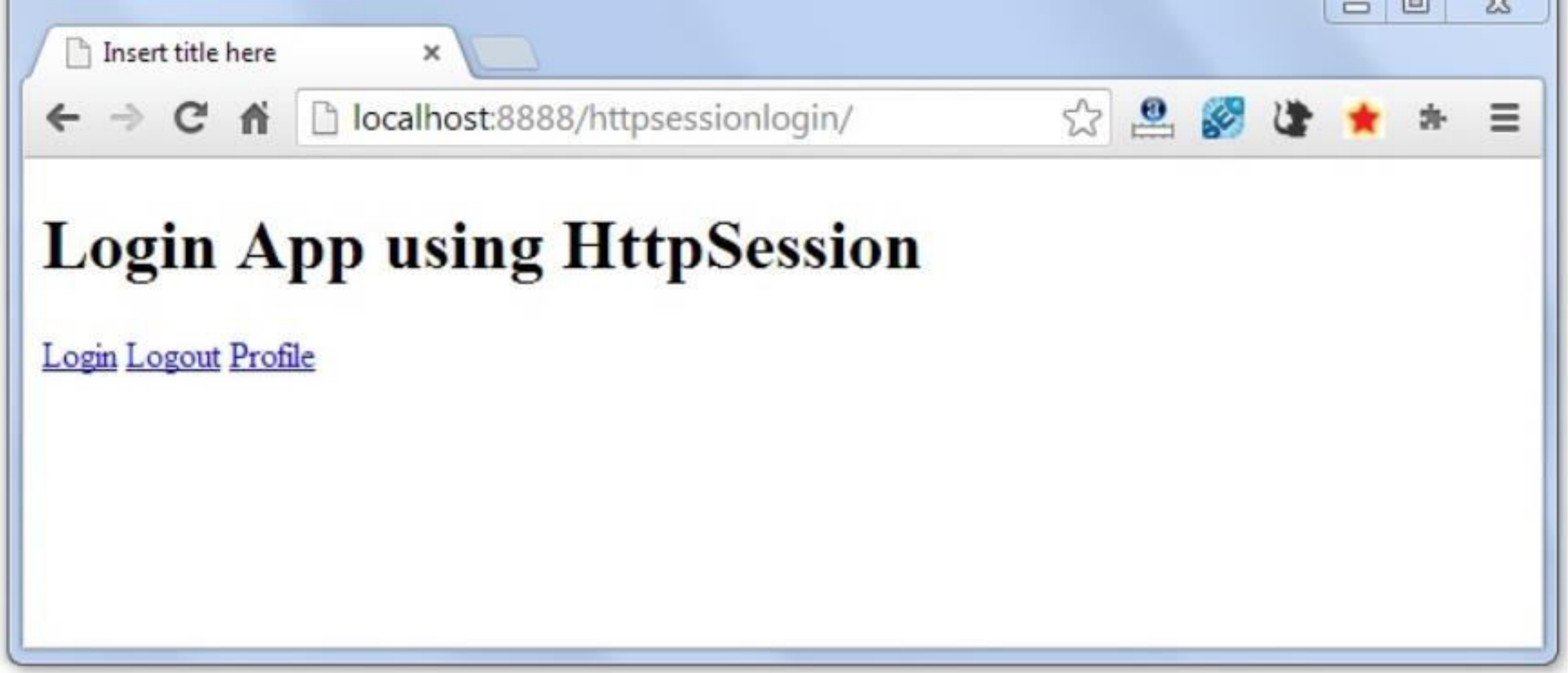
```java
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
public class LoginServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
                    throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();
        request.getRequestDispatcher("link.html").include(request, response);
        String name=request.getParameter("name");
        String password=request.getParameter("password");
        if(password.equals("admin123")){
        out.print("Welcome, "+name);
        HttpSession session=request.getSession();
        session.setAttribute("name",name);
        }
        else{
            out.print("Sorry, username or password error!");
            request.getRequestDispatcher("login.html").include(request, response);
        }
        out.close();
    }
}
```

File: LogoutServlet.java

```java
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
public class LogoutServlet extends HttpServlet {
        protected void doGet(HttpServletRequest request, HttpServletResponse response)
                                throws ServletException, IOException {
            response.setContentType("text/html");
            PrintWriter out=response.getWriter();

            request.getRequestDispatcher("link.html").include(request, response);

            HttpSession session=request.getSession();
            session.invalidate();

            out.print("You are successfully logged out!");

            out.close();
    }
```

File: ProfileServlet.java

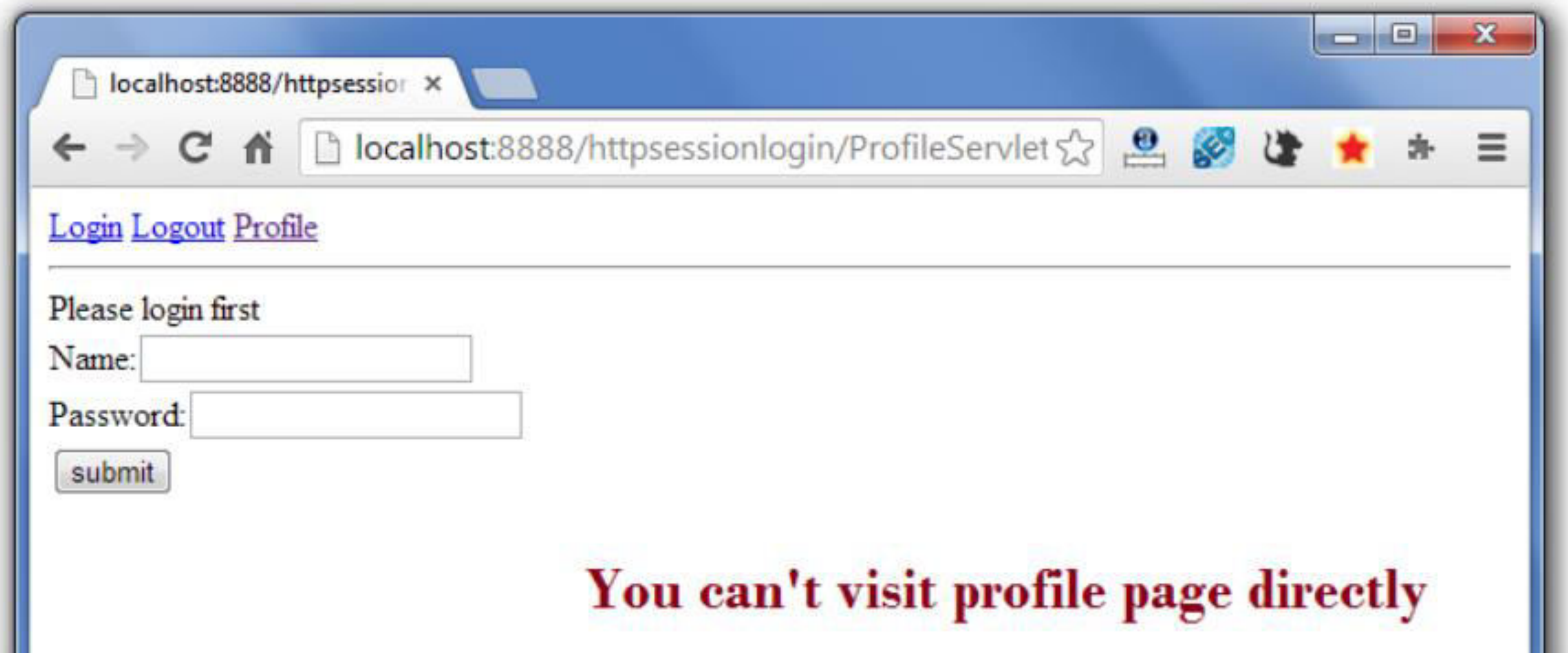
```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
public class ProfileServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)  throws ServletException, I
OException {               response.setContentType("text/html");
        PrintWriter out=response.getWriter();
        request.getRequestDispatcher("link.html").include(request, response);
        HttpSession session=request.getSession(false);
        if(session!=null){
        String name=(String)session.getAttribute("name");
        out.print("Hello, "+name+" Welcome to Profile");
        }
        else{
            out.print("Please login first");
            request.getRequestDispatcher("login.html").include(request, response);
        }
        out.close();
    }
}
```


Insert title here

localhost:8888/httpsessionlogin/

Login App using HttpSession

Login Logout Profile


localhost:8888/httpsessio

localhost:8888/httpsessionlogin/ProfileServlet

Login Logout Profile

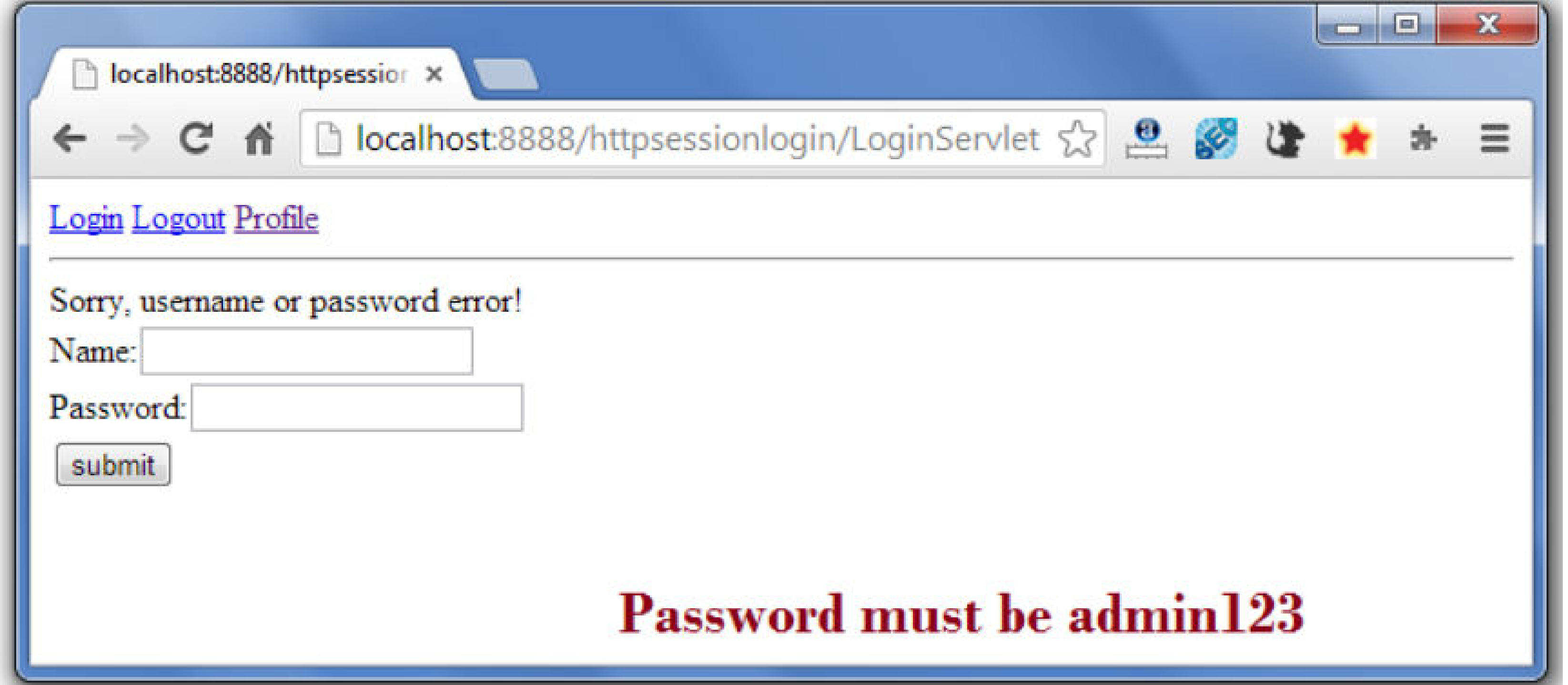
Please login first

Name:

Password:

submit

You can't visit profile page directly



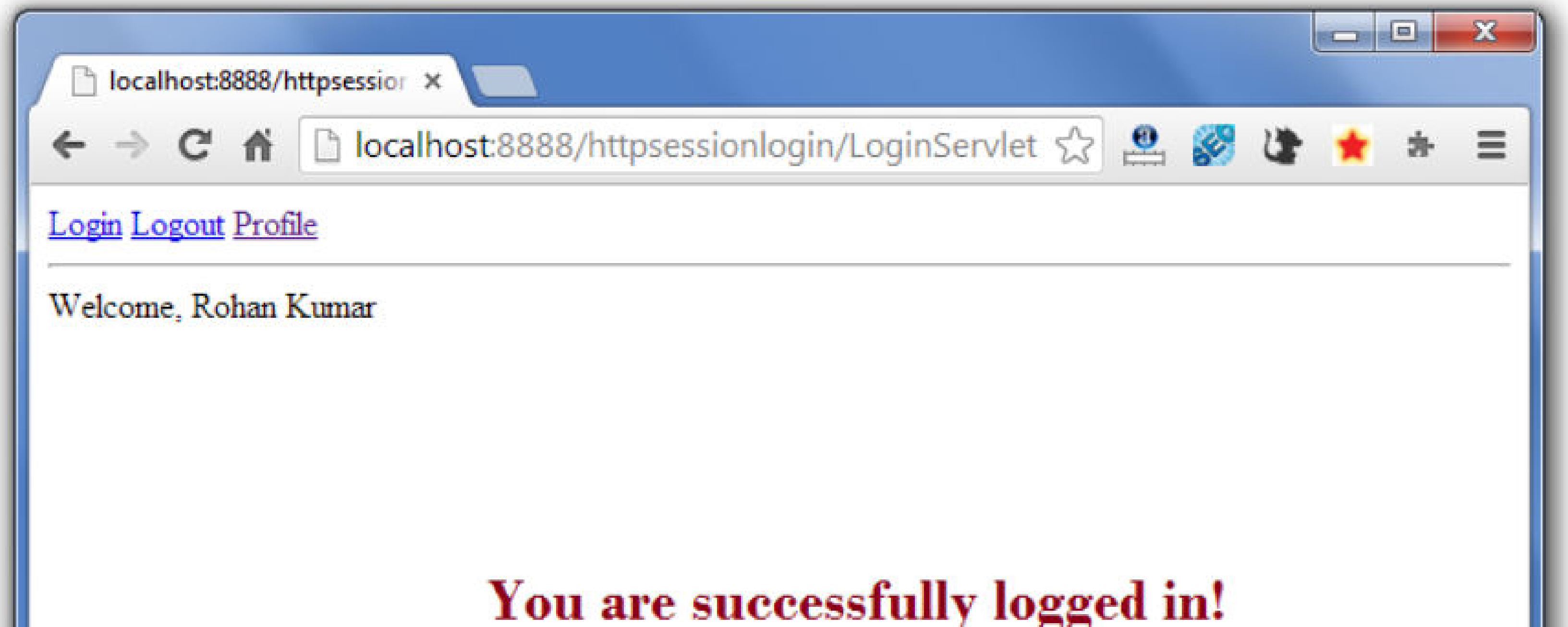
localhost:8888/httpsessior
localhost:8888/httpsessionlogin/LoginServlet
Login Logout Profile

Sorry, username or password error!

Name:

Password:

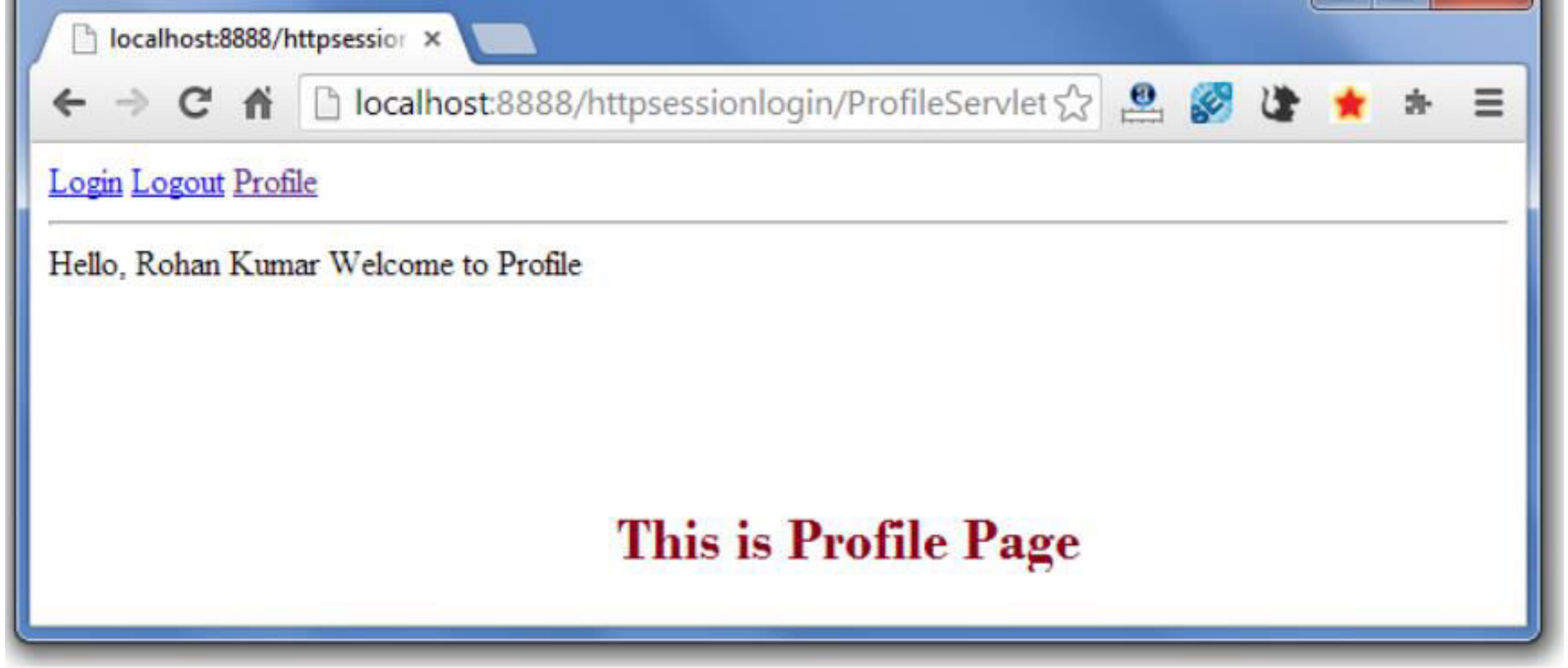
Password must be admin123



localhost:8888/httpsessior
localhost:8888/httpsessionlogin/LoginServlet
Login Logout Profile

Welcome, Rohan Kumar

You are successfully logged in!

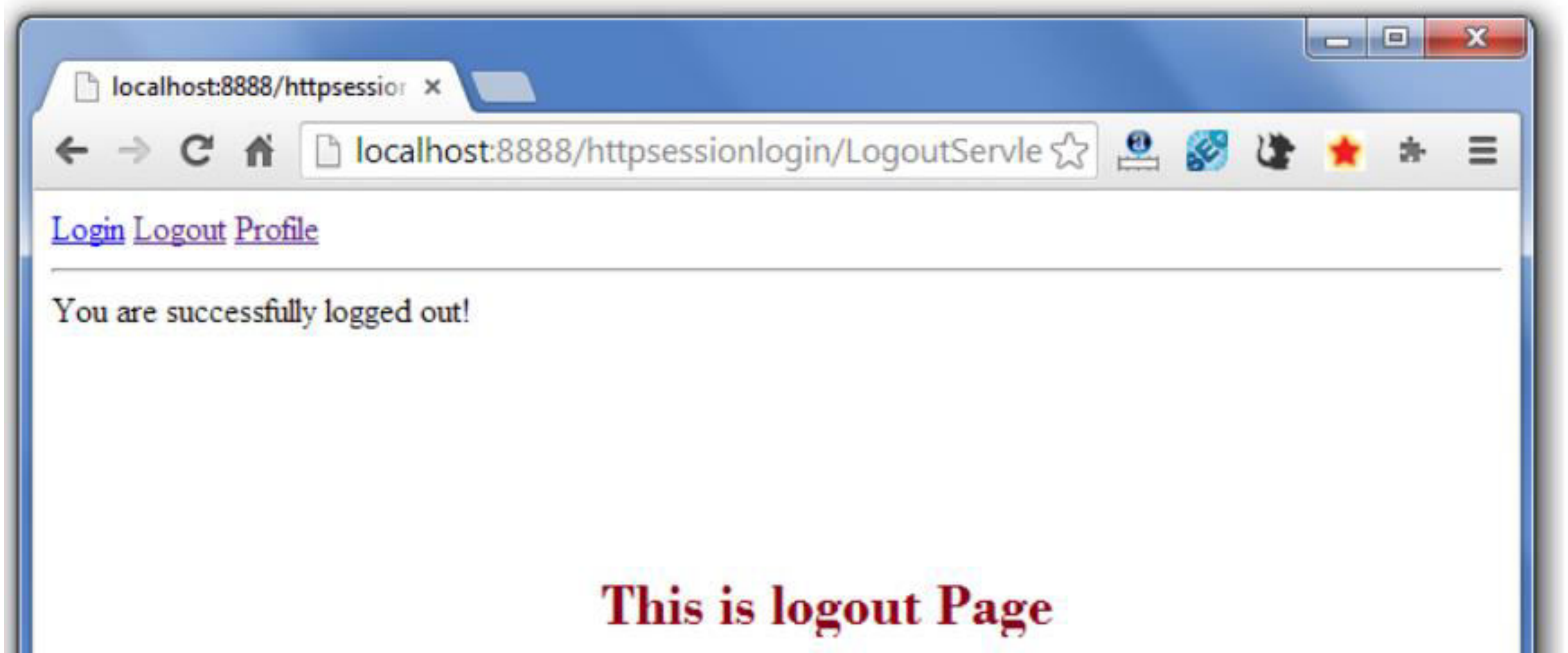


localhost:8888/httpsessionlogin/ProfileServlet

Login Logout Profile

Hello, Rohan Kumar Welcome to Profile

This is Profile Page



localhost:8888/httpsessionlogin/LogoutServle

Login Logout Profile

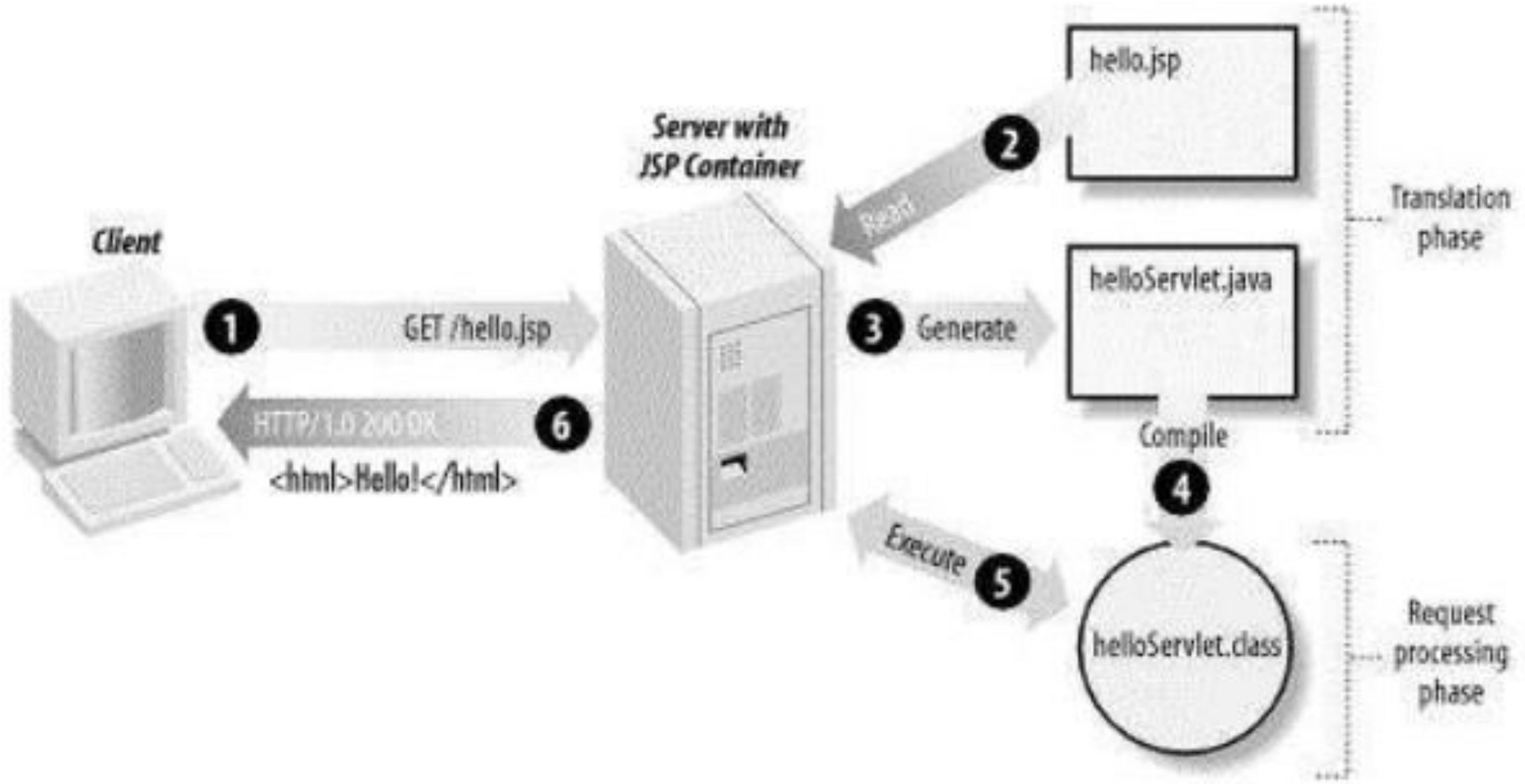
You are successfully logged out!

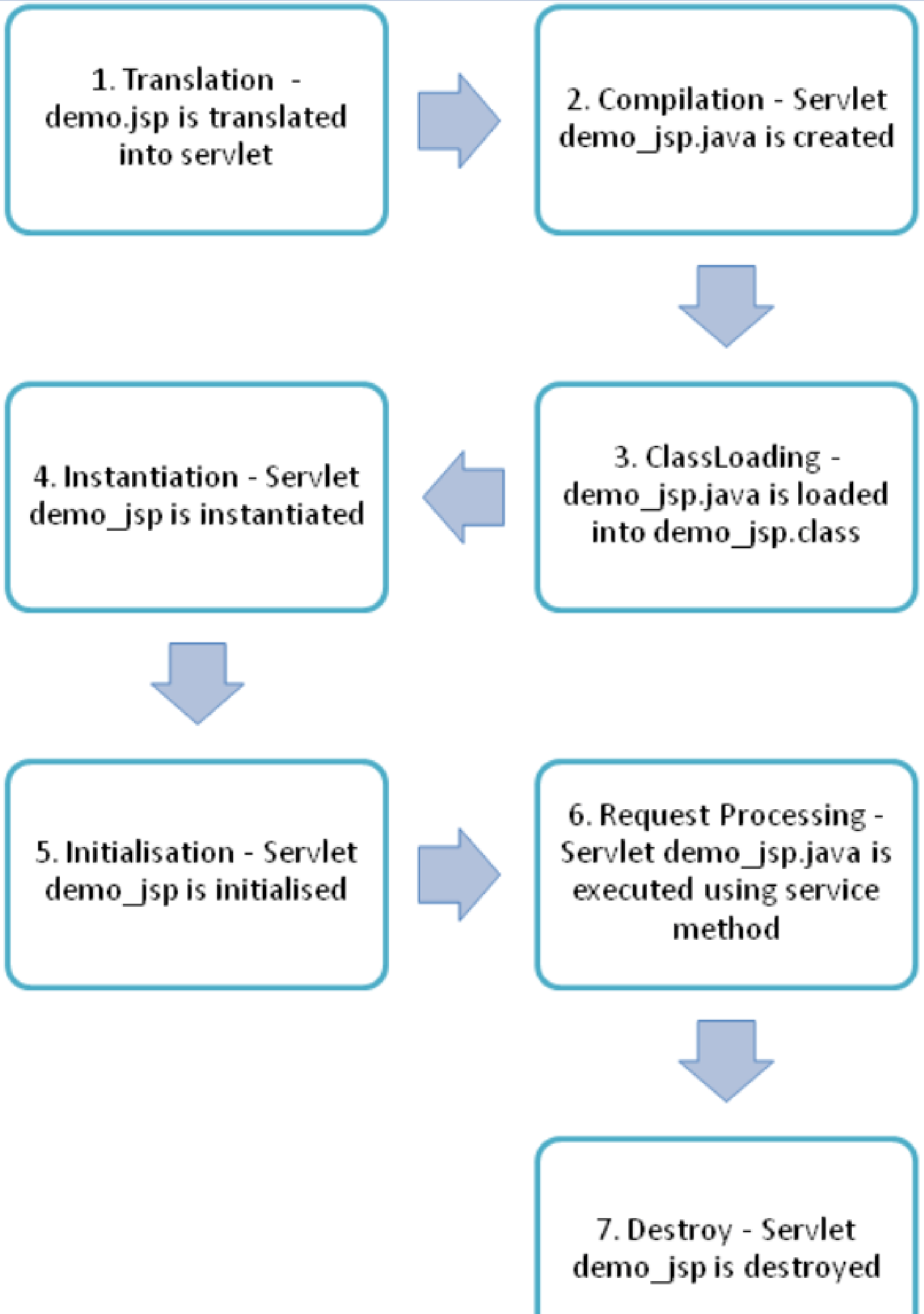
This is logout Page

JSP Processing

The following steps explain how the web server creates the Webpage using JSP −

- As with a normal page, your browser sends an HTTP request to the web server.
- The web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine. This is done by using the URL or JSP page which ends with **.jsp** instead of **.html**.
- The JSP engine loads the JSP page from disk and converts it into a servlet content. This conversion is very simple in which all template text is converted to println( ) statements and all JSP elements are converted to Java code. This code implements the corresponding dynamic behavior of the page.
- The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.
- A part of the web server called the servlet engine loads the Servlet class and executes it. During execution, the servlet produces an output in HTML format. The output is furthur passed on to the web server by the servlet engine inside an HTTP response.
- The web server forwards the HTTP response to your browser in terms of static HTML content.
- Finally, the web browser handles the dynamically-generated HTML page inside the HTTP response exactly as if it were a static page.


Client
Server with JSP Container
1
GET /hello.jsp
6
HTTP/1.0 200 OK
<html>Hello!</html>
2
Read
hello.jsp
3
Generate
helloServlet.java
Compile
4
Execute
5
helloServlet.class
Translation phase
Request processing phase


1. Translation - demo.jsp is translated into servlet
2. Compilation - Servlet demo_jsp.java is created
3. ClassLoading - demo_jsp.java is loaded into demo_jsp.class
4. Instantiation - Servlet demo_jsp is instantiated
5. Initialisation - Servlet demo_jsp is initialised
6. Request Processing - Servlet demo_jsp.java is executed using service method
7. Destroy - Servlet demo_jsp is destroyed

1. **Translation of the JSP Page:**

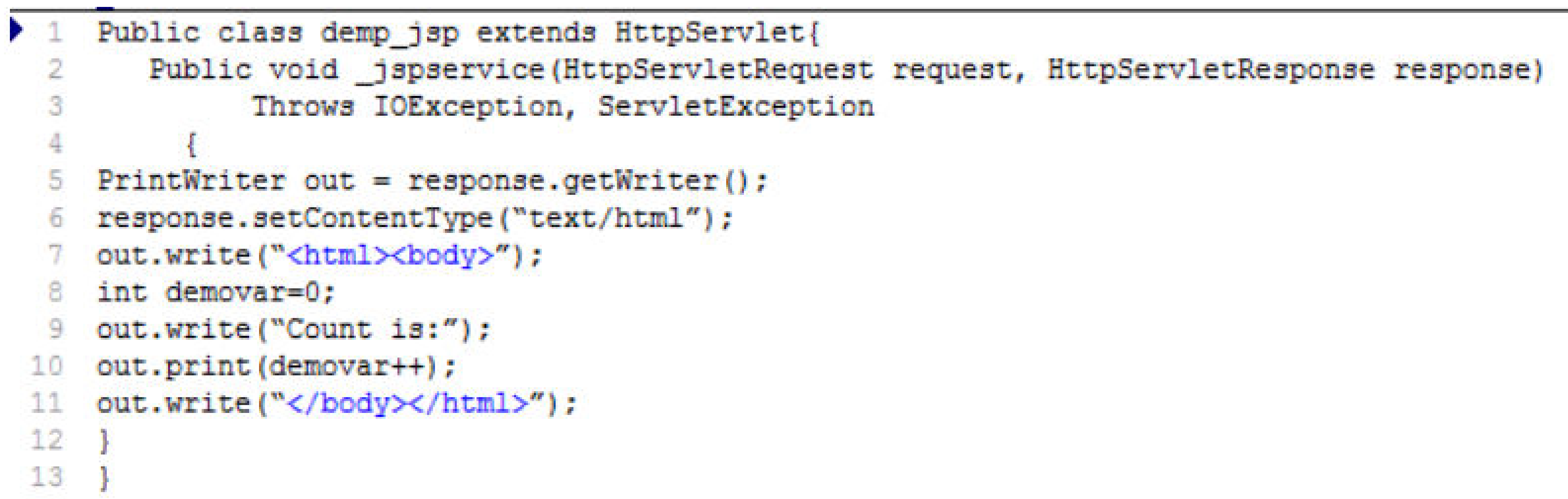
A Java servlet file is generated from a JSP source file. This is the first step of JSP life cycle. In translation phase, container validates the syntactic correctness of JSP page and tag files.

- The JSP container interprets the standard directives and actions, and the custom actions referencing tag libraries (they are all part of JSP page and will be discussed in the later section) used in this JSP page.
- In the above pictorial description, demo.jsp is translated to demo_jsp.java in the first step
- Let's take an example of "demo.jsp" as shown below:

Demo.jsp

```
<html>
<head>
<title>Demo JSP</title>
</head>
<%
int demvar=0;%>
<body>
Count is:
<% Out.println(demovar++); %>
<body>
</html>
```

Demo JSP Page is converted into demo_jsp servlet in the below code.



```
Public class demp_jsp extends HttpServlet{
  Public void _jspservice(HttpServletRequest request, HttpServletResponse response)
        Throws IOException, ServletException
     {
PrintWriter out = response.getWriter();
response.setContentType("text/html");
out.write("<html><body>");
int demovar=0;
out.write("Count is:");
out.print(demovar++);
out.write("</body></html>");
}
}
```

2. **Compilation of the JSP Page**

- The generated java servlet file is compiled into java servlet class
- The translation of java source page to its implementation class can happen at any time between the deployment of JSP page into the container and processing of the JSP page.
- In the above pictorial description demo_jsp.java is compiled to a class file demo_jsp.class

3. **Classloading**

- Servlet class that has been loaded from JSP source is now loaded into the container

4. **Instantiation**

- In this step the object i.e. the instance of the class is generated.
- The container manages one or more instances of this class in the response to requests and other events. Typically, a JSP container is built using a servlet container. A JSP container is an extension of servlet container as both the container support JSP and servlet.
- A JSPPage interface which is provided by container provides init() and destroy () methods.
- There is an interface HttpJSPPage which serves HTTP requests, and it also contains the service method.

5. **Initialization**

```
public void jspInit()
{
        //initializing the code
}
```

- _jspinit() method will initiate the servlet instance which was generated from JSP and will be invoked by the container in this phase.
- Once the instance gets created, init method will be invoked immediately after that
- It is only called once during a JSP life cycle, the method for initialization is declared as shown above

6. **Request processing**

```
void _jspservice(HttpServletRequest request HttpServletResponse response)
{
        //handling all request and responses
}
```

- _jspservice() method is invoked by the container for all the requests raised by the JSP page during its life cycle
- For this phase, it has to go through all the above phases and then only service method can be invoked.
- It passes request and response objects
- This method cannot be overridden
- The method is shown above: It is responsible for generating of all HTTP methods i.eGET, POST, etc.

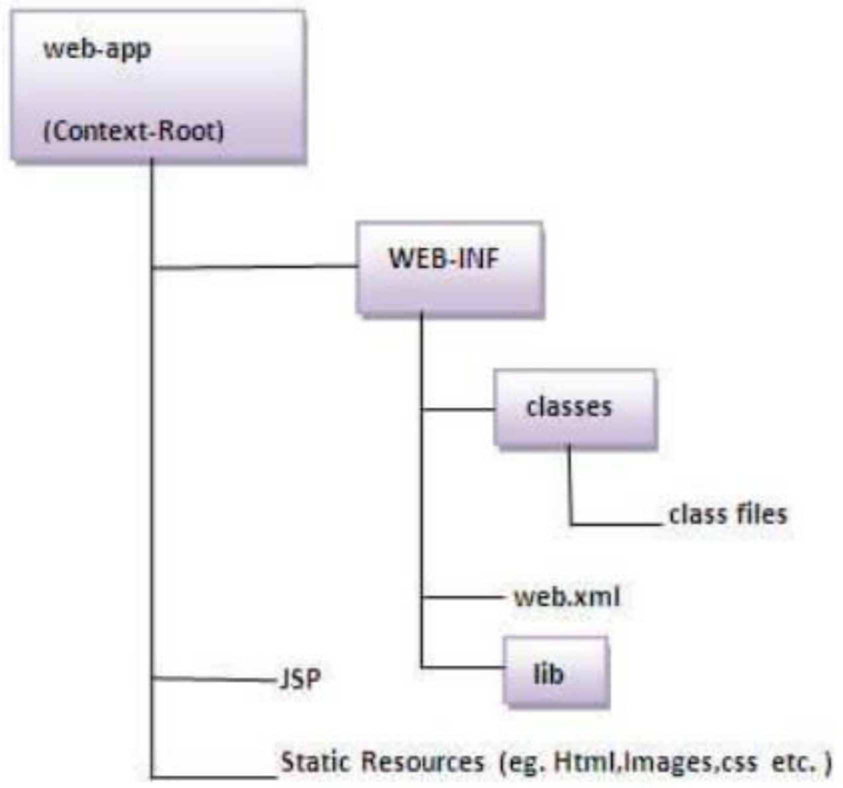
7. **Destroy**

```
public void _jspdestroy()
{
            //all clean up code
}
```

- _jspdestroy() method is also invoked by the container
- This method is called when container decides it no longer needs the servlet instance to service requests.
- When the call to destroy method is made then, the servlet is ready for a garbage collection
- This is the end of the life cycle.

The Directory structure of JSP

The directory structure of JSP page is same as Servlet. We contain the JSP page outside the WEB-INF folder or in any directory.



JSP Declaration Tag

The **JSP declaration tag** is used *to declare fields and methods.*

The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet.

So it doesn't get memory at each request.

```
<%!  field or method declaration %>
```

Difference between JSP Scriptlet tag and Declaration tag

Jsp Scriptlet Tag	Jsp Declaration Tag
The jsp scriptlet tag can only declare variables not methods.	The jsp declaration tag can declare variables as well as methods.
The declaration of scriptlet tag is placed inside the _jspService() method.	The declaration of jsp declaration tag is placed outside the _jspService() method.

Example of JSP declaration tag that declares field

index.jsp

```
<html>
<body>
<%! int data=50; %>
<%= "Value of the variable is:"+data %>
</body>
</html>
```

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Guru Declaration Tag</title>
</head>
<body>
<%! int count =10; %>
<% out.println("The Number is " +count); %>
</body>
</html>
```

Example of JSP declaration tag that declares method

```
<html>
<body>
<%!
int cube(int n){
return n*n*n*;
}
%>
<%= "Cube of 3 is:"+cube(3) %>
</body>
</html>
```

JSP Scriptlet

- Scriptlet tag allows to write Java code into JSP file.
- JSP container moves statements in _jspservice() method while generating servlet from jsp.
- For each request of the client, service method of the JSP gets invoked hence the code inside the Scriptlet executes for every request.
- A Scriptlet contains java code that is executed every time JSP is invoked.

Syntax of Scriptlet tag:

```
<% java code %>
```

```
</head>
<body>
<% int num1=10;
   int num2=40;
   int num3 = num1+num2;
   out.println("Scriplet Number is " +num3);
%>
</body>
</html>
```

```
<html>
<body>
<% out.print("welcome to jsp"); %>
</body>
</html>
```

File: index.html

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

File: welcome.jsp

```
<html>
<body>
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
</form>
</body>
</html>
```

JSP expression tag

The code placed within **JSP expression tag** is *written to the output stream of the response*. So you need not write out.print() to write data. It is mainly used to print the values of variable or method.

Syntax of JSP expression tag

```
<%=  statement %>
```

Example of JSP expression tag

In this example of jsp expression tag, we are simply displaying a welcome message.

```
<html>
<body>
<%= "welcome to jsp" %>
</body>
</html>
```

Note: Do not end your statement with semicolon in case of expression tag.

Example of JSP expression tag that prints current time

To display the current time, we have used the getTime() method of Calendar class. The getTime() is an instance method of Calendar class, so we have called it after getting the instance of Calendar class by the getInstance() method.

```
<html>
<body>
Current Time: <%= java.util.Calendar.getInstance().getTime() %>
</body>
</html>
```

```
<body>
<% out.println("The expression number is "); %>
<% int num1=10; int num2=10; int num3 = 20; %>
<%= num1*num2+num3 %>
</body>
```

Example of JSP expression tag that prints the user name

In this example, we are printing the username using the expression tag. The index.html file gets the username and sends the request to the welcome.jsp file, which displays the username.

File: index.jsp

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname"><br/>
<input type="submit" value="go">
</form>
</body>
</html>
```

File: welcome.jsp

```
<html>
<body>
<%= "Welcome "+request.getParameter("uname") %>
</body>
</html>
```

JSP Comments

Comments are the one when JSP container wants to ignore certain texts and statements.

When we want to hide certain content, then we can add that to the comments section.

Syntax:

```
<% -- JSP Comments %>
```

T his tags are used to comment in JSP and ignored by the JSP container.

```
<body>
<%-- Guru Comments section --%>
<% out.println("This is comments example"); %>

</body>
```

JSP Tags

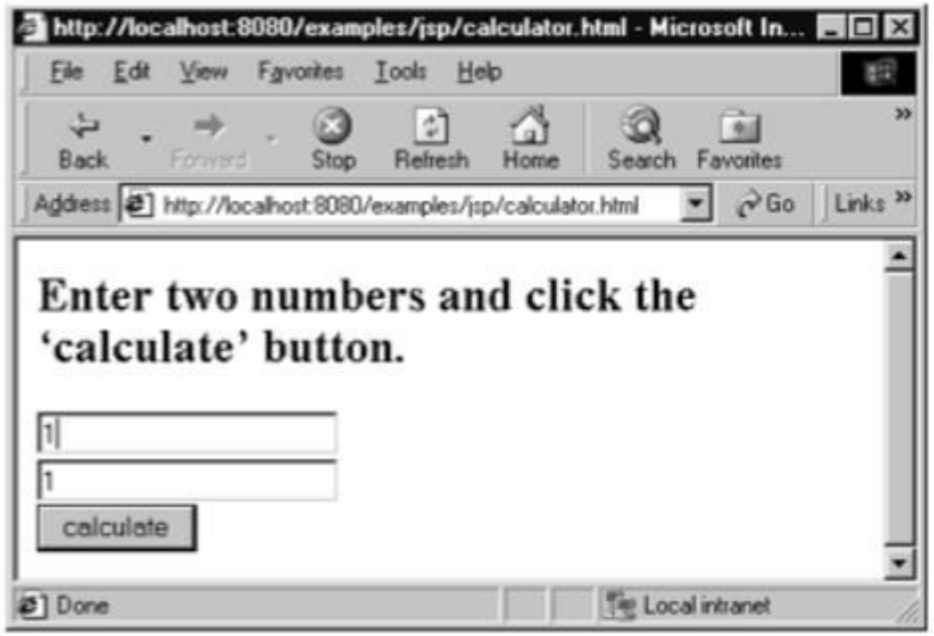
- Comments <%-- …...text…... --%>
- Declaration <%! int i; %>

  <%! int numOfStudents(arg1,..) {} %>
- Expression <%= 1+1 %>
- Scriptlets <% … java code … %>
- include file <%@ include file="*.jsp" %>
- …...

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Guru JSP Example</title>
</head>
<body>

<%-- This is a JSP example with scriplets, comments , expressions --%>
<% out.println("This is guru JSP Example"); %>
<% out.println("The number is "); %>
<%! int num12 = 12; int num32 = 12; %>
<%= num12*num32 %>
Today's date: <%= (new java.util.Date()).toLocaleString()%>
</body>
</html>
```

A First JSP

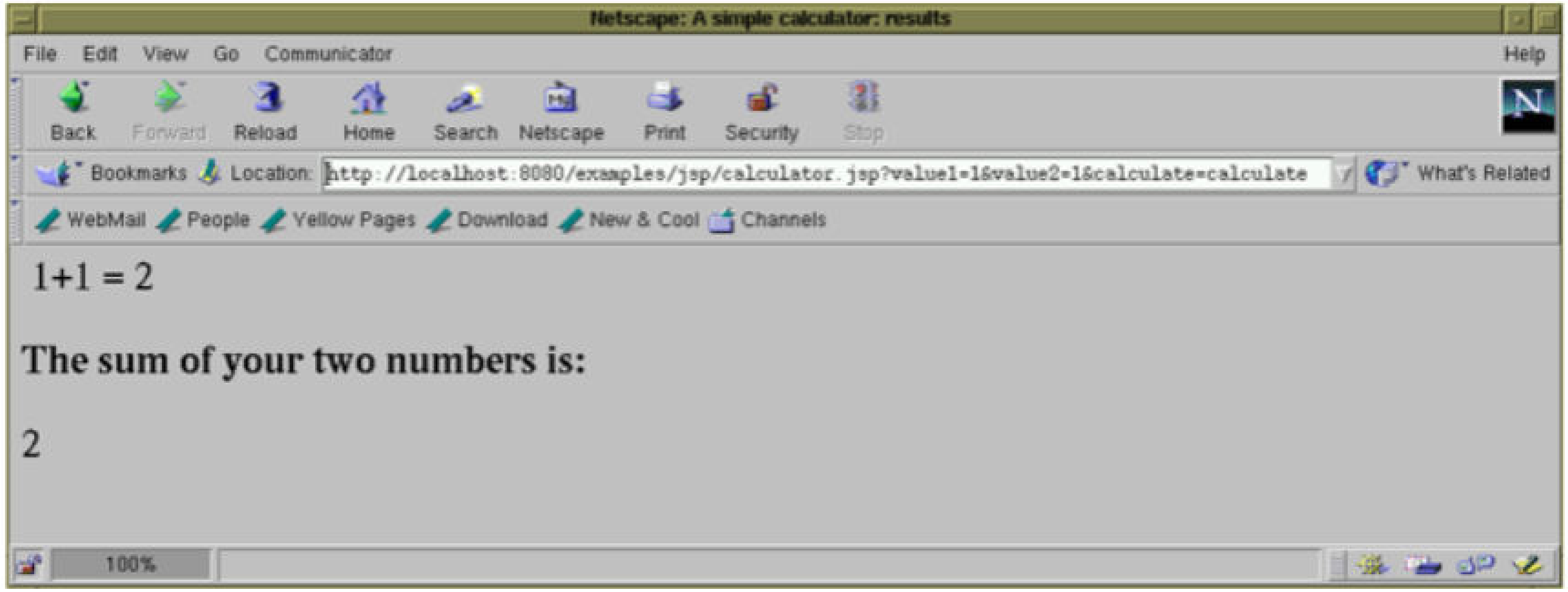
```
<html>
<head></head>
<body>
<p>Enter two numbers and click the
'calculate' button.</p>

<form action="calculator.jsp" method="get">
<input type=text name=value1><br>
<input type=text name=value2 ><br>
<input type=submit name=calculate value=calculate>
</form>
</body>
</html>
```


http://localhost:8080/examples/jsp/calculator.html - Microsoft In...

Enter two numbers and click the 'calculate' button.

File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address http://localhost:8080/examples/jsp/calculator.html Go Links
1
1
calculate
Done
Local intranet

Calculator.html


Netscape: A simple calculator: results
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://localhost:8080/examples/jsp/calculator.jsp?value1=1&value2=1&calculate=calculate
What's Related
WebMail People Yellow Pages Download New & Cool Channels
100%

1+1 = 2

The sum of your two numbers is:

2

```
<html>
<head><title>A simple calculator: results</title></head>
<body>
<%-- A simpler example 1+1=2 --%>
1+1 = <%= 1+1 %>
<%-- A simple calculator --%>
<h2>The sum of your two numbers is:</h2>
<%= Integer.parseInt(request.getParameter("value1")) +
    Integer.parseInt(request.getParameter("value2")) %>
</body>
</html>
```

Calculator.jsp

Directives

- Directives supply directions and messages to a JSP container.
- The directives provide global information about the entire page of JSP. Hence, they are an essential part of the JSP code.
- The **jsp directives** are messages that tells the web container how to translate a JSP page into the corresponding servlet.
- Directives can contain several attributes that are separated by a comma and act as key-value pairs.
- In JSP, directives are described with a pair of <%@ .... %> tags.
- Syntax: `<%@ directive attribute="value" %>`
- There are three types of directives:
  - page directive
  - include directive
  - taglib directive

JSP Page directive

Syntax of Page directive:

```
<%@ page…%>
```

Following is the basic syntax of the page directive −

```
<%@ page attribute = "value" %>
```

You can write the XML equivalent of the above syntax as follows −

```
<jsp:directive.page attribute = "value" />
```

- It provides attributes that get applied to entire JSP page.
- It defines page dependent attributes, such as scripting language, error page, and buffering requirements.
- It is used to provide instructions to a container that pertains to current JSP page.

Following are its list of attributes associated with page directive:

1. Language
2. Extends
3. Import
4. contentType
5. info
6. session
7. isThreadSafe
8. autoflush
9. buffer
10. IsErrorPage
11. pageEncoding
12. errorPage
13. isELIgonored

language: It defines the programming language (underlying language) being used in the page.**Syntax of language:**

```
<%@ page language="value" %>
```

Here value is the programming language (underlying language)
Extends: This attribute is used to extend (inherit) the class like JAVA does
Import: This attribute is most used attribute in page directive attributes.It is used to tell the container to import other java classes, interfaces, enums, etc. while generating servlet code.It is similar to import statements in java classes, interfaces.
contentType:

It defines the character encoding scheme i.e. it is used to set the content type and the character set of the response
The default type of contentType is “text/html; charset=ISO-8859-1”.
PageEncoding:

“pageEncoding” attribute defines the character encoding for JSP page.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    import="java.util.Date" pageEncoding="ISO-8859-1"%>

<%@ page extends="demotest.DemoClass" %>
```

info

It defines a string which can be accessed by getServletInfo() method.
This attribute is used to set the servlet description.

Session

JSP page creates session by default.
Sometimes we don't need a session to be created in JSP, and hence, we can set this attribute to false in that case.The default value of the session attribute is true, and the session is created.When it is set to false, then we can indicate the compiler to not create the session by default.

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

AutoFlush:

This attribute specifies that the buffered output should be flushed automatically or not and default value of that attribute is true.

If the value is set to false the buffer will not be flushed automatically and if its full, we will get an exception.

When the buffer is none then the false is illegitimate, and there is no buffering, so it will be flushed automatically.

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

Buffer:

Using this attribute the output response object may be buffered.
We can define the size of buffering to be done using this attribute and default size is 8KB.
It directs the servlet to write the buffer before writing to the response object.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    buffer="16KB"%>
```

isErrorPage:

It indicates that JSP Page that has an errorPage will be checked in another JSP page
Any JSP file declared with “isErrorPage” attribute is then capable to receive exceptions from other JSP pages which have error pages.
Exceptions are available to these pages only.
The default value is false.

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

errorPage:

This attribute is used to set the error page for the JSP page if JSP throws an exception and then it redirects to the exception page.

```
<%@ page language="java" contentType="text/html;" pageEncoding="ISO-8859-1"
    errorPage="errorHandler.jsp"%>
```

import	Import list of packages, classes an interfaces in servlet class definition. **Example:** <%@ page import = "java.util.Date" %>
extends	Used to extend the parent class that will be generated by Servlet. **Example:** <%@ page extends = "mypackage.DemoClass"%>
language	Defines which scripting language to be used in the page. **Example:** <%@ page language = "value"%>
session	Specifies the JSP page participating in an HTTP session. **Example:** <%@ page session = "true"%>
buffer	Specifies a buffer model to handle output stream generated by JSP page. **Example:** <%@ page buffer = "4kb"%>
autoFlush	Specifies that buffer should be flushed automatically. The default value of autoFlush attribute is **'true'**. **Example:** <%@ page autoFlush = "false"%>
info	Sets the information of the JSP page which is retrieved later by using **getServletInfo( )** method. **Example:** <%@ page info = "Given by Surendra Maurya"%>
contentType	Sets the content of the JSP page. **Example:** <%@ page contentType="text/html"%>
isThreadSafe	It is used to define the threading model for the JSP page which is generated by Servlet because JSP and servlet both are thread safe. **Example:** <%@ page isThreadSafe="false"%>
errorPage	Define the error page, if any error generates in current page, it will be redirected to the error page. **Example:** <%@ page errorPage="erroropage.jsp"%>
isErrorPage	Defines whether the current JSP page is error page or not. **Example:** <%@ page isErrorPage="true" %>
isELIgnored	Specifies whether the expression will be evaluated or not in JSP page.

Include Directive

The JSP "include directive" is used to include one file in another JSP file. This includes HTML, JSP, text, and other files. This directive is also used to create templates according to the developer's requirement and breaks the pages in the header, footer, and sidebar.

The **include** directive is used to include a file during the translation phase. This directive tells the container to merge the content of other external files with the current JSP during the translation phase. You may code the ***include*** directives anywhere in your JSP page.

The general usage form of this directive is as follows −

```
<%@ include file = "relative url" >
```

You can write the XML equivalent of the above syntax as follows −

```
<jsp:directive.include file = "relative url" />
```

```
<html>
<body>

<%@ include file="header.html" %>

Today is: <%= java.util.Calendar.getInstance().getTime() %>

</body>
</html>
```

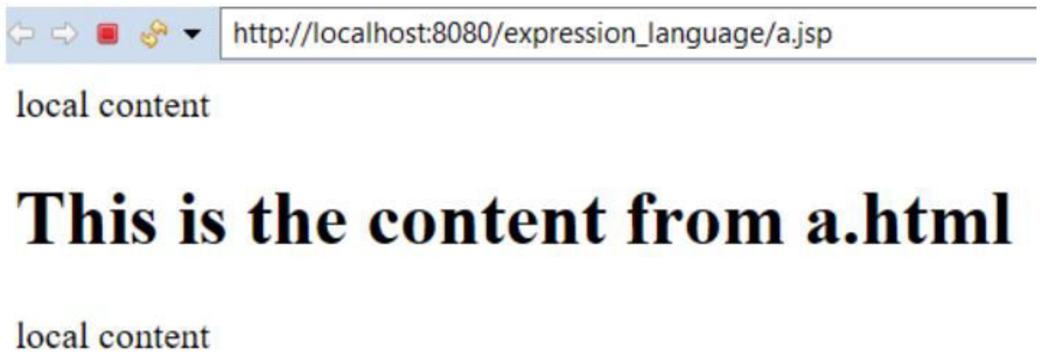
a.html

```
<h1>This is the content of a.html</h1>
```

index.jsp

```
<% = Local content%>
<%@include file = "a.html"%>
<% = local content%>
```

- **Output :**



http://localhost:8080/expression_language/a.jsp

local content

This is the content from a.html

local content

- **Taglib Directive :** The taglib directive is used to mention the library whose custom-defined tags are being used in the JSP page. It's major application is JSTL(JSP standard tag library).
  Syntax:

```
<%@taglib uri = "library url" prefix="the prefix to
identify the tags of this library with"%>
```

```
<%-- JSP code to demonstrate
taglib directive--%>
<%@ taglib uri = "http://java.sun.com/jsp/jstl/core"
prefix = "c" %>

<c:out value = "${'This is Sparta'}"/>
```

Implicit objects

- JSP implicit objects are created during the translation phase of JSP to the servlet.
- These objects can be directly used in scriplets that goes in the service method.
- They are created by the container automatically, and they can be accessed using objects.

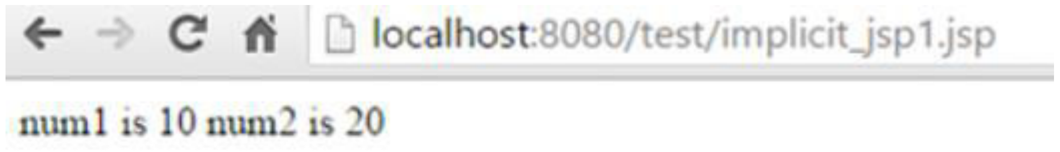
There are 9 types of implicit objects available in the container:

1. Out
2. Request
3. Response
4. Config
5. Application
6. Session
7. PageContext
8. Page
9. Exception

Out

- Out is one of the implicit objects to write the data to the buffer and send output to the client in response
- Out object allows us to access the servlet's output stream
- Out is object of javax.servlet.jsp.jspWriter class
- While working with servlet, we need printwriter object

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<body>
<% int num1=10;int num2=20;
out.println("num1 is " +num1);
out.println("num2 is "+num2);
%>
</body>
</html>
```



localhost:8080/test/implicit_jsp1.jsp

```
num1 is 10 num2 is 20
```

request Object

The JSP request is an implicit object which is provided by HttpServletRequest. In the servlet, we have to first import javax.servlet.http.HttpServletRequest then we have to create its object for taking input from any HTML form as.

Example of JSP request implicit object

index.html

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

welcome.jsp

```
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
```

response object

This is the HttpServletResponse object associated with the response to the client. The response object also defines the interfaces that deal with creating new HTTP headers. Through this object the JSP programmer can add new cookies or date stamps, HTTP status codes, etc.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<%
String name=request.getParameter("username");
out.print("Welcome "+ name);
%>
</body>
</html>
```

The session Object

The session object is an instance of **javax.servlet.http.HttpSession** and behaves exactly the same way that session objects behave under Java Servlets.

Syntax: session.getAttribute()

Methods used with session object are:

a. setAttribute(String, object) – This method will assign a string to the object and save it in session.

b. getAttribute(String name) – This method gets the object that is set by setAttribute.

c. removeAttribute(String name) – Objects that are in the session can be removed using this method.

d. getAttributeNames – It returns enumeration of the objects in session.

e. getCreationTime – This method returns time when the session was in active mode.

f. getId –This method gives out the unique id of the session.

g. isNew – This method checks if a session is new or not. It would return true if cookies are not there.

h. getMaxInactiveInterval – Returns time span, the session was inactive.

i. getLastAccessedTime – This method gives the last accessed time of a session.

```
<html>
<head><title>Config</title></head>
<body>
<form action="first.jsp">
<input type="text" name="username">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

```
 First.jsp
<html>
<head><title>Config</head></title>
<body>
<%
String myname=request.getParameter("username");
out.print("Welcome "+myname);
  session.setAttribute("user",myname);  %>
  <a href="second.jsp">second jsp page</a>
 </body>
</html>
```

```
second.jsp
<html>
<body>
<%
 String name=(String)session.getAttribute("user");
out.print("Hello "+myname);  %>
</body>
</html>
```

Exception Handling in JSP

The exception is normally an object that is thrown at runtime. Exception Handling is the process to handle the runtime errors. There may occur exception any time in your web application. So handling exceptions is a safer side for the web developer. In JSP, there are two ways to perform exception handling:

1. By **errorPage** and **isErrorPage** attributes of page directive
2. By **<error-page>** element in web.xml file

index.jsp

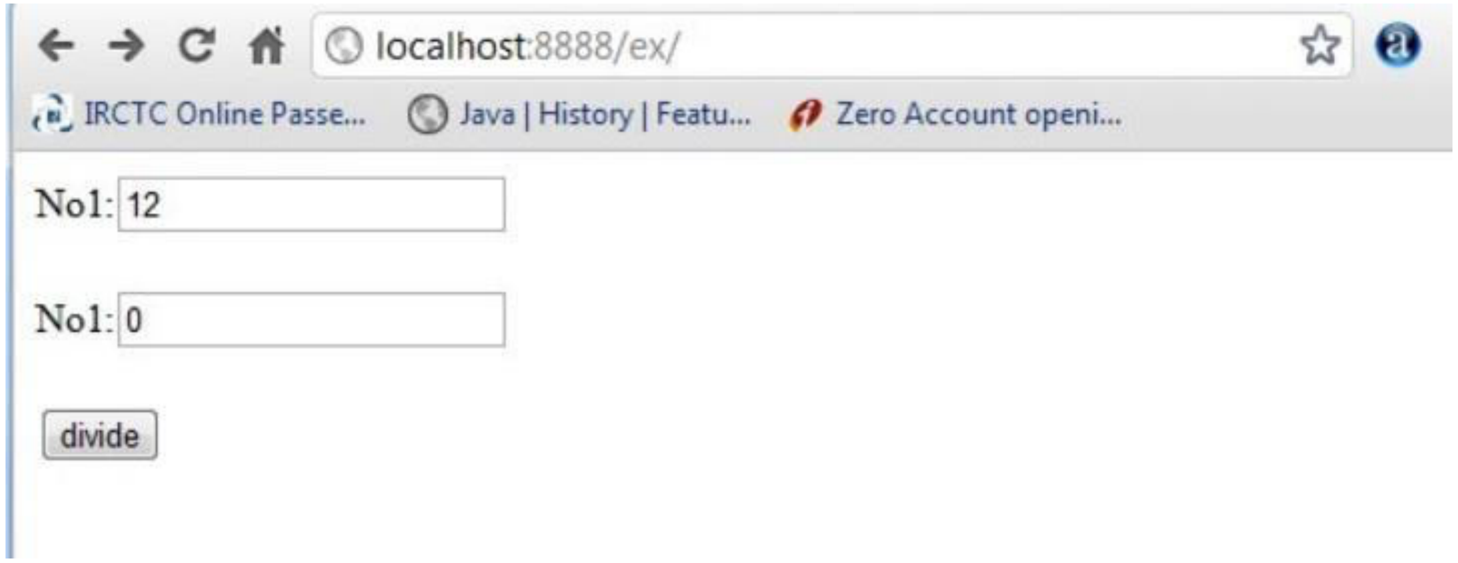
```
<form action="process.jsp">
No1:<input type="text" name="n1" /><br/><br/>
No1:<input type="text" name="n2" /><br/><br/>
<input type="submit" value="divide"/>
</form>
```

process.jsp

```
<%@ page errorPage="error.jsp" %>
<%


String num1=request.getParameter("n1");
String num2=request.getParameter("n2");


int a=Integer.parseInt(num1);
int b=Integer.parseInt(num2);
int c=a/b;
out.print("division of numbers is: "+c);


%>
```

error.jsp

```
<%@ page isErrorPage="true" %>

<h3>Sorry an exception occured!</h3>

Exception is: <%= exception %>
```



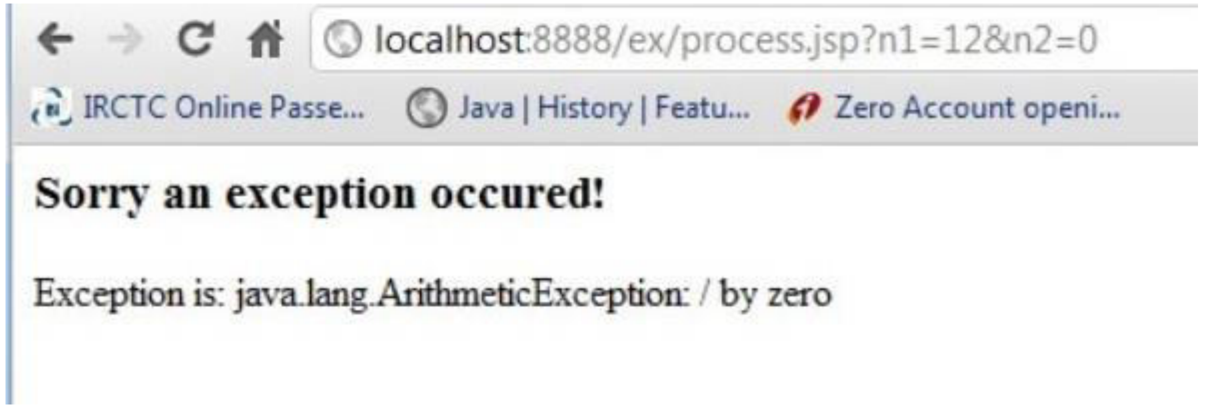
localhost:8888/ex/

IRCTC Online Passe... Java | History | Featu... Zero Account openi...

No1: 12

No1: 0

divide



localhost:8888/ex/process.jsp?n1=12&n2=0

IRCTC Online Passe... Java | History | Featu... Zero Account openi...

Sorry an exception occured!

Exception is: java.lang.ArithmeticException: / by zero

JSP Form Processing using Get and Post Methods

There are two commonly used methods to send and receive back information to the webserver.

The browser uses two methods to pass this information to the web server. These methods are the GET Method and the POST Method.

1.GET Method:
•This is the default method to pass information from browser to web server.
•It sends the encoded information separated by ?character appended to URL page.
•It also has a size limitation, and we can only send 1024 characters in the request.
•We should avoid sending password and sensitive information through GET method.

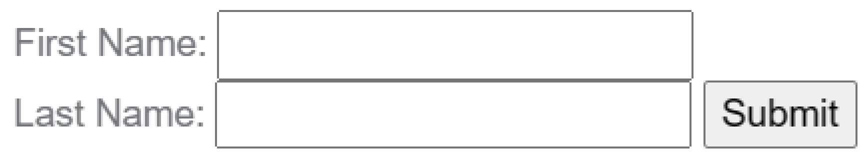
2.POST Method:
•Post method is a most reliable method of sending information to the server.
•It sends information as separate message.
•It is commonly used to send information which are sensitive.

JSP handles form data processing by using following methods:
1.getParameter():It is used to get the value of the form parameter.
2.getParameterValues():It is used to return the multiple values of the parameters.
3.getParameterNames()It is used to get the names of parameters.
4.getInputStream()It is used to read the binary data sent by the client.

GET Method Example Using Form

Following is an example that passes two values using the HTML FORM and the submit button. We are going to use the same JSP main.jsp to handle this input.

```
<html>
   <body>

      <form action = "main.jsp" method = "GET">
         First Name: <input type = "text" name = "first_name">
         <br />
         Last Name: <input type = "text" name = "last_name" />
         <input type = "submit" value = "Submit" />
      </form>

   </body>
</html>
```



main.jsp

```
<html>
   <head>
      <title>Using GET Method to Read Form Data</title>
   </head>

   <body>
      <h1>Using GET Method to Read Form Data</h1>
      <ul>
         <li><p><b>First Name:</b>
            <%= request.getParameter("first_name")%>
         </p></li>
         <li><p><b>Last  Name:</b>
            <%= request.getParameter("last_name")%>
         </p></li>
      </ul>

   </body>
</html>
```

```html
<html>
   <body>
      
      <form action = "main.jsp" method = "POST">
         First Name: <input type = "text" name = "first_name">
         <br />
         Last Name: <input type = "text" name = "last_name" />
         <input type = "submit" value = "Submit" />
      </form>
      
   </body>
</html>
```

No change in main.jsp file

box.html

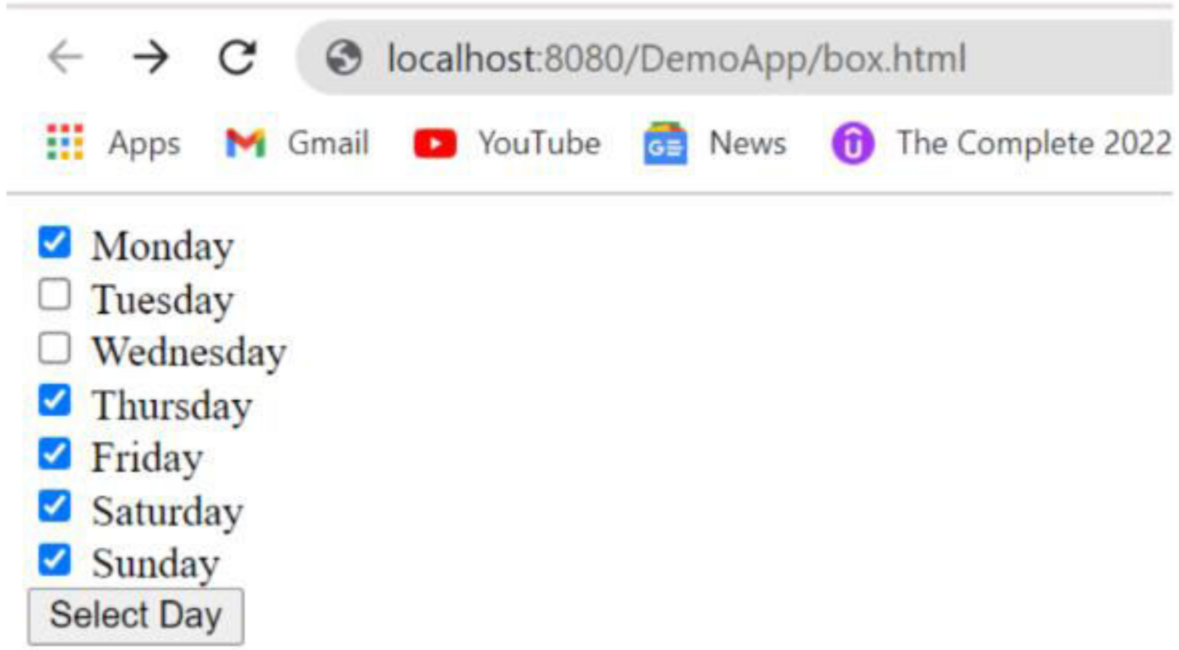
```
<!DOCTYPE html>
<html>
<head>
<title>Processing Checkbox data</title>
</head>
<body>
  <form action="checkbox.jsp" method="POST" target="_blank">
  <input type="checkbox" name="Monday" checked="checked" /> Monday</br>
  <input type="checkbox" name="Tuesday" /> Tuesday</br>
  <input type="checkbox" name="Wednesday" /> Wednesday</br>
  <input type="checkbox" name="Thrusday"checked="checked" /> Thrusday</br>
  <input type="checkbox" name="Friday" checked="checked" /> Friday</br>
  <input type="checkbox" name="Saturday" checked="checked" /> Saturday</br>
  <input type="checkbox" name="Sunday" checked="checked" /> Sunday</br>
  <input type="submit"value="Select Day" />
  </form>
</body>
</html>
```

box.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<title>Reading Checkbox Data</title>
</head>
<body>
 <h1>Working Days</h1>
 <ul>
  <li>
      <p>
         <b>Monday:</b>
         <%=request.getParameter("Monday")%>
      </p>
  </li>
    <li>
      <p>
         <b>Tuesday:</b>
         <%=request.getParameter("Tuesday")%>
      </p>
    </li>
    <li>
      <p>
         <b>Wednesday:</b>
         <%=request.getParameter("Wednesday")%>
      </p>
    </li>
```

```
    <li>
       <p>
          <b>Thrusday:</b>
          <%=request.getParameter("Thrusday")%>
       </p>
    </li>
    <li>
       <p>
          <b>Friday:</b>
          <%=request.getParameter("Friday")%>
       </p>
    </li>
    <li>
       <p>
          <b>Saturday:</b>
          <%=request.getParameter("Saturday")%>
       </p>
    </li>
    <li>
       <p>
          <b>Sunday:</b>
          <%=request.getParameter("Sunday")%>
       </p>
    </li>
  </ul>
 </body>
 </html>
```

Output:

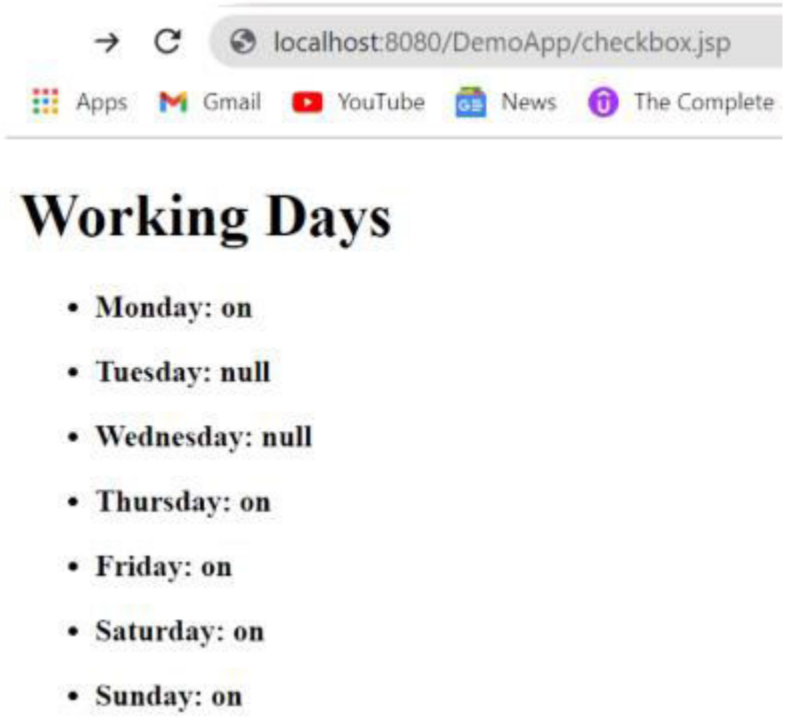


localhost:8080/DemoApp/box.html

Apps Gmail YouTube News The Complete 2022

[✓] Monday
[ ] Tuesday
[ ] Wednesday
[✓] Thursday
[✓] Friday
[✓] Saturday
[✓] Sunday

Select Day



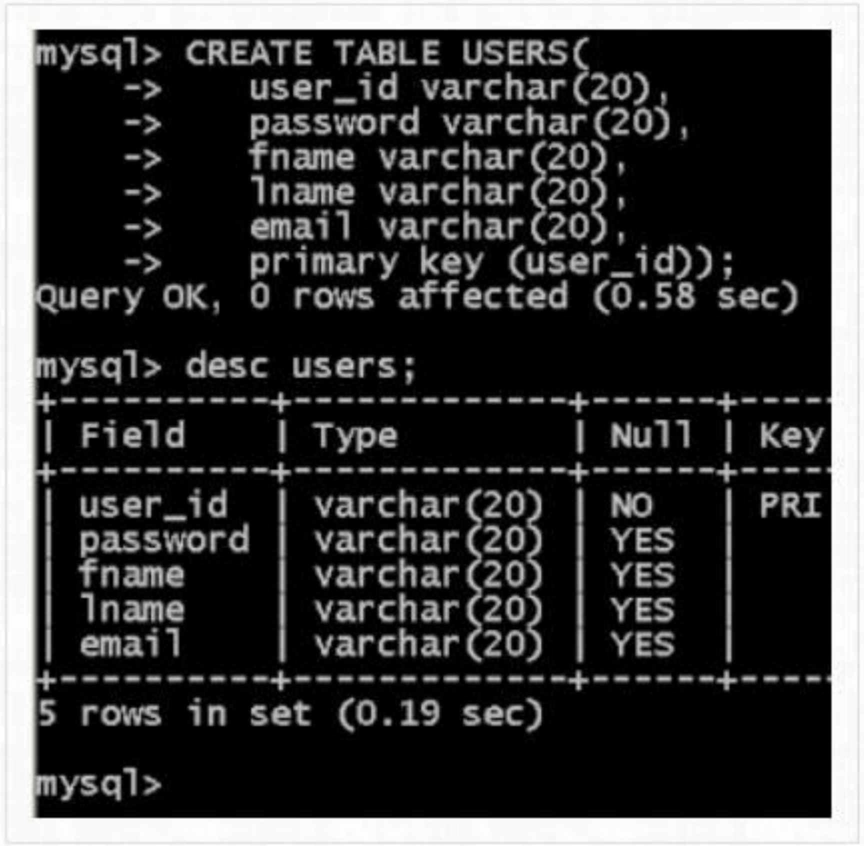
localhost:8080/DemoApp/checkbox.jsp

Apps Gmail YouTube News The Complete

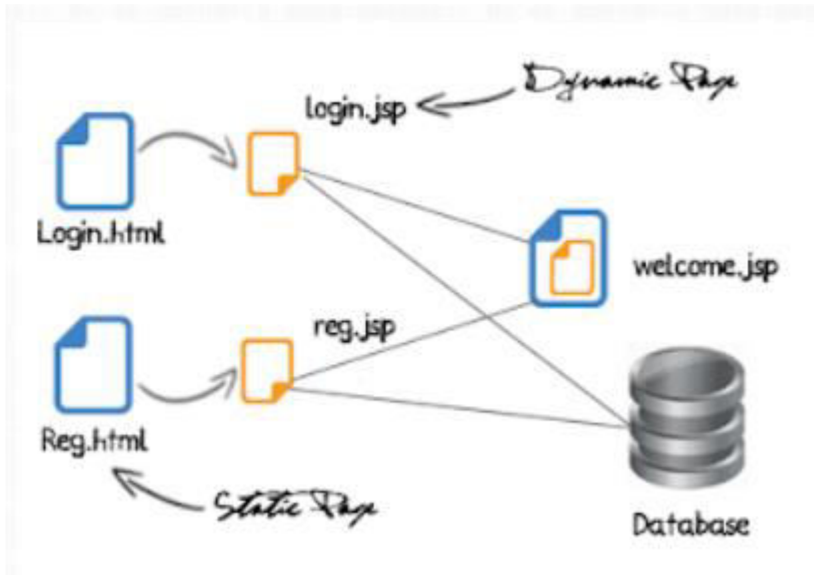
Working Days

- **Monday: on**
- **Tuesday: null**
- **Wednesday: null**
- **Thursday: on**
- **Friday: on**
- **Saturday: on**
- **Sunday: on**

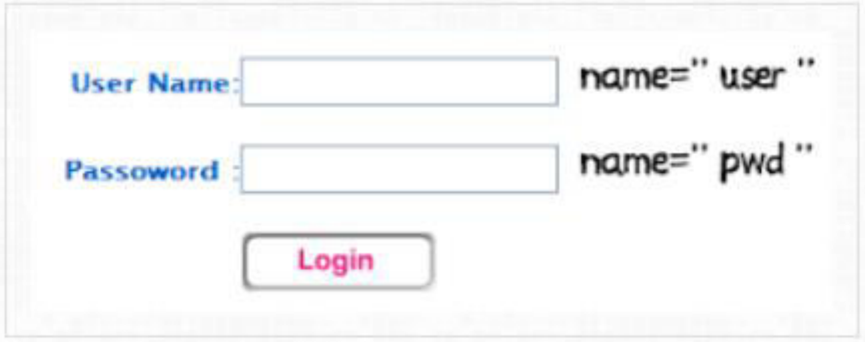
Connecting to database in JSP



```
mysql> CREATE TABLE USERS(
    ->     user_id varchar(20),
    ->     password varchar(20),
    ->     fname varchar(20),
    ->     lname varchar(20),
    ->     email varchar(20),
    ->     primary key (user_id));
Query OK, 0 rows affected (0.58 sec)

mysql> desc users;
+----------+-------------+------+----
| Field    | Type        | Null | Key
+----------+-------------+------+----
| user_id  | varchar(20) | NO   | PRI
| password | varchar(20) | YES  |
| fname    | varchar(20) | YES  |
| lname    | varchar(20) | YES  |
| email    | varchar(20) | YES  |
+----------+-------------+------+----
5 rows in set (0.19 sec)

mysql>
```



Login.html

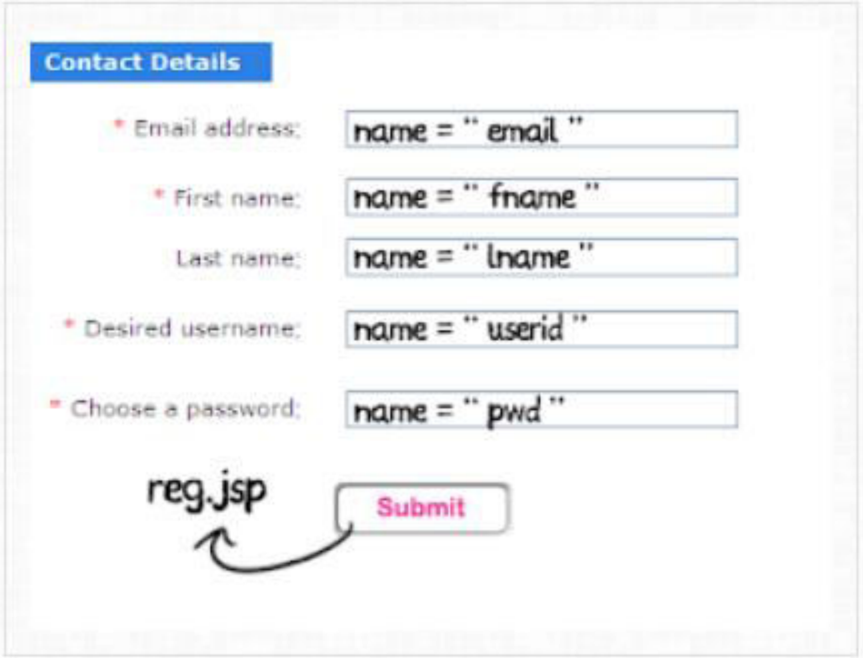


```
<body>
<form action="login.jsp" method="post">

User name :<input type="text" name="usr" />
password :<input type="password" name="pwd" />
<input type="submit" />


</form>
</body>
```

login.jsp

```
<%@ page import ="java.sql.*" %>
<%@ page import ="javax.sql.*" %>
<%
String userid=request.getParameter("user");
session.putValue("userid",userid);
String pwd=request.getParameter("pwd");
Class.forName("com.mysql.jdbc.Driver");
java.sql.Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/
test","root","root");
Statement st= con.createStatement();
ResultSet rs=st.executeQuery("select * from users where user_id='"+userid+"'");
if(rs.next())
{
if(rs.getString(2).equals(pwd))
{
out.println("welcome"+userid);

}
else
{
out.println("Invalid password try again");
}
}
else
%>
```

Reg.html


Contact Details

* Email address:	name = " email "
* First name:	name = " fname "
Last name:	name = " lname "
* Desired username:	name = " userid "
* Choose a password:	name = " pwd "

```
<form action="reg.jsp" method="post">

Email :<input type="text" name="email" />
First name :<input type="text" name="fname" />
Last name :<input type="text" name="lname" />
User name :<input type="text" name="userid" />
password :<input type="password" name="pwd" />
<input type="submit" />


</form>
```

reg.jsp

```
<%@ page import ="java.sql.*" %>
<%@ page import ="javax.sql.*" %>
<%
String user=request.getParameter("userid");
session.putValue("userid",user);
String pwd=request.getParameter("pwd");
String fname=request.getParameter("fname");
String lname=request.getParameter("lname");
String email=request.getParameter("email");
Class.forName("com.mysql.jdbc.Driver");
java.sql.Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/test",
"root","root");
Statement st= con.createStatement();
ResultSet rs;
int i=st.executeUpdate("insert into users values ('"+user+"','"+pwd+"','"+fname+"',
'"+lname+"','"+email+"')");


%>
```

Java Beans

- JavaBeans components are Java classes that can be easily reused and composed together into applications.
- A JavaBean can be defined as a reusable software component.
- We can write a JavaBean that can then be used in a variety of other Java based softwares such as applications, Servlets or JSP pages.
- In this way we can define our business logic within a JavaBean and then consistently use that logic in separate applications.
- **Benefits of JavaBeans**
- By using JavaBeans we can fully separate the business logic from the generation of the display.
- The other advantage of using JavaBeans is that the business logic can be used by more than one application.
- example, both a client based Java application and a JSP page can access the same JavaBean thus guaranteeing the same functionality.

More specifically a JavaBean is just a Java class with the following rules,

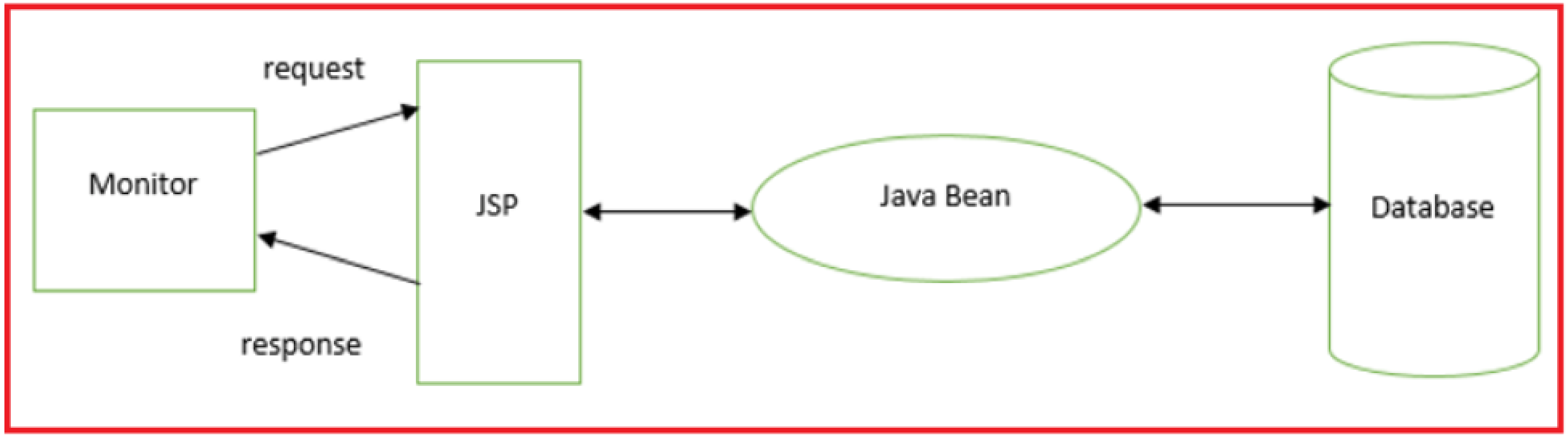
- It is a public class.
- It has a public constructor with no arguments.
- It has public getter and setter methods to read and write to properties.

•Properties are always accessed using a common naming convention. For each property two methods must exist: a getXxx() and a setXxx() method where xxx is the name of the property and of a private instance variable. They are called *getter* and *setter* methods respectively.

•For example, if we want a property called **fileName** then we would need to define an instance variable called fileName and two methods: getFileName() and setFileName().

Working of JavaBeans in JSP

First, the browser sends the request for the JSP page. Then the JSP page accesses Java Bean and invokes the business logic. After invoking the business logic Java Bean connects to the database and gets/saves the data. At last, the response is sent to the browser which is generated by the JSP.



Example 1

EmployeeBean.java

```
package com.example;

public class EmployeeBean {
  private String firstName = "";
  private String lastName  = "";

  //First Name property
  public void setFirstName(String name) {
    firstName = name;
  }

  public String getFirstName() {
    return firstName;
  }

  //Last Name Property
  public void setLastName(String name) {
    lastName = name;
  }

  public String getLastName() {
    return lastName;
  }

  //Full Name Property - Read Only
  public String getFullName() {
    return firstName + " " + lastName;
  }

}
```

JavaBeans and JSP:

Java Server Pages technology directly supports using JavaBeans components with JSP language elements. We can easily create and initialize beans and get and set the values of their properties.

JSP Actions for JavaBeans:

To use a JavaBean in our JSP page we make use of three JSP actions that we saw earlier. They are,

- <jsp: useBean/> – find or instantiate a JavaBean
- <jsp: setProperty/> – sets the property of a JavaBean
- <jsp: getProperty/> – gets the value of a JavaBean property

The first of these is used to specify which JavaBean we want to use in our page. The next two actions, as their names suggest, are used to set and get values of properties of the JavaBean.

Attributes and Usage of jsp:useBean action tag

1. **id:** is used to identify the bean in the specified scope.
2. **scope:** represents the scope of the bean. It may be page, request, session or application. The default scope is page.
    - **page:** specifies that you can use this bean within the JSP page. The default scope is page.
    - **request:** specifies that you can use this bean from any JSP page that processes the same request. It has wider scope than page.
    - **session:** specifies that you can use this bean from any JSP page in the same session whether processes the same request or not. It has wider scope than request.
    - **application:** specifies that you can use this bean from any JSP page in the same application. It has wider scope than session.
3. **class:** instantiates the specified bean class (i.e. creates an object of the bean class) but it must have no-arg or no constructor and must not be abstract.

Syntax:

```
<jsp:useBean id="beanID" scope="beanScope" class="package.beanName" />

 <jsp:useBean id="EmployeeBean" scope="page" class="com.example.EmployeeBean" />
```

The setProperty and getProperty Actions

These are used to set and read property values.

Syntax

```
<jsp:setProperty name="bean" property="propertyName" value="val" />

<jsp:getProperty name="bean" property="propertyName" />
```

where beanID is the variable name used in the <jsp:useBean/> tag and propertyName is the name of the property we are setting or getting.

To use a variable as either a property or a more often as a value then, between the quotes, enter a JSP expression that returns the value of the variable.

For example,

```
<jsp:setProperty name="empBean" property="firstName"  value="<%= strFirst %>" />
```

where **strFirst** is a variable used in the page.

Example of jsp:setProperty action tag if you have to set all the values of incoming request in the bean: `<jsp:setProperty name="bean" property="*" />`

EmpBeanTest.jsp

```
<html>
<head>
<title>Simple Java Bean</title>
</head>

<body>

<jsp:useBean id="EmployeeBean" scope="page"
class="com.example.EmployeeBean" />

<%-- Set bean properties --%>
<jsp:setProperty name="empBean" property="firstName" value="AAA" />

<jsp:setProperty name="empBean" property="lastName" value="BBB" />

<%-- Get bean properties --%>
<P>
<b>Full Name:</b> <jsp:getProperty name="empBean" property="fullName"
/>
</P>

</body>
</html>
```

Example 2:

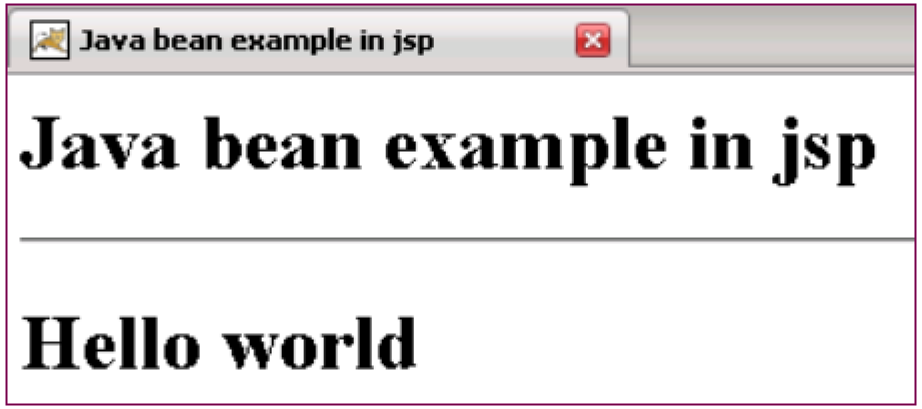
MyBean.java

```
package my;
public class MyBean {

 private String name=new String();

 public String getName() {
 return name;
 }
 public void setName(String name) {
 this.name = name;
 }
 }
```

UseBean.jsp

```
<html>
<title>Java bean example in jsp</title>
<head>
<h1>Java bean example in jsp</h1>
<hr></hr>
</head>
<body>
<jsp:useBean id="mybean" class="my.MyBean" scope="session" >
<jsp:setProperty name="mybean" property="name" value=" Hello world" />
</jsp:useBean>

<h1> <jsp:getProperty name="mybean" property="name" /></h1>
</body>
</html>
```

Output of the program



Java bean example in jsp

Java bean example in jsp

Hello world

Example 3

```
//info.html
<html>
<head>
<title>Request More Information</title>
</head>
<body>
<h1>More Information</h1>
Please use this basic form to select the course
that you would like further information on.

<form method="get" action="moreInformationRequestWithBean.jsp">
  <br><input type="radio" name="courses" value="Java Programming"> Java Programming
  <br><input type="radio" name="courses" value="Java Web Development"> Java Web Development
  <br><input type="radio" name="courses" value="J2EE Development"> J2EE Development
  <br><input type="radio" name="courses" value="XML Introduction"> XML Introduction
  <br><input type="radio" name="courses" value="XML Schema"> XML Schema
  <br><input type="radio" name="courses" value="Web Services"> Web Services

  <p>First name: <input type="text" name="firstName">
  <br>Last name: <input type="text" name="lastName">
  <br>Email: <input type="text" name="email">

  <p><input type="submit" name="Submit">
</form>
</body>
</html>
```

```
<!-- moreInformationRequestWithBean.jsp -->

<%@ page import="com.java2s.*"%>
<jsp:useBean id="infoRequest" scope="session"
type="MoreInfoRequest"/>
<jsp:setProperty name="infoRequest" property="*"/>
<html>
<head>
<title>Thankyou for your request</title>
</head>
<body>
<h1>Thankyou for your request</h1>
Thankyou for your request for more information.
It will be sent to you shortly.
<p>Click <a href="displayYourRequest.jsp">here</a> to view your request.
</body>
</html>
```

```java
package com.java2s;

public class MoreInfoRequest
{
  public String getFirstName() {
    return firstName;
  }
  public void setFirstName(String firstName) {
    this.firstName = firstName;
  }

  public String getLastName() {
    return lastName;
  }
  public void setLastName(String lastName) {
    this.lastName = lastName;
  }

  public String getCourses() {
    return courses;
  }
  public void setCourses(String courses) {
    this.courses = courses;
  }

  public String getEmail() {
    return email;
  }
    public void setEmail(String email) {
    this.email = email;
  }

  private String firstName;
  private String lastName;
  private String email;
  private String courses;
```

Example 4

1. bean.java

```
package myexample;
import java.io.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import java.sql.ResultSet;
public class bean
{
  private int msgid;
  private String message;
  private Connection connection=null;
  private ResultSet rs = null;
  private Statement st = null;
  String connectionURL = "jdbc:mysql://192.168.10.59/messagepaging";
  public bean()
  {
 try {
   Class.forName("com.mysql.jdbc.Driver"); // Load the database driver
    connection = DriverManager.getConnection(connectionURL, "root", "root"); // Connection to the db
  }catch(Exception e){
  System.out.println("Exception is ;"+e);
  }
  }
public void setmsgid(int msgid)
 {
 this.msgid = msgid;
 }
 public int getmsgid()
 {
 return (this.msgid);
 }
 public void setmessage(String message)
 {
 this.message = message;
 }
  public String getmessage()      {       return (this.message);      }
  public void insert()
 {
  try
{
 String sql = "insert into message(id,message) values('"+msgid+"','"+message+"')";
 Statement s = connection.createStatement();
 s.executeUpdate (sql);
 s.close ();
 }catch(Exception e){
 System.out.println("Exception is ;"+e);
 }    }    }
```

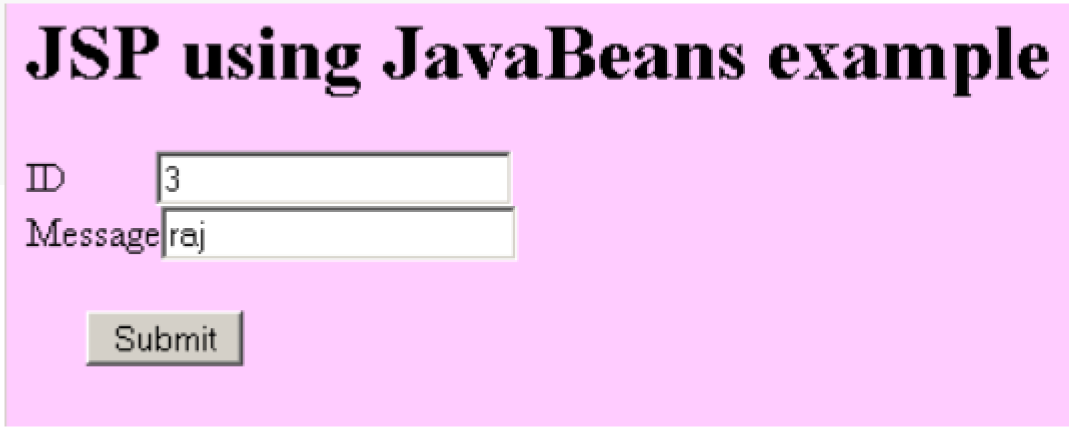
2. jspBean.jsp

```
<%@ page language="Java" import="java.sql.*" %>

<html>
    <head><title>JSP with Javabeans</title></head>
<body bgcolor="#ffccff">
<h1>JSP using JavaBeans example</h1>
    <form name="form1" method="POST">

         ID&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;
         <input type="text" name ="msgid"> <br>
         Message<input type="text" name ="message"> <br>
         <br>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;
         <input type = "submit" value="Submit">
         <jsp:useBean id="sample" class="myexample.bean" scope="page">
            <jsp:setProperty name="sample" property="*"/>
        </jsp:useBean>
    </form>
    <% sample.insert();%>
</body>
</html>
```

```
CREATE TABLE `message` (
`id` int(11) NOT NULL
auto_increment,
`message` varchar(256) default
NULL,
PRIMARY KEY (`id`)
)
```

To run this example you have to follow these steps:

- Create and save "bean.java".
- Put this package "myexample" in the classes folder of **WEB-INF.**
- Create and save jspBean.jsp .


JSP using JavaBeans example

ID 3
Message raj