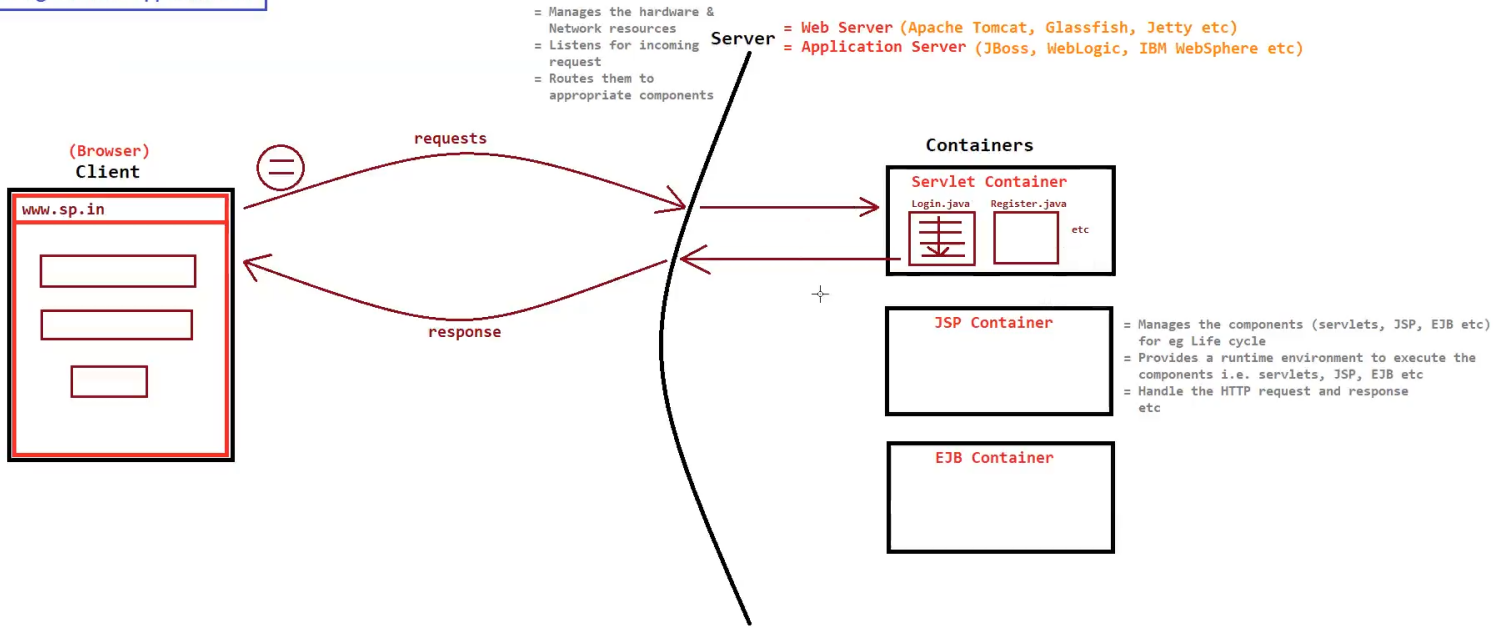
Working of Web Application



```
=> Web Application :-
    -> A web application is a type of software application that is accessed and used through a web
    browser over the network
    -> Advantages of web application :-
        1. Platform Independent
        2. No any other software installation required
        3. Real-time updates
        4. Responsive design
           etc

    -> Working of web application (in context of java) :-
        = Diagram

    -> Create simple java web application in eclipse :-
        = steps

    -> Web technologies in java :-
        1. Servlet
        2. JSP (Java Server Pages)
        3. Frameworks
            = Spring WEB Module (primarily used for back-end but can be used for front-end part also)
            = Apache Struts (primarily used for front-end part)
            = JSF (JavaServer Faces) (primarily used for front-end part)
              etc
```

=> Servlet :-

-> What is Servlet ?

1. Servlet is a "Web Technology" which is used to create dynamic web application

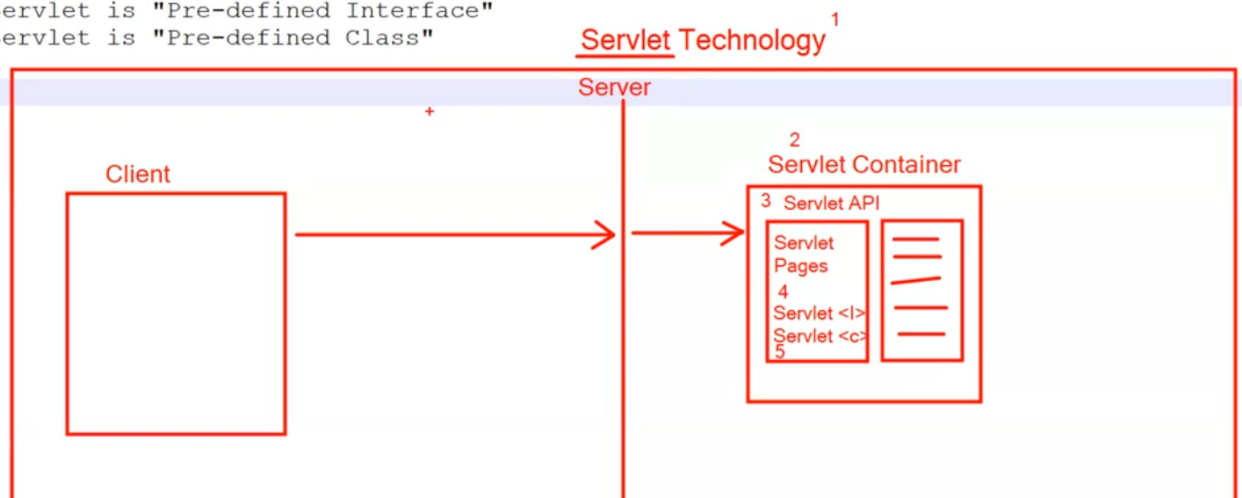
2. Servlet is a "Web Container" which is used to process the dynamic requests (Servlet Container)

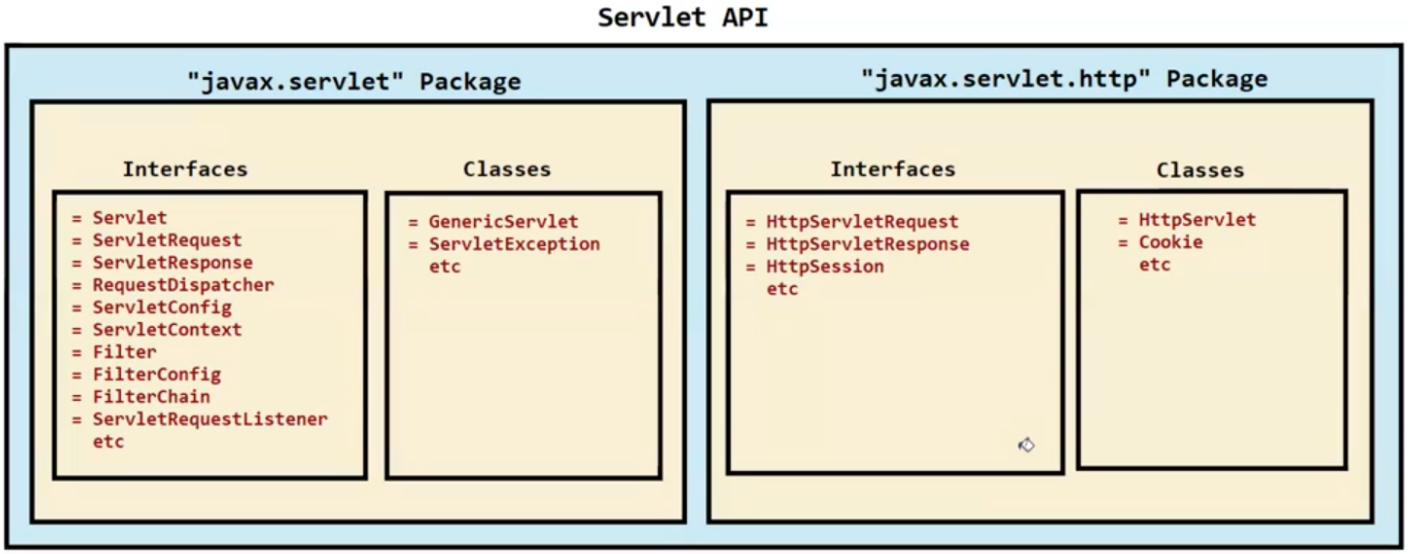
3. Servlet is an "API" (which contains classes and interfaces and documentation)

4. Servlet is "Pre-defined Interface"

5. Servlet is "Pre-defined Class"

->




Servlet API
"javax.servlet" Package
Interfaces
= Servlet
= ServletRequest
= ServletResponse
= RequestDispatcher
= ServletConfig
= ServletContext
= Filter
= FilterConfig
= FilterChain
= ServletRequestListener
etc
Classes
= GenericServlet
= ServletException
etc
"javax.servlet.http" Package
Interfaces
= HttpServletRequest
= HttpServletResponse
= HttpSession
etc
Classes
= HttpServlet
= Cookie
etc

=> Servlet :-

-> What is Servlet ?

1. Servlet is a "Web Technology" which is used to create dynamic web application

2. Servlet is a "Web Container" which is used to process the dynamic requests (Servlet Container)

3. Servlet is an "API" (which contains classes and interfaces and documentation)
4. Servlet is "Pre-defined Interface"
5. Servlet is "Pre-defined Class"

-> Servlet API :-

= It contains 2 main packages :-

1. "javax.servlet" package
2. "javax.servlet.http" package

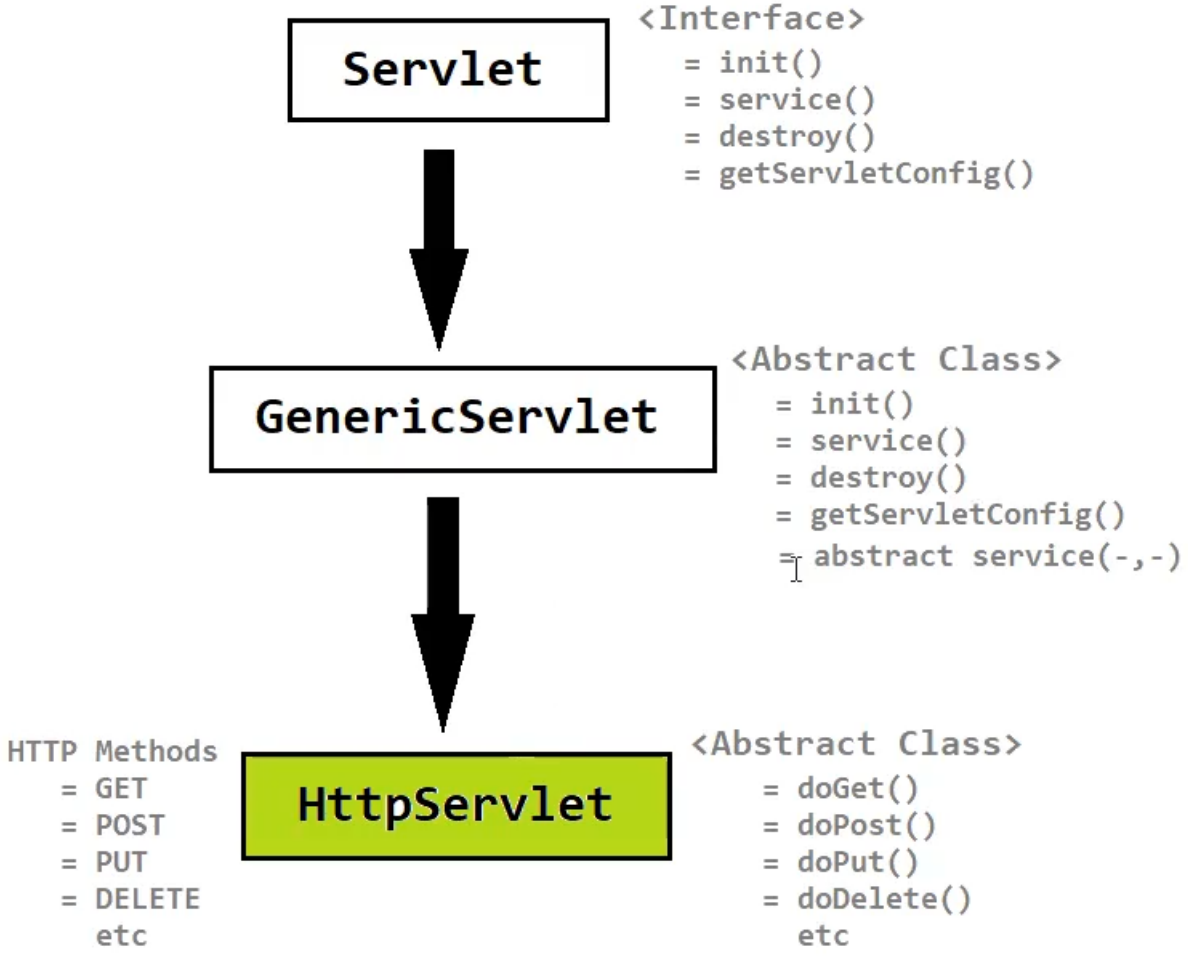
-> How to create Servlet :-

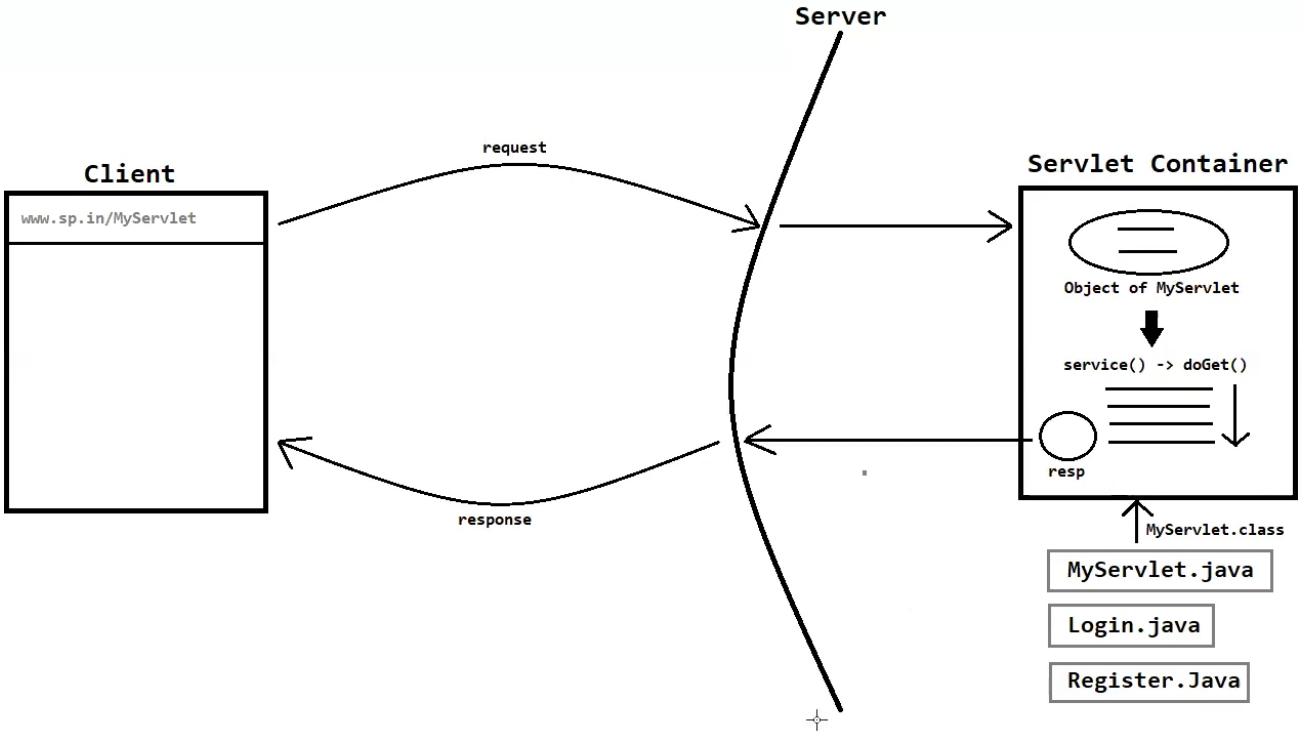
= There are 3 ways to create Servlet :-

1. By implementing "Servlet" interface
2. By extending "GenericServlet" class
3. By extending "HttpServlet" class

= NOTE :-

- We mostly create servlet by extending "HttpServlet" class because it follows the HTTP protocol (which is important in web development)


Servlet
<Interface>
= init()
= service()
= destroy()
= getServletConfig()
GenericServlet
<Abstract Class>
= init()
= service()
= destroy()
= getServletConfig()
= abstract service(-,-)
HTTP Methods
= GET
= POST
= PUT
= DELETE
etc
HttpServlet
<Abstract Class>
= doGet()
= doPost()
= doPut()
= doDelete()
etc


Server
Client
www.sp.in/MyServlet
request
response
Servlet Container
Object of MyServlet
service() -> doGet()
resp
MyServlet.class
MyServlet.java
Login.java
Register.Java

*new 1 - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

Web App Introduction.txt | Servlet Introduction.txt | new 1

```
=> Servlet Life-Cycle :-
    -> Below are the steps of Servlet Life-Cycle :-
        1. Loading & Instantiation :-
            = When the servlet is first requested or when the web application starts, the servlet class is loaded in the servlet container and new instance of that servlet class is created
        2. Initialization :-
            = After instantiation, the init() method is called by servlet container to initialize the servlet object
            = We can override the init() method to perform any one time setup operations for eg. loading configuration data, establish database connection, initializing resources etc
        3. Request Handling :-
            = Once the servlet is initialized, it is ready to handle the client requests
            = The service() method is called by the container for each incoming HTTP request
            = The service() method examins the request, determines the appropriate HTTP method (eg GET, POST etc) and deligates the request to the corresponding doXXX() method (eg doGet(), doPost() etc)
        4. Response Generation :-
            = During the request handling process, the servlet generates a response that may include HTML, XML, JSON etc
            = The response will be written to the HttpServletResponse object associated with the request
        5. Termination / Destruction :-
            = When the servlet container decides to shut down the web application or unload the servlet, it will call the destroy() method
            = The destroy() method allows us to perform cleanup operations for eg. closing the database connections, releasing resources etc
        6. Servlet Deinstantiation :-
            = After calling the destroy() method, the servlet container removes the servlet instance from memory
```

Normal text file | length : 1,907 lines : 27 | Ln : 27 Col : 8 Pos : 1,908 | Windows (CR LF) | UTF-8 | INS

ENG

-> NOTE :-
    = Servlet life-cycle is managed by "Servlet Container"
    = Servlet Loading, Servlet Instantiation and Servlet Initialization are executed only once
    = When the client sends the multiple requests, a new thread will be created which will be associated with the request and every thread will execute service() method seperately

=> "web.xml" File :-

-> To access any servlet from browser, we have to tell the servlet container that which servlet it has to deploye and according to provided URL which servlet it has to invoke. For this purpose we use "web.xml" file

-> "web.xml" file is also known as "deployment descriptor file"

-> Responsibilities of web.xml file :-

1. Servlet configurations
2. JSP file configurations
3. Filters configurations
4. Listeners configurations
5. Initialization parameters configurations
6. Context parameters configurations
7. Session timeout configurations
8. Welcome file configurations
9. Error page configurations
10. Servlet load-on-startup configurations

etc

-> Syntax :-

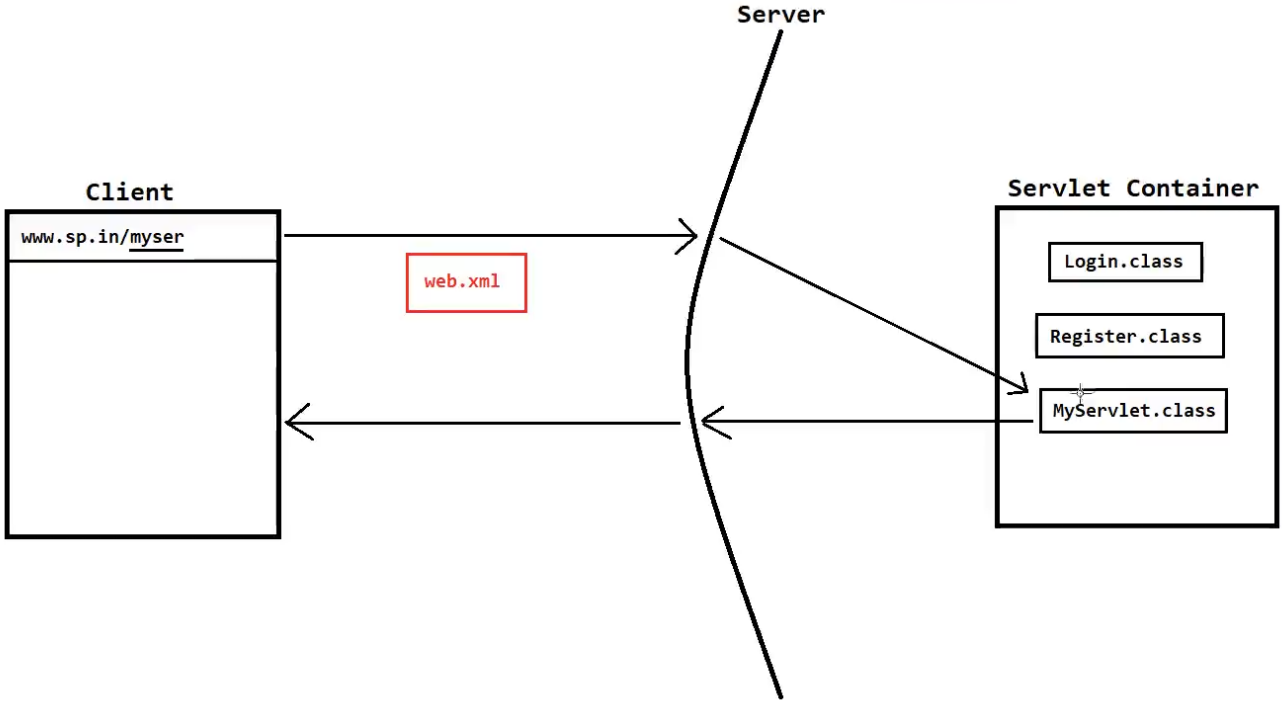
```
<web-app>
    <servlet>
        <servlet-name> ms </servlet-name>
        <servlet-class> package_name.MyServlet </servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name> ms </servlet-name>
        <url-pattern> /myser </url-pattern>
    </servlet-mapping>
</web-app>
```

-> NOTE :-

= We create "web.xml" file in "WEB-INF" folder

= In latest application, we normally use annotations. In legacy projects or in case of complex configurations we use "web.xml" file

```
<web-app>
    <servlet>
        <servlet-name> ms </servlet-name>
        <servlet-class> in.sp.backend.MyServlet </servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name> ms </servlet-name>
        <url-pattern> /myser </url-pattern>
    </servlet-mapping>
</web-app>
```


Server
Client
www.sp.in/myser
web.xml
Servlet Container
Login.class
Register.class
MyServlet.class

=> Annotations in Java :
-> Annotations are the metadata (information) which are added to the programming elements i.e. class, interface, method, constructor, variable etc
-> Annotations are used for configuration, documentation and to convey addition information at compile-time or runtime
-> Annotations starts with '@'
-> Categories of annotations :-
1. Marker Annotations
= It is used to mark a declaration. It does not contain any members or data
= For eg. @Override, @Depricated, @FunctionalInterface
2. Single Value Annotations
= In this annotations, there is only one member
= For eg. @SuppressWarnings("unchecked")
3. Full Annotations
= These annotations consists of multiple data members (name-value pair)
= For eg. @Bean(name="----", initMethod="-----", destroyMethod="----")

==========

=> Annotations in Servlet :
-> In java servlets, annotations are used to simplify the configuration of servlets and other components in web application i.e. Filters, Listeners etc
-> Annotations provides a way to declare various settings and behaviour in the source code directly, rather than configuring them in web.xml file. This makes our code more concise and easier to maintain
-> Some commonly used annotations in servlet are :-
1. @WebServlet
2. @WebFilter
3. @WebListener
4. @WebInitParam
5. @MultipartConfig
etc

=> @WebServlet :-
-> This is an annotation used to define a servlet. We can specify the servlet's name, URL pattern and other configuration settings within this annotation
-> Syntax :-

```
@WebServlet(name="----", urlPatterns={----})
public class MyServlet extends HttpServlet
{
    //override the methods
}
```

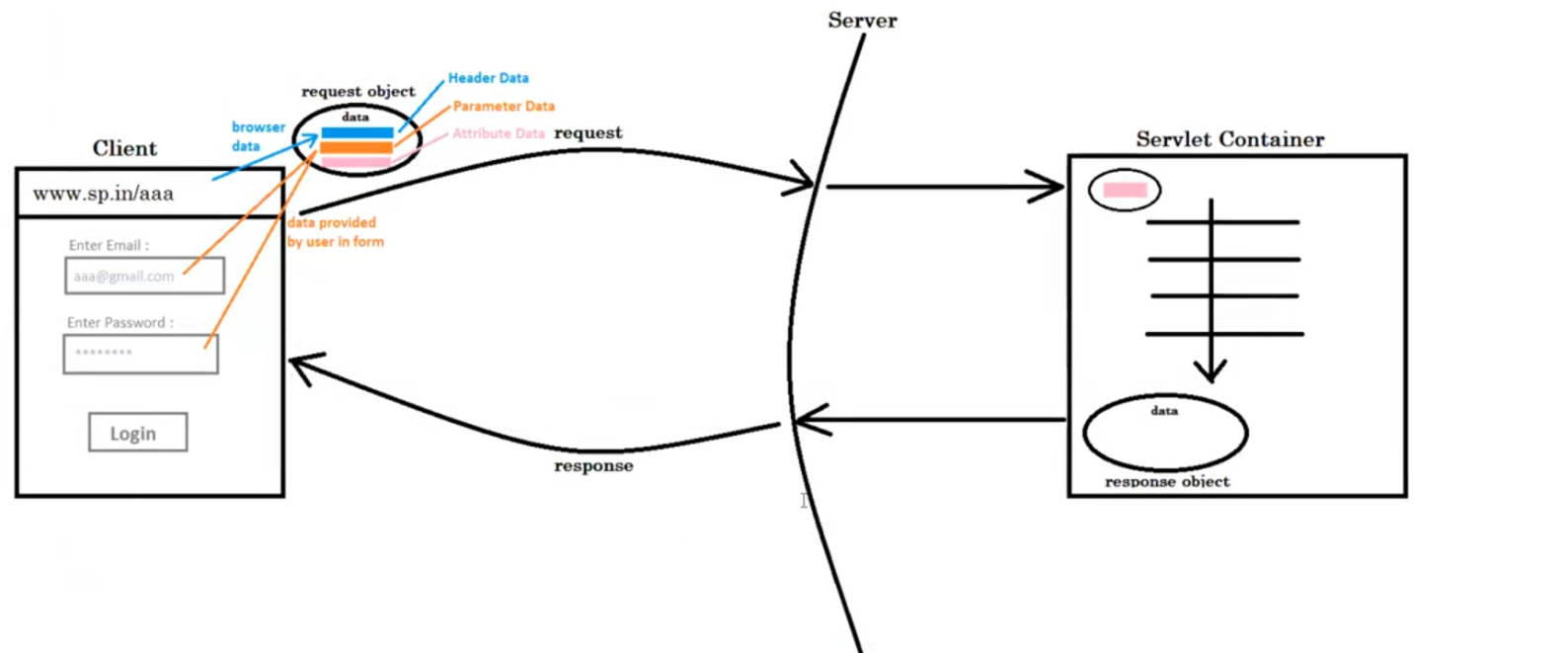
==========

```
=> Request Object :-
    -> It is an object that is transfered from client to server
    -> It contains 3 types of data :-
        1. Header Data
        2. Parameter Data
        3. Attribute Data
    -> Object data (metadata) is stored in the form of key-value pair for eg
        = name : deepak

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

1. Header Data :-
    -> It contains metadata (information) about the browser
    -> How to get headers data :-
        = public Enumeration getHeaderNames()
        = public String getHeader(String key)
        = public String[] getHeaders(String key)

-------------------------------------------------
```

2. Parameter Data :-
    -> The data which is provided by the user at browser form is parameter data
    -> How to get parameters data :-
        = public String getParameter(String key)
        = public Enumeration getParameterNames();
        = public String[] getParameterValues(String key)

---

3. Attribute Data :-
    -> The data provided by the programmer at backend part (servlets) after creation of request object
    -> How to set and get attribute data :-
        = public void setAttribute(String key, Object value)
        = public Object getAttribute(String key)
        = public void removeAttribute(String key)
        = public Enumeration getAttributeNames()


Server
request object
data
Header Data
Parameter Data
Attribute Data
browser
data
request
Client
www.sp.in/aaa
data provided
by user in form
Enter Email :
aaa@gmail.com
Enter Password :
****
Login
response
Servlet Container
data
response object

=> HttpServletRequest :-
-> It is an interface which is used to interact with incoming HTTP requests in servlet application
-> It provides methods to get the information about an incoming HTTP request like header data, parameters data etc
-> Some commonly used methods are :-
= public Enumeration getHeaderNames()
= public String getHeader(String key)

= public String getParameter(String key)

= public void setAttribute(String key, Object value)
= public Object getAttribute(String key)
= public void removeAttribute(String key)

= getSession()
= getCookies()

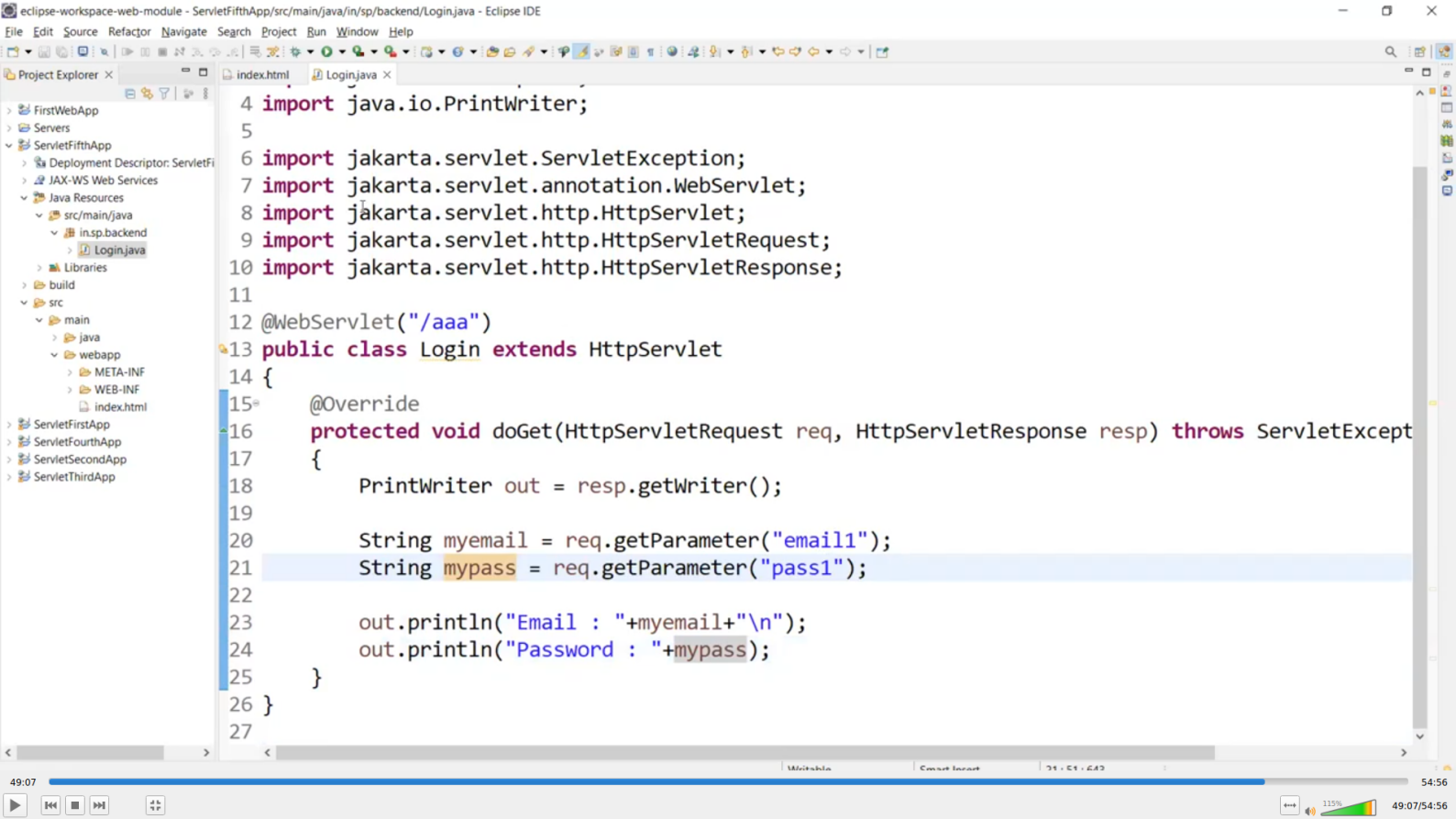
=> HttpServletResponse :-
    -> It is an interface which is used to interact with outgoing HTTP request in servlet application
    -> It provides methods to set attributes of an HTTP response such as response content, status code etc
    -> Some commonly used methods are :-
        = setStatus(int status)
        = sendRedirect(String location)

        = getWriter()
        = getOutputStream()

        = setContentType(String type)
        = setContentLength(int len)

==========

```
<!DOCTYPE html>
<html>

    <head>
        <meta charset="ISO-8859-1">
        <title>Insert title here</title>
    </head>

    <body>
        <h3> Login Here </h3>

        <form action="/aaa" method="GET">
            Enter Email :  <input type="text" name="email1" /> <br/>
            Enter Password : <input type="password" name="pass1" /> <br/>
            <input type="submit" value="Login" />
        </form>
    </body>

</html>
```


eclipse-workspace-web-module - ServletFifthApp/src/main/java/in/sp/backend/Login.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
FirstWebApp
Servers
ServletFifthApp
Deployment Descriptor: ServletFi
JAX-WS Web Services
Java Resources
src/main/java
in.sp.backend
Login.java
Libraries
build
src
main
java
webapp
META-INF
WEB-INF
index.html
ServletFirstApp
ServletFourthApp
ServletSecondApp
ServletThirdApp
index.html
Login.java
4 import java.io.PrintWriter;
5
6 import jakarta.servlet.ServletException;
7 import jakarta.servlet.annotation.WebServlet;
8 import jakarta.servlet.http.HttpServlet;
9 import jakarta.servlet.http.HttpServletRequest;
10 import jakarta.servlet.http.HttpServletResponse;
11
12 @WebServlet("/aaa")
13 public class Login extends HttpServlet
14 {
15 @Override
16 protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletExcept
17 {
18 PrintWriter out = resp.getWriter();
19
20 String myemail = req.getParameter("email1");
21 String mypass = req.getParameter("pass1");
22
23 out.println("Email : "+myemail+"\n");
24 out.println("Password : "+mypass);
25 }
26 }
27
49:07
54:56
49:07/54:56

```html
<!DOCTYPE html>
<html>

    <head>
        <meta charset="ISO-8859-1">
        <title>Insert title here</title>
    </head>

    <body>
        <h3> Login Here </h3>

        <form action="aaa" method="get">
            Enter Email :  <input type="text" name="email1" /> <br/> <br/>
            Enter Password : <input type="password" name="pass1" /> <br/> <br/>
            <input type="submit" value="Login" />
        </form>
    </body>

</html>
```

=> HTTP (Hyper Text Transfer Protocol) :-
-> HTTP is a protocol (set of rules and regulation) that defines how data is transmitted and formatted and how web servers and web browser should respond to various commands and requests
-> In simple terms, HTTP is a protocol that enables us to request and receive web pages, images, videos or any other resources when we browse the internet

=> HTTP Methods :-
-> HTTP methods are the commands or actions that indicate the desired action to be performed on a resource identified by a URL. These methods define the operations we want to carry out on a server then making an HTTP request
-> Some HTTP methods are :-
1. GET
2. POST
3. PUT
4. DELETE
5. HEAD
6. OPTIONS
7. PATCH
8. TRACE
etc

==========

=> Difference between GET and POST :-

1. Purpose :-
   = GET is used to retrieve the data from server. It does not perform any change at server side.
   = POST is used to send data to the server for processing. It performs some changes at server side.

2. How data is transfered :-
   = In case of GET, data is transfered to the server through URL.
   = In case of POST, data is transfered to the server using request body.

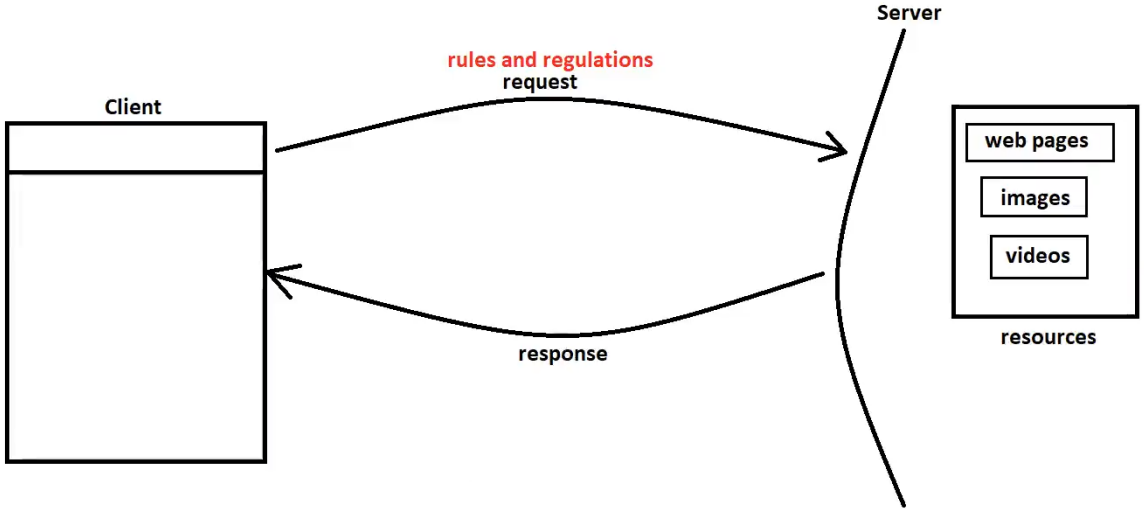
3. Security :-
   = GET is less secured because data is visible in the URL.
   = POST is more secured because data is transfered through request body which is not visible.

4. Amount of data :-
   = We cannot send large amount of data using GET because it transfers the data using URL.
   = We can send large amount of data (images, files, videos etc) using POST because it transfers the data using request body.

5. Examples :-
   = GET :-
      - Fetching web pages
      - Performing the search query
        etc
   = POST :-
      - Submitting the form
      - Uploading any file, images or videos
      - Making payment through website
        etc

6. Bookmarked :-
   = GET request can be bookmarked
   = POST request cannot be bookmarked

7. Caching :-
   = GET requests can be cached by browsers and intermediary systems. This can improvide the performace in some cases
   = POST requests cannot be cached

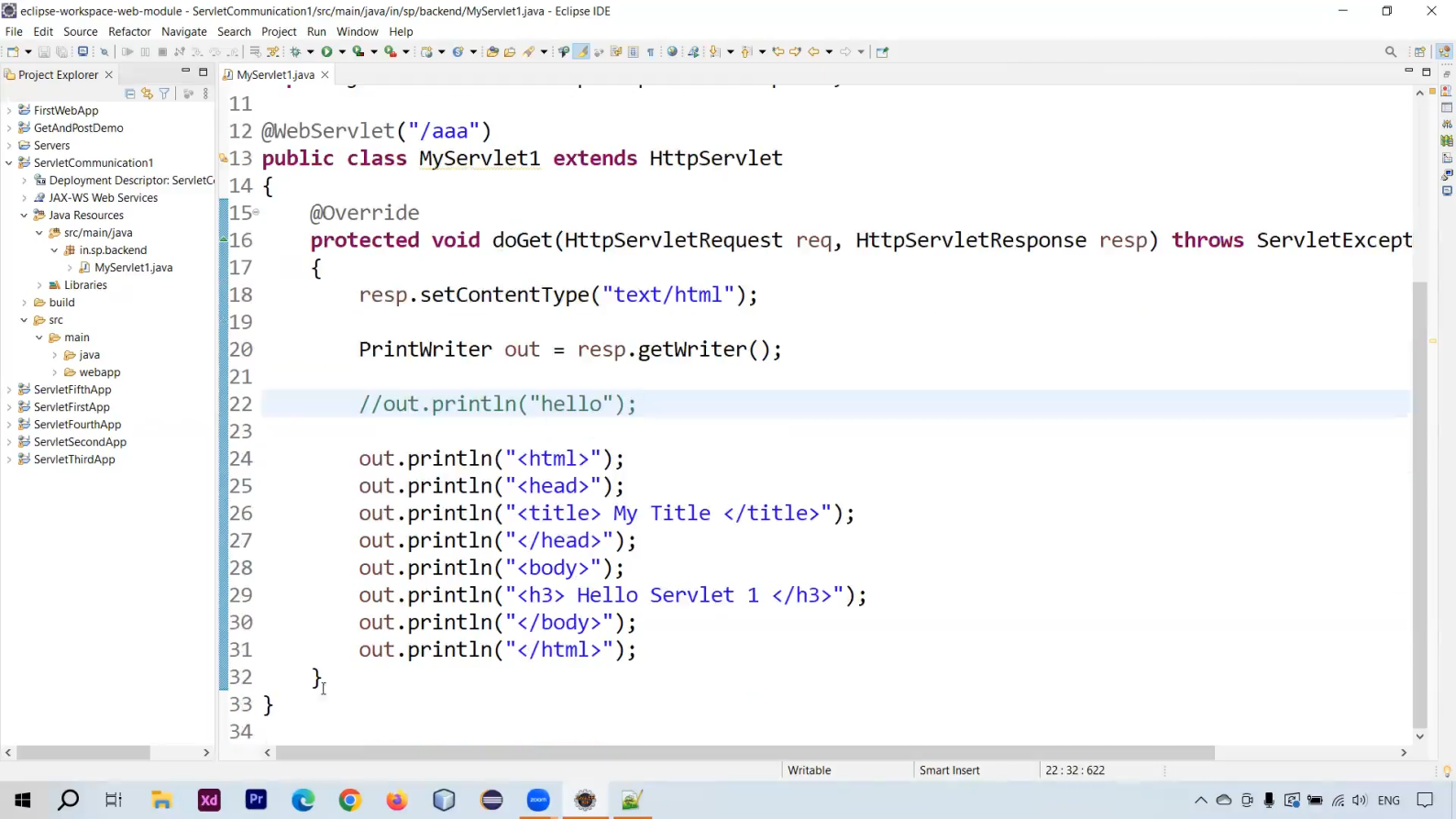

Server
rules and regulations
request
Client
web pages
images
videos
resources
response

=> Servlet Communication :-
    -> It is the process by which servlet communicates with others i.e. browsers, web components (html, servlet, JSP etc) etc
    -> Types of Servlet Communication :-
        1. Servlet Communication with Browser
            -> Direct communication : Interacting with the browser directly
        2. Servlet Communication with Web Components
            -> Indirect communication : Facilitating the communication between web components i.e. HTML, servlet, JSP etc

---

=> Servlet Communication with Browser :-
    -> Servlet can communicate with browser by 3 ways :-
        1. Through request-response objects
        2. Through sendError() method of HttpServletResponse
        3. Request redirection
            3.1 Request redirection by hyperlinks
            3.2 Request redirection by setting header methods (setStatus() & setHeader())
            3.3 Request redirection by using sendRedirect() method of HttpServletResponse

---


eclipse-workspace-web-module - ServletCommunication1/src/main/java/in/sp/backend/MyServlet1.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
FirstWebApp
GetAndPostDemo
Servers
ServletCommunication1
Deployment Descriptor: ServletC
JAX-WS Web Services
Java Resources
src/main/java
in.sp.backend
MyServlet1.java
Libraries
build
src
main
java
webapp
ServletFifthApp
ServletFirstApp
ServletFourthApp
ServletSecondApp
ServletThirdApp
MyServlet1.java
11
12 @WebServlet("/aaa")
13 public class MyServlet1 extends HttpServlet
14 {
15 @Override
16 protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletExcept
17 {
18 resp.setContentType("text/html");
19
20 PrintWriter out = resp.getWriter();
21
22 //out.println("hello");
23
24 out.println("<html>");
25 out.println("<head>");
26 out.println("<title> My Title </title>");
27 out.println("</head>");
28 out.println("<body>");
29 out.println("<h3> Hello Servlet 1 </h3>");
30 out.println("</body>");
31 out.println("</html>");
32 }
33 }
34
Writable
Smart Insert
22 : 32 : 622
ENG

HTTP Status 404 – Not Found

Type Status Report

Message Page not found, please try again later..!! (this is custom message)

Description The origin server did not find a current representation for the target resource or is not willing to disclose that one exists.

Apache Tomcat/10.1.13

```java
@WebServlet("/aaa")
public class MyServlet1 extends HttpServlet
{
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletExcept
    {
        resp.sendError(404,"Page not found, please try again later..!! (this is custom message
    }
}
```

```java
@WebServlet("/aaa")
public class MyServlet1 extends HttpServlet
{
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletExcept
    {
        resp.setStatus(HttpServletResponse.SC_FOUND);
        resp.setHeader("Location", "https://www.google.com");
    }
}
```

```java
@WebServlet("/aaa")
public class MyServlet1 extends HttpServlet
{
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletExcept
    {
        resp.sendRedirect("https://www.google.com");
    }
}
```

```
--------------------------------------------------------------------------------

=> Servlet Communication with Web Components :-
    -> Servlet can communicate with web components by 2 ways :-
        1. Using forward() method
        2. Using include() method

    -> NOTE : forward() and include() method are present in RequestDispatcher interface

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

=> NOTE :
    = Request redirection is used for external application redirection but forward() and include() are used for
    internal application redirection

================================================================================================================
```

```java
@WebServlet("/bbb")
public class MyServlet2 extends HttpServlet
{
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletExcept
    {
        resp.getWriter().println("Hello Servlet 2");
    }
}
```

```
@WebServlet("/aaa")
public class MyServlet1 extends HttpServlet
{
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletExcept
    {
//      RequestDispatcher rd = req.getRequestDispatcher("/newPage.html");
//      rd.forward(req, resp);


        RequestDispatcher rd = req.getRequestDispatcher("/bbb");
        rd.forward(req, resp);
    }
}
```

=> What is difference between sendRedirect() and RequestDispatcher :-

1. sendRedirect() is used for external request redirection
   RequestDispatcher is used for internal request redirection

2. sendRedirect() redirects the request to a different application or URL
   RequestDispatcher is used to forward or include the request to the same application or URL

3. sendRedirect() is the method of HttpServletResponse
   RequestDispatcher has 2 methods i.e. forward() and include()

4. sendRedirect() will change the URL on the browser
   RequestDispatcher does not change the URL on the browser

=> HTTP (Hypertext Trasfer Protocol) :-
    -> It is fundamental protocol for communication over the network. It enables data exchange between client and server
    -> NOTE :-
        = HTTP is text-based protocol and it operates over the TCP/IP network
        = HTTP is stateless protocol. It means that server is not able to judge whether the request is coming from single client or multiple client

=> Session :-
    -> It is a period during which a client interacts with the server
    -> It keeps the track of user's action, state and data during this time

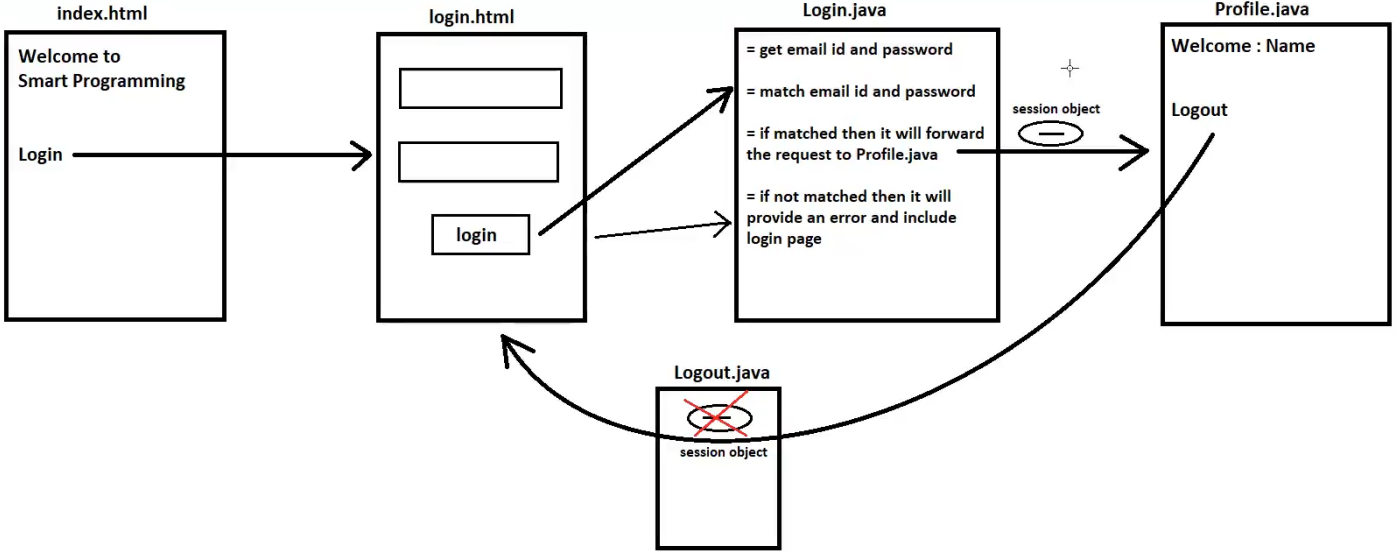
=> What is "State" and "Data" :-
    -> State :-
        = State refers to the ability to maintain information about a client's interaction with the server ober multiple HTTP requests. It allows us to remember the client's identity and track their action across different pages or requests
        = For example :-
            1. User Authentication State :
                = Wheher the user is logged in or logged out
            2. Shopping Cart State :-
                = Wheher the user added some items to cart or not
            3. Form Progress State :-
                = When a user fills out a multip-step form, the step we are currently on, along with the data we have entered is the part of state. This allows us to resume where we left off
            4. Game Progress State :-
                = In online games, the level or stage a player has reached is the part of state
            5. Video Playback State :-
                = In video streaming services, the user's progress in a video (i.e. where we have stopped watching or any bookmarks we have added) represents the state
            etc

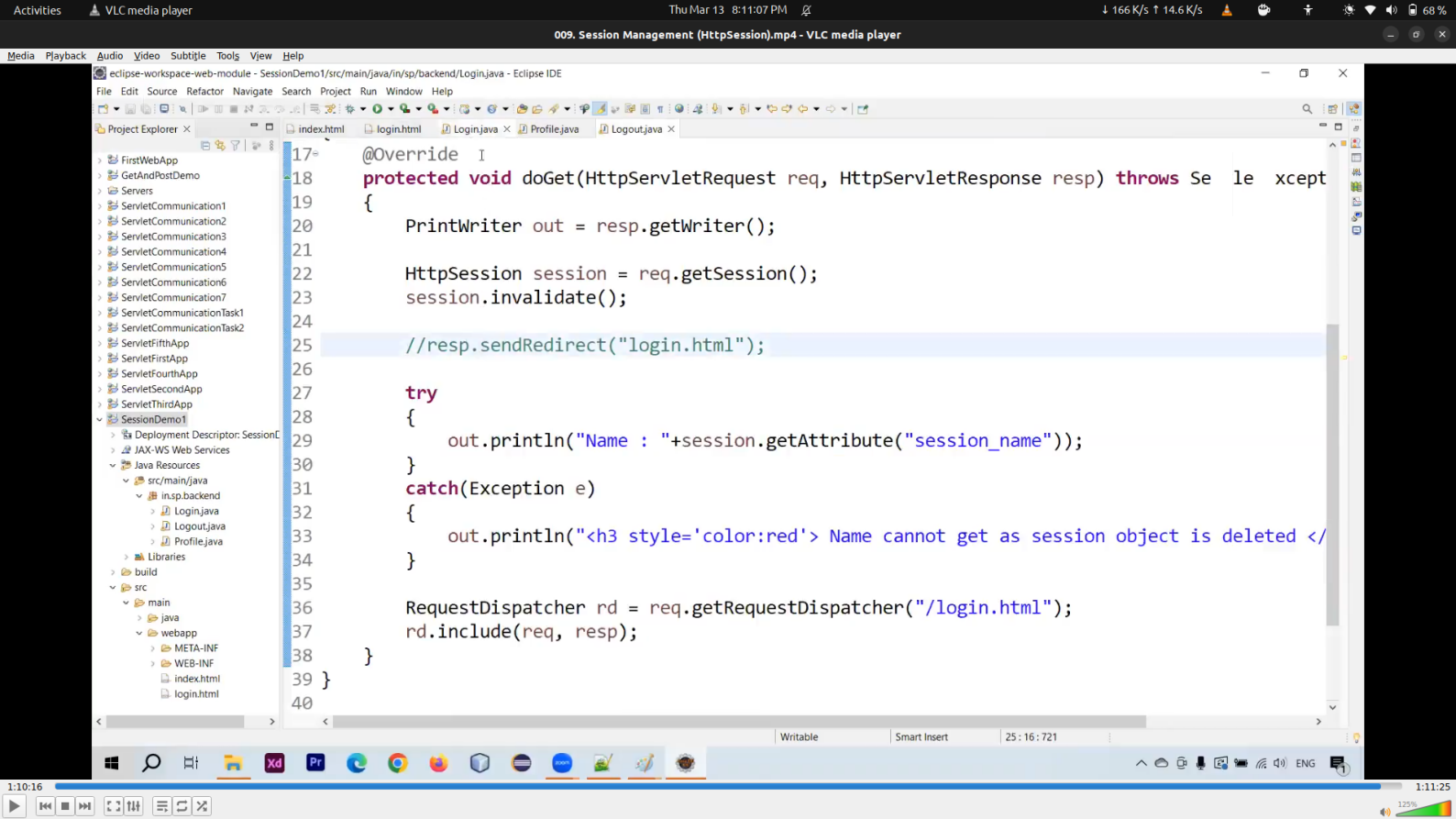
-> Data :-
    = Data refers to the users information that has been tracked
    = For example :-
        1. User Profile Data :-
            - User specific details i.e. name, email, profile pic etc
        2. Shopping Cart Data :-
            - The details of items in shooping cart i.e. item name, item price, item quantity etc
        3. Session History Data :-
            - A record of users recent interactions i.e. browsing history, search queries, viewing products etc
        4. Form Input Data :-
            - Data entered by the user in web form during their session
        5. User Preferences Data :-
            - Specific settings and preferences choosed by the user i.e. saved address, saved payment modes etc

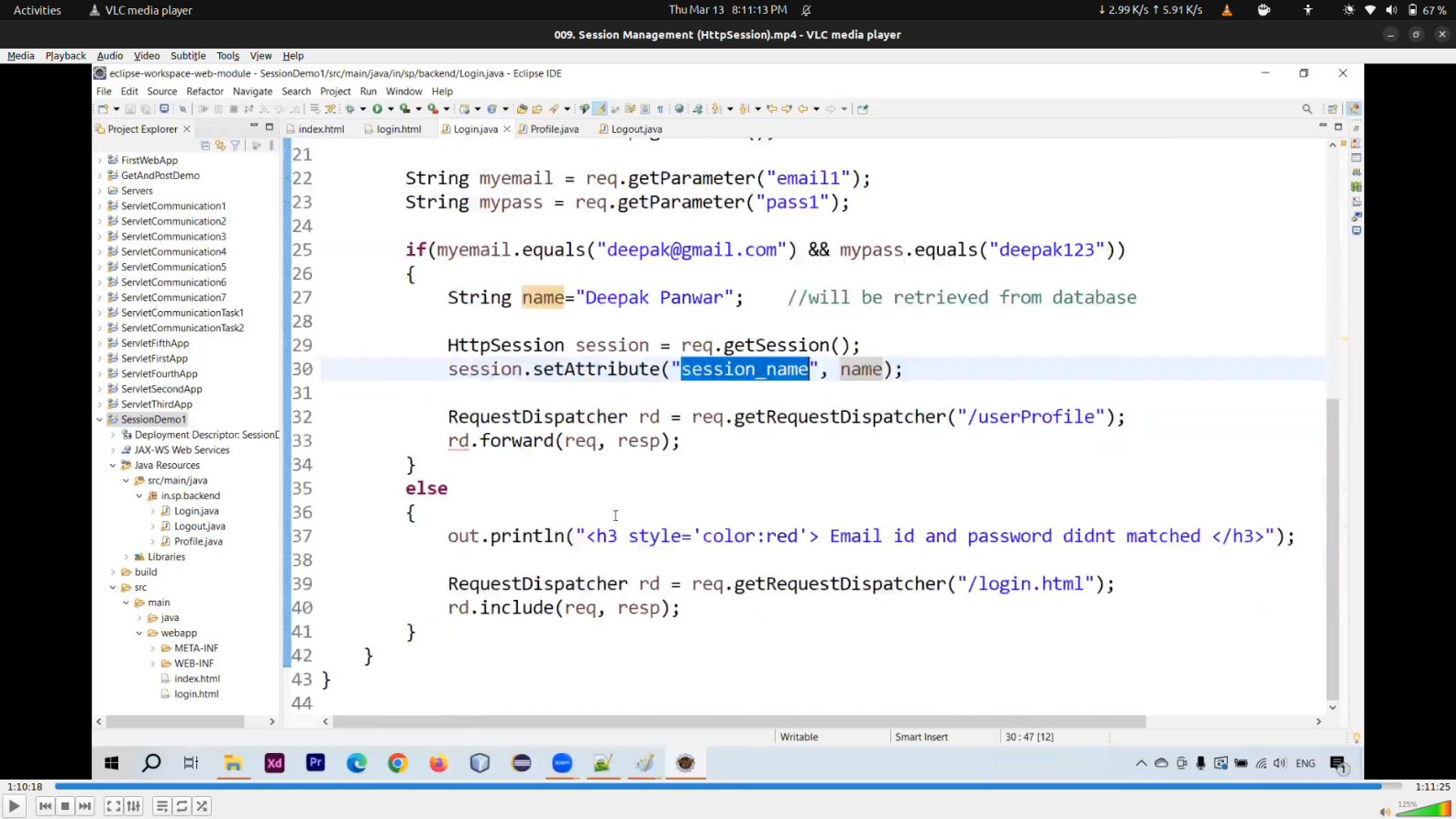
=> Session Management :-
-> Session Management is the process of maintaining the state and data for individual users during their interaction with the server (web application)
-> We can achieve session management by "HttpSession Objects" (In conjection with session tracking mechanisms like cookies, URL rewriting, hidden form fields etc)

=> Session Tracking Mechanism :-
-> It is the technique by which server tracks whether the request is coming from one client or not
-> We can achieve it by :-
1. Cookies
2. URL Rewriting
3. Hidden Form Fields
etc

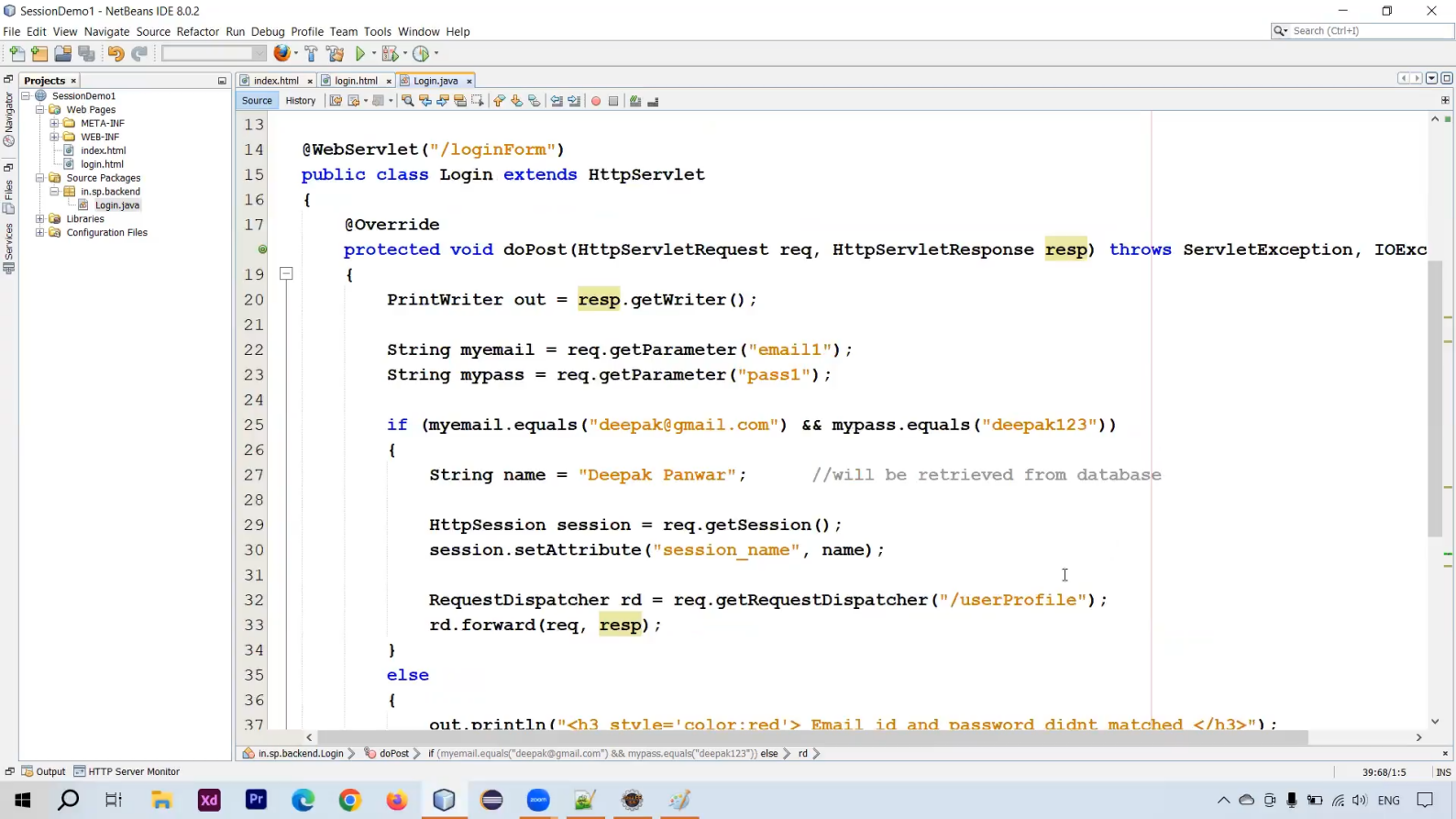
```
=> HttpSession :-
    -> HttpSession is an interface provided in Java Servlet API
    -> It's implementation is provided by the "web containers" for eg Apache Tomcat etc
    -> While it is referred as an interface, it is used to represent an object that provides the session management in
    servlet-based applications


=> HttpSession Object :-
    -> How to create/get HttpSession Object :-
        1. public HttpSession getSession()
        2. public HttpSession getSession(boolean b)
           (Above methods are present in "HttpServletRequest" interface)

    -> How to handle the data in HttpSession Object :-
        1. public void setAttribute(String name, Object value)
        2. public Object getAttribute(String name)
        3. public Enumeration getAttributeNames()
        4. public void removeAttributes(String name)

    -> How to destroy the HttpSession Object :-
        1. public void invalidate()
        2. public void setMaxInactiveInterval(int time)
```

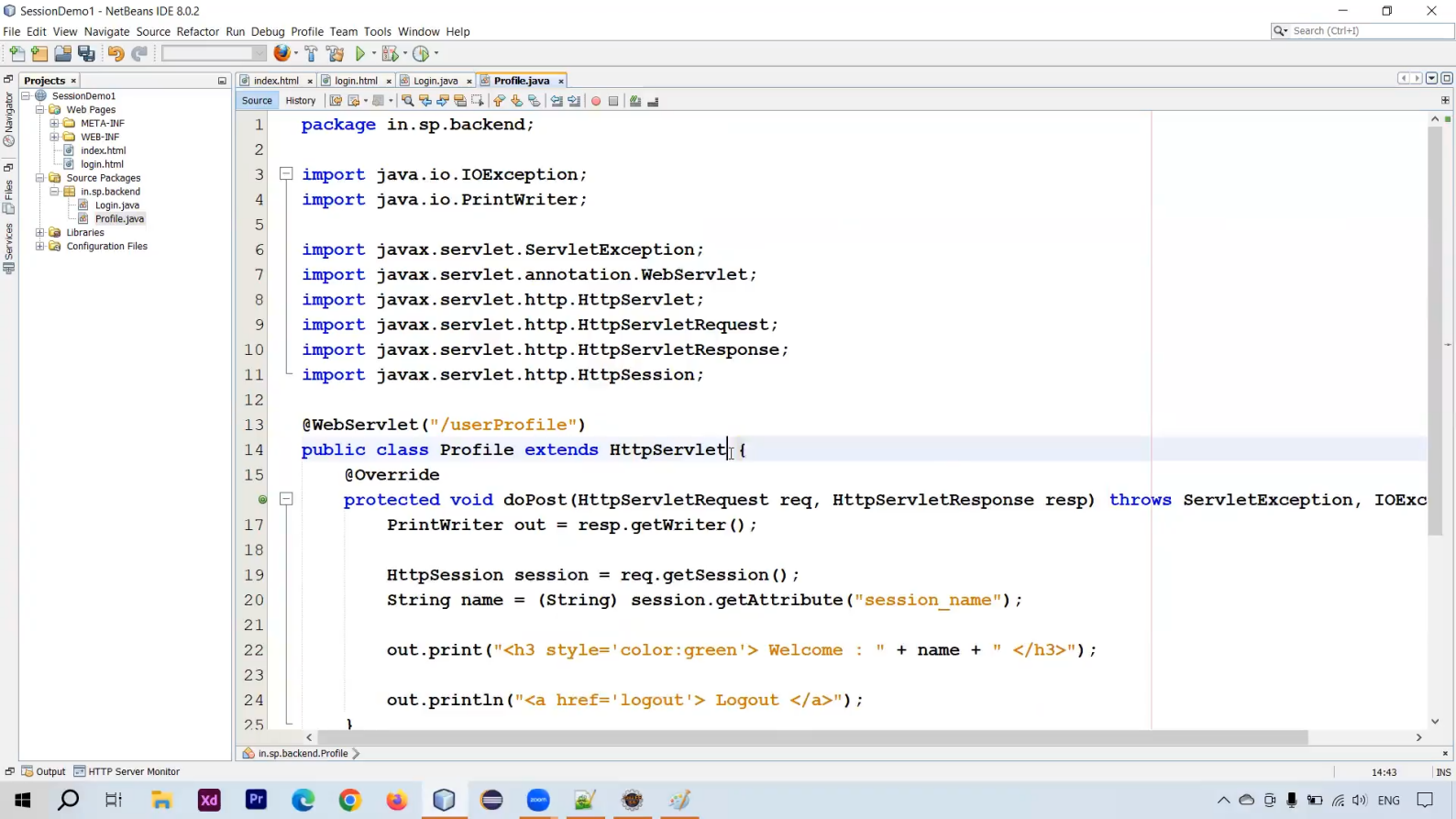

index.html
Welcome to
Smart Programming
Login
login.html
login
Login.java
= get email id and password
= match email id and password
= if matched then it will forward the request to Profile.java
= if not matched then it will provide an error and include login page
session object
Profile.java
Welcome : Name
Logout
Logout.java
session object

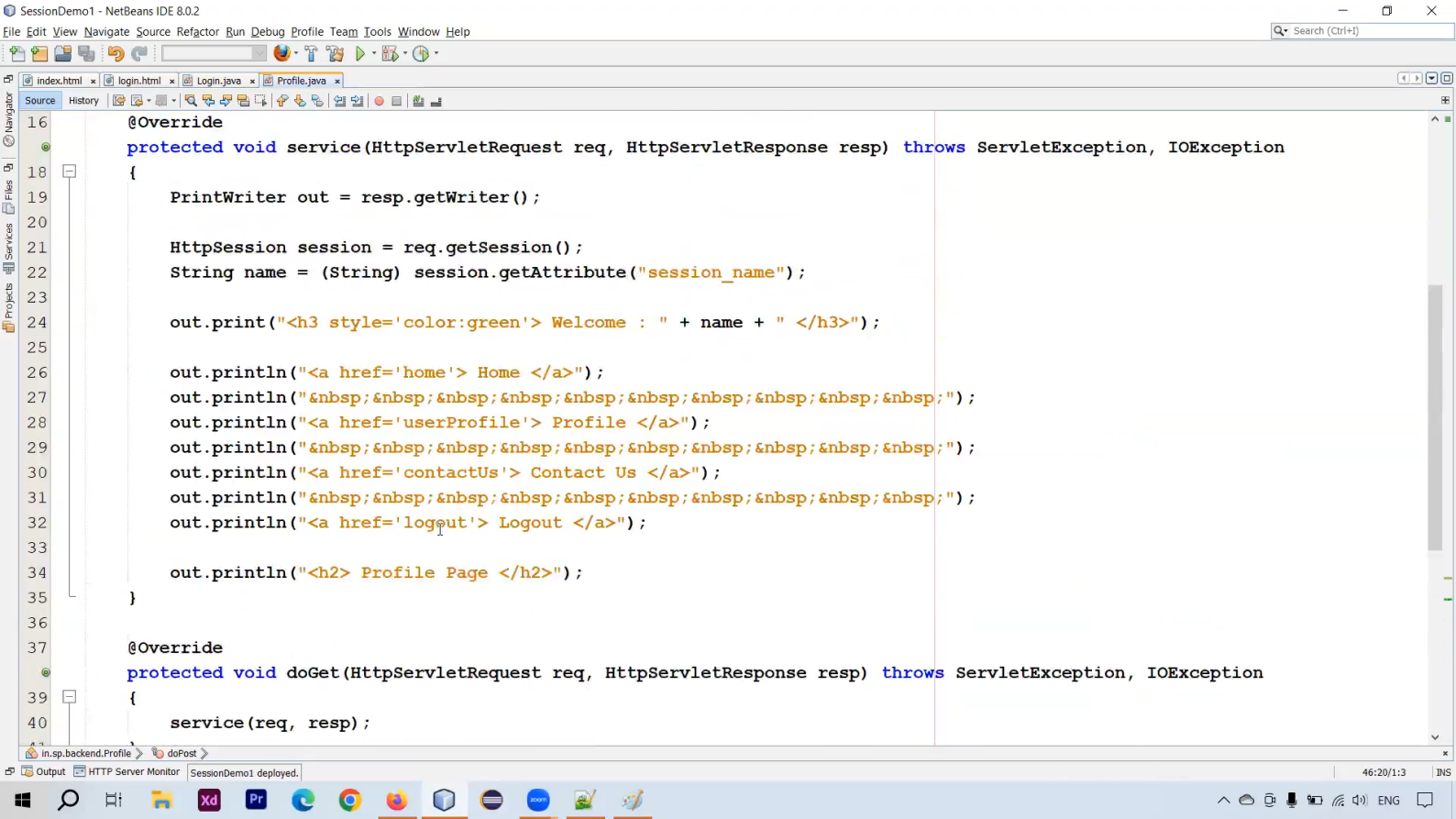

Activities
VLC media player
Thu Mar 13 8:11:07 PM
009. Session Management (HttpSession).mp4 - VLC media player
Media Playback Audio Video Subtitle Tools View Help
eclipse-workspace-web-module - SessionDemo1/src/main/java/in/sp/backend/Login.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
index.html
login.html
Login.java
Profile.java
Logout.java
FirstWebApp
GetAndPostDemo
Servers
ServletCommunication1
ServletCommunication2
ServletCommunication3
ServletCommunication4
ServletCommunication5
ServletCommunication6
ServletCommunication7
ServletCommunicationTask1
ServletCommunicationTask2
ServletFifthApp
ServletFirstApp
ServletFourthApp
ServletSecondApp
ServletThirdApp
SessionDemo1
Deployment Descriptor: SessionD
JAX-WS Web Services
Java Resources
src/main/java
in.sp.backend
Login.java
Logout.java
Profile.java
Libraries
build
src
main
java
webapp
META-INF
WEB-INF
index.html
login.html
@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws Se le xcept
{
PrintWriter out = resp.getWriter();
HttpSession session = req.getSession();
session.invalidate();
//resp.sendRedirect("login.html");
try
{
out.println("Name : "+session.getAttribute("session_name"));
}
catch(Exception e)
{
out.println("<h3 style='color:red'> Name cannot get as session object is deleted </
}
RequestDispatcher rd = req.getRequestDispatcher("/login.html");
rd.include(req, resp);
}
}
Writable
Smart Insert
25 : 16 : 721
ENG
1:10:16
1:11:25

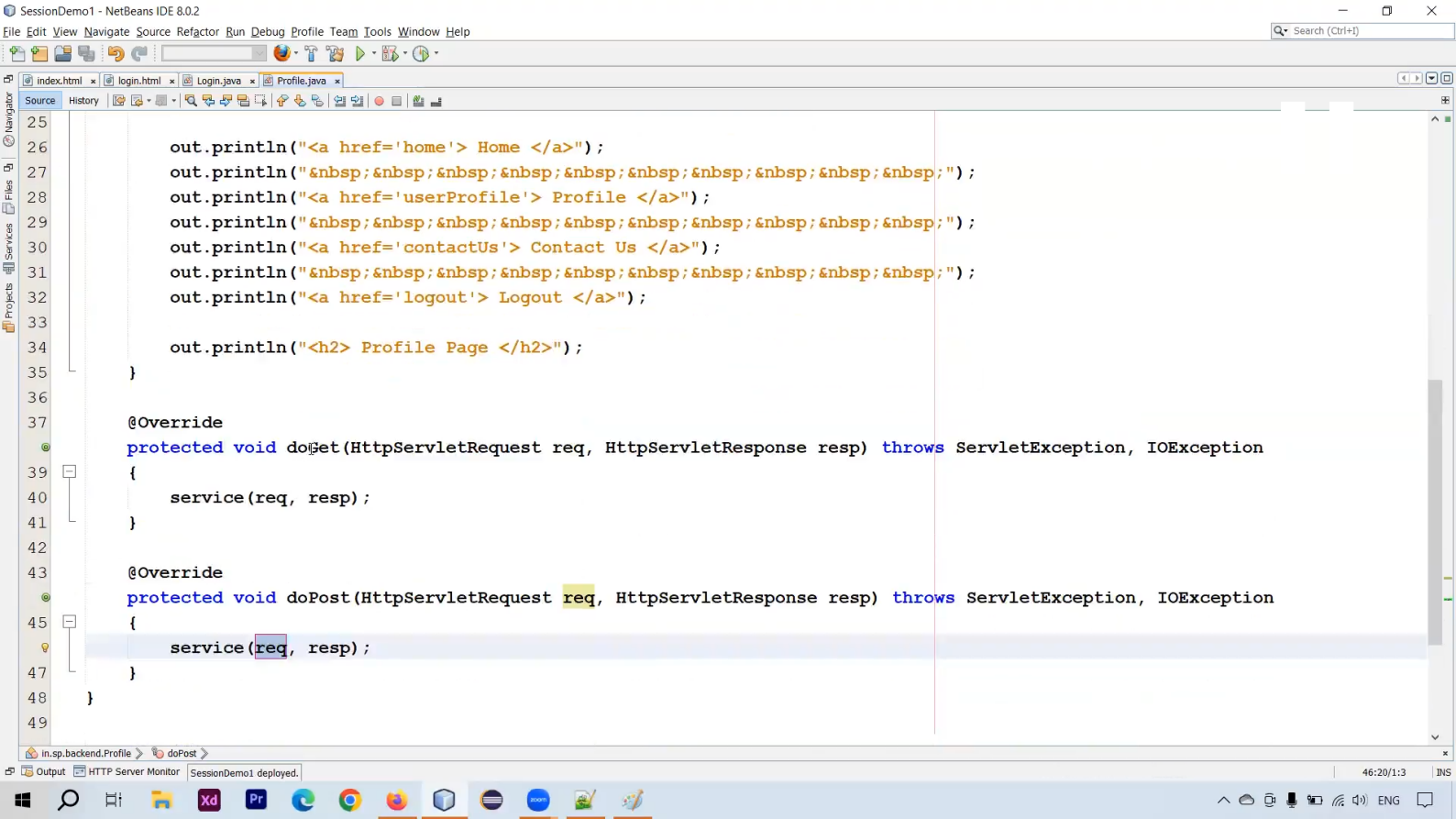

Activities
VLC media player
Thu Mar 13 8:11:13 PM
009. Session Management (HttpSession).mp4 - VLC media player
Media Playback Audio Video Subtitle Tools View Help
eclipse-workspace-web-module - SessionDemo1/src/main/java/in/sp/backend/Login.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
index.html
login.html
Login.java
Profile.java
Logout.java
FirstWebApp
GetAndPostDemo
Servers
ServletCommunication1
ServletCommunication2
ServletCommunication3
ServletCommunication4
ServletCommunication5
ServletCommunication6
ServletCommunication7
ServletCommunicationTask1
ServletCommunicationTask2
ServletFifthApp
ServletFirstApp
ServletFourthApp
ServletSecondApp
ServletThirdApp
SessionDemo1
Deployment Descriptor: SessionD
JAX-WS Web Services
Java Resources
src/main/java
in.sp.backend
Login.java
Logout.java
Profile.java
Libraries
build
src
main
java
webapp
META-INF
WEB-INF
index.html
login.html
String myemail = req.getParameter("email1");
String mypass = req.getParameter("pass1");
if(myemail.equals("deepak@gmail.com") && mypass.equals("deepak123"))
{
String name="Deepak Panwar"; //will be retrieved from database
HttpSession session = req.getSession();
session.setAttribute("session_name", name);
RequestDispatcher rd = req.getRequestDispatcher("/userProfile");
rd.forward(req, resp);
}
else
{
out.println("<h3 style='color:red'> Email id and password didnt matched </h3>");
RequestDispatcher rd = req.getRequestDispatcher("/login.html");
rd.include(req, resp);
}
}
}
Writable
Smart Insert
30 : 47 [12]
ENG
1:10:18
1:11:25

Untitled - Paint
File
Home
View
Cut
Copy
Paste
Clipboard
Crop
Resize
Rotate
Select
Image
Brushes
Tools
Outline
Fill
Shapes
Size
Color 1
Color 2
Edit colors
Edit with Paint 3D
Colors
Profile.java
Home Profile Contact Us Logout
Welcome : Name
I am in Profile Page
index.html
login.html
Login.java
Home.java
Home Profile Contact Us Logout
Welcome : Name
I am in Home Page
ContactUs.java
Home Profile Contact Us Logout
Welcome : Name
I am in Contact Us Page
1902 × 814px
100%
ENG


SessionDemo1 - NetBeans IDE 8.0.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
Projects
SessionDemo1
Web Pages
META-INF
WEB-INF
index.html
login.html
Source Packages
in.sp.backend
Login.java
Libraries
Configuration Files
index.html
login.html
Login.java
Source
History
@WebServlet("/loginForm")
public class Login extends HttpServlet
{
@Override
protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOExc
{
PrintWriter out = resp.getWriter();
String myemail = req.getParameter("email1");
String mypass = req.getParameter("pass1");
if (myemail.equals("deepak@gmail.com") && mypass.equals("deepak123"))
{
String name = "Deepak Panwar"; //will be retrieved from database
HttpSession session = req.getSession();
session.setAttribute("session_name", name);
RequestDispatcher rd = req.getRequestDispatcher("/userProfile");
rd.forward(req, resp);
}
else
{
out.println("<h3 style='color:red'> Email id and password didnt matched </h3>");
in.sp.backend.Login
doPost
if (myemail.equals("deepak@gmail.com") && mypass.equals("deepak123")) else
rd
Output
HTTP Server Monitor
39:68/1:5
INS
ENG


SessionDemo1 - NetBeans IDE 8.0.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
Projects
SessionDemo1
Web Pages
META-INF
WEB-INF
index.html
login.html
Source Packages
in.sp.backend
Login.java
Profile.java
Libraries
Configuration Files
index.html
login.html
Login.java
Profile.java
Source
History
1 package in.sp.backend;
2
3 import java.io.IOException;
4 import java.io.PrintWriter;
5
6 import javax.servlet.ServletException;
7 import javax.servlet.annotation.WebServlet;
8 import javax.servlet.http.HttpServlet;
9 import javax.servlet.http.HttpServletRequest;
10 import javax.servlet.http.HttpServletResponse;
11 import javax.servlet.http.HttpSession;
12
13 @WebServlet("/userProfile")
14 public class Profile extends HttpServlet {
15 @Override
protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOExc
17 PrintWriter out = resp.getWriter();
18
19 HttpSession session = req.getSession();
20 String name = (String) session.getAttribute("session_name");
21
22 out.print("<h3 style='color:green'> Welcome : " + name + " </h3>");
23
24 out.println(" Logout ");
25 }
in.sp.backend.Profile
Output
HTTP Server Monitor
14:43
INS
ENG


SessionDemo1 - NetBeans IDE 8.0.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
index.html
login.html
Login.java
Profile.java
Source
History
@Override
protected void service(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException
{
PrintWriter out = resp.getWriter();
HttpSession session = req.getSession();
String name = (String) session.getAttribute("session_name");
out.print("<h3 style='color:green'> Welcome : " + name + " </h3>");
out.println(" Home ");
out.println("   ");
out.println(" Profile ");
out.println("   ");
out.println(" Contact Us ");
out.println("   ");
out.println(" Logout ");
out.println("<h2> Profile Page </h2>");
}
@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException
{
service(req, resp);
in.sp.backend.Profile
doPost
Output
HTTP Server Monitor
SessionDemo1 deployed.
46:20/1:3
INS
ENG


SessionDemo1 - NetBeans IDE 8.0.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
index.html
login.html
Login.java
Profile.java
Source
History
out.println(" Home ");
out.println("   ");
out.println(" Profile ");
out.println("   ");
out.println(" Contact Us ");
out.println("   ");
out.println(" Logout ");
out.println("<h2> Profile Page </h2>");
}
@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException
{
service(req, resp);
}
@Override
protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException
{
service(req, resp);
}
}
in.sp.backend.Profile
doPost
Output
HTTP Server Monitor
SessionDemo1 deployed.
46:20/1:3
INS
ENG

SessionDemo1 - NetBeans IDE 8.0.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

index.html | login.html | Login.java | Profile.java | Home.java | ContactUs.java | Logout.java

Source History

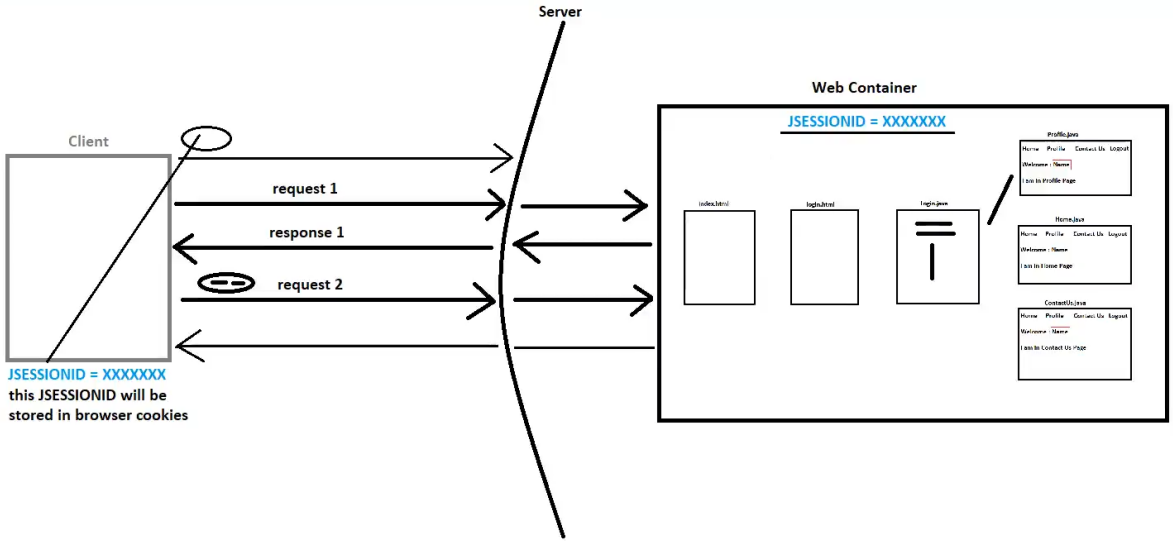
```
13  @WebServlet("/logout")
14  public class Logout extends HttpServlet
15  {
16      @Override
        protected void service(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException
18      {
19          HttpSession session = req.getSession();
20
21          session.invalidate();
22
23          resp.sendRedirect("login.html");
24      }
25
26      @Override
        protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException
28      {
29          service(req, resp);
30      }
31
32      @Override
        protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException
34      {
35          service(req, resp);
36      }
37  }
```

in.sp.backend.Logout > service

Output | HTTP Server Monitor | SessionDemo1 deployed.

23:41 INS

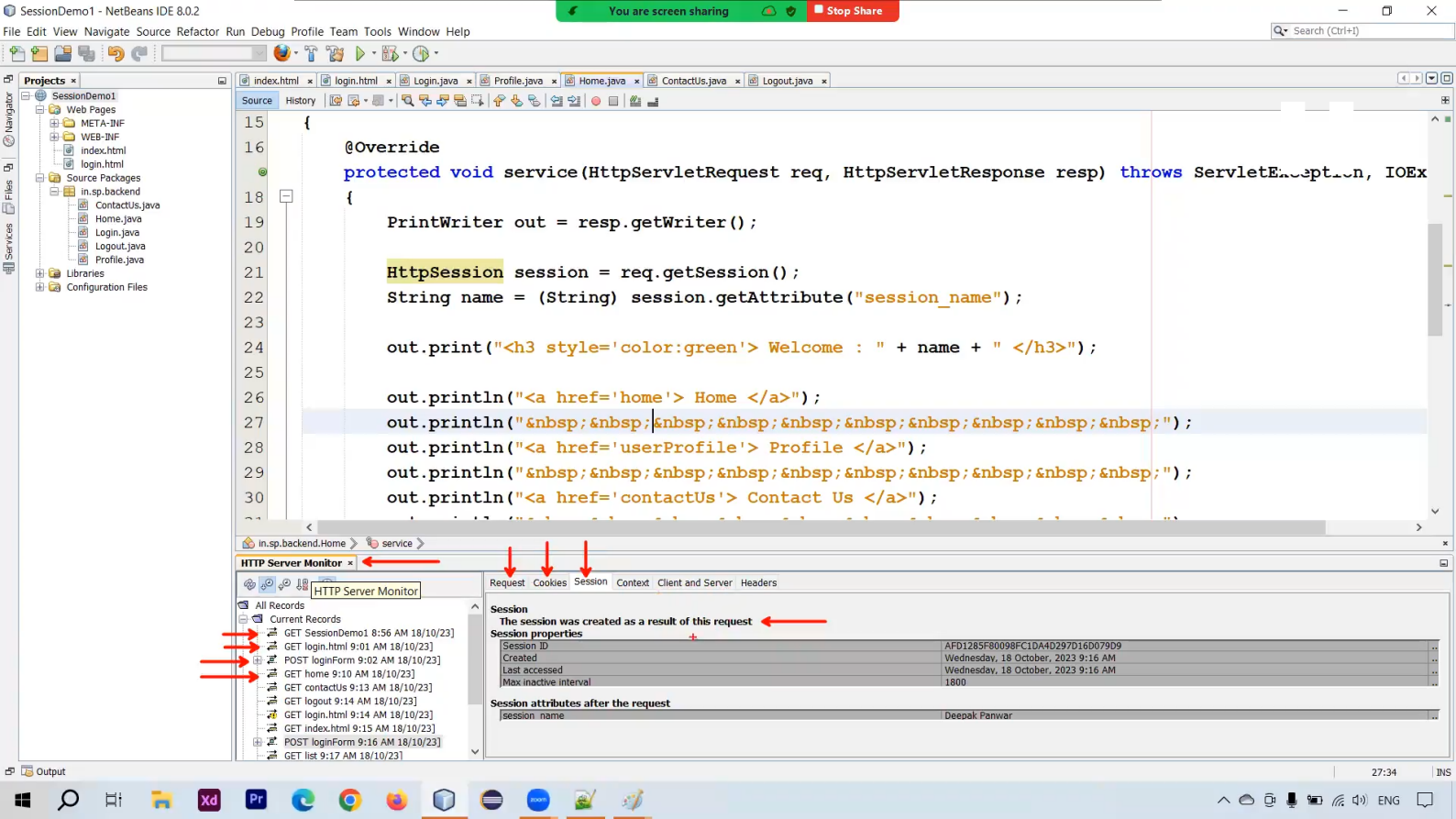
ENG


Server
Web Container
JSESSIONID = XXXXXXX
Client
request 1
response 1
request 2
index.html
login.html
login.jsp
Profile.jsp
Home.jsp
ContactUs.jsp
JSESSIONID = XXXXXXX
this JSESSIONID will be
stored in browser cookies

=> Internal working of Session Management :-
-> When we achieve Session Management using HttpSession, internally cookies (Session Tracking Mechanism ar used
-> But there is one disadvantage of cookies i.e. client can disable the cookies in browser

-> In case of session management, we container creates one session id i.e. JSESSIONID. This JSESSIONID is transfered from client to server in request header body. This JSESSIONID is matched by web container and it get to know that the client is new or old

==========


SessionDemo1 - NetBeans IDE 8.0.2
You are screen sharing
Stop Share
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
Projects
SessionDemo1
Web Pages
META-INF
WEB-INF
index.html
login.html
Source Packages
in.sp.backend
ContactUs.java
Home.java
Login.java
Logout.java
Profile.java
Libraries
Configuration Files
index.html
login.html
Login.java
Profile.java
Home.java
ContactUs.java
Logout.java
Source
History
15 {
16 @Override
protected void service(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOEx
18 {
19 PrintWriter out = resp.getWriter();
21 HttpSession session = req.getSession();
22 String name = (String) session.getAttribute("session_name");
24 out.print("<h3 style='color:green'> Welcome : " + name + " </h3>");
26 out.println(" Home ");
27 out.println("   ");
28 out.println(" Profile ");
29 out.println("   ");
30 out.println(" Contact Us ");
in.sp.backend.Home
service
HTTP Server Monitor
HTTP Server Monitor
All Records
Current Records
GET SessionDemo1 8:56 AM 18/10/23]
GET login.html 9:01 AM 18/10/23]
POST loginForm 9:02 AM 18/10/23]
GET home 9:10 AM 18/10/23]
GET contactUs 9:13 AM 18/10/23]
GET logout 9:14 AM 18/10/23]
GET login.html 9:14 AM 18/10/23]
GET index.html 9:15 AM 18/10/23]
POST loginForm 9:16 AM 18/10/23]
GET list 9:17 AM 18/10/23]
Request
Cookies
Session
Context
Client and Server
Headers
Session
The session was created as a result of this request
Session properties
Session ID
AFD1285F80098FC1DA4D297D16D079D9
Created
Wednesday, 18 October, 2023 9:16 AM
Last accessed
Wednesday, 18 October, 2023 9:16 AM
Max inactive interval
1800
Session attributes after the request
session_name
Deepak Panwar
Output
27:34
INS
ENG

=> Cookie :-
-> Cookie is a small piece of data (object) that is stored on the client side i.e browser and sent back to the server with each subsequent request
-> Cookie stores the data in "name-value" pair, for eg.
= Expires : Sat, 01 Jan 2025 00:00:00 GMT
= Secure : true
= HttpOnly : true

-> There are 2 types of cookies :-
1. Temporary Cookie (Session Cookie)
= These cookies are available for the duration of user's browser session. When the user closes their browser, temporary cookies gets deleted.
= These cookies are stored in browser's memory (RAM)
2. Permanent Cookie (Persistent Cookie)
= These cookies are available till their expiration date even when the user closes the browser.
= These cookies are stored in the file on user's computer i.e. hard disk

-> Use of cookies :-
1. Session Management
2. Remember Me / Login Credentials
3. Add to cart in shopping application
4. Language preferences
etc

-> Advantages of cookies :-
   1. It is the simplest way to achieve session tracking mechanism
   2. It is managed at client side

-> Disadvantages of cookies :-
   1. If the cookies are disabled at client side then session tracking mechanism will not work
   2. Cookies are not much secured
   3. We can store only textual information in cookies

```
=> Cookie :-
    -> Cookie is the class which is present in "javax.servlet.http" package ("jakarta.servlet.http" for newer version)

    -> Cookie has one constructor i.e.
        = Cookie(String name, String value)

    -> How to add cookie in response object :-
        = public void addCookie(Cookie c)
          (This method is present in HttpServletResponse interface)

    -> How to get cookie from request object :-
        = public Cookie[] getCookies()
          (This method is present in HttpServletRequest interface)

    -> Methods of Cookie class :-
        1. Get name and value from cookie :-
            = public String getName()
            = public String getValue()

        2. Cookie age or expiry date :-
            = public void setMaxAge(int sec)
            = public int getMaxAge()
```

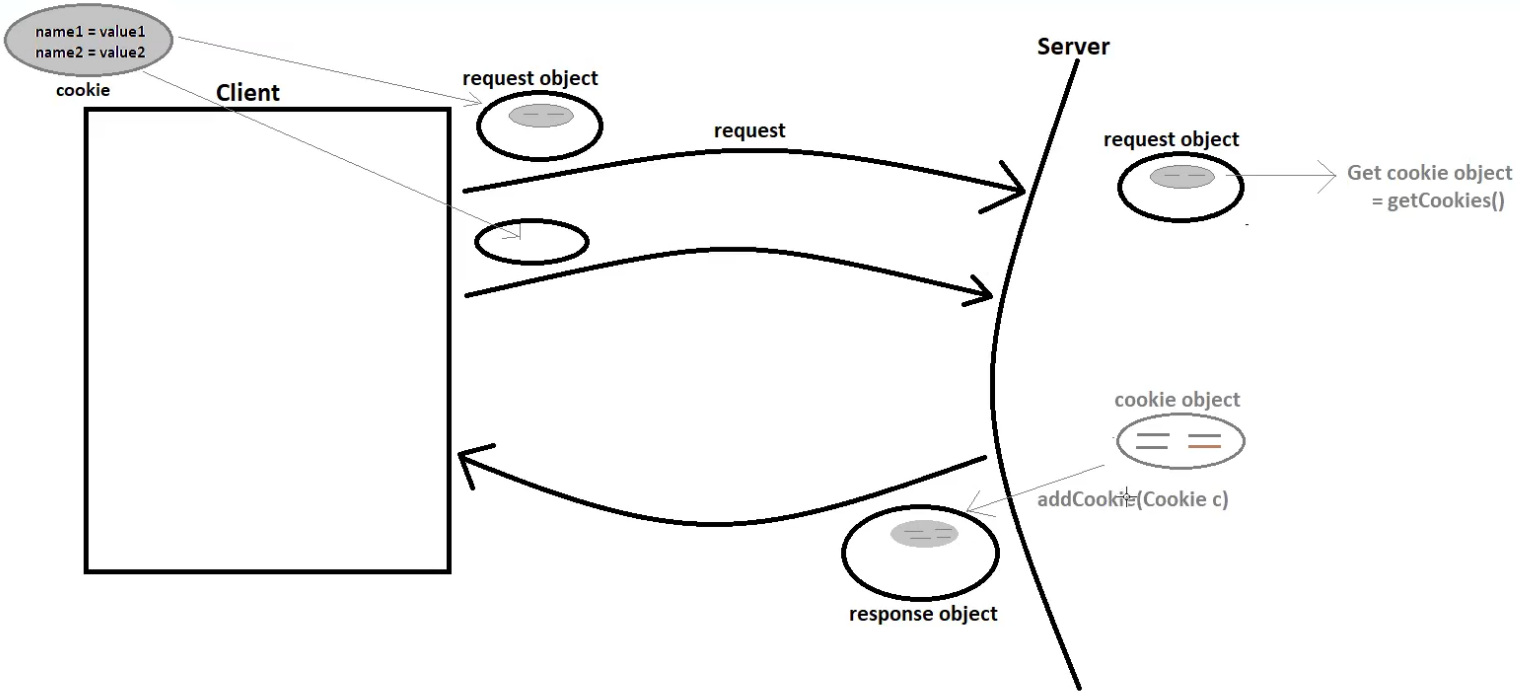
2. Cookie age or expiry date :-
   = public void setMaxAge(int sec)
   = public int getMaxAge()

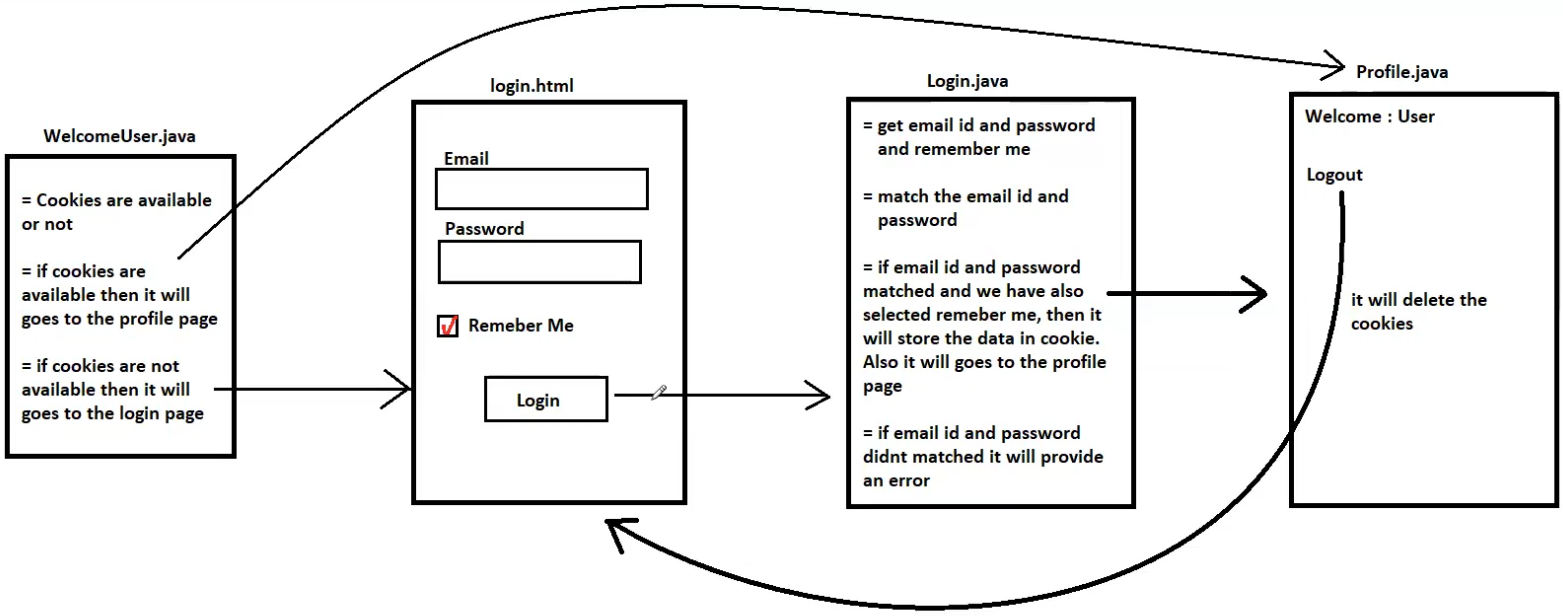
3. Comments related methods for cookies :-
   = public void setComment(String comment)
   = public String getComment()

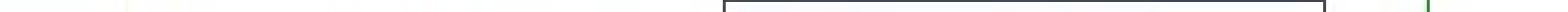
4. Version number related methods for cookies :-
   = public void setVersion(int version_no)
   = public int getVersion()

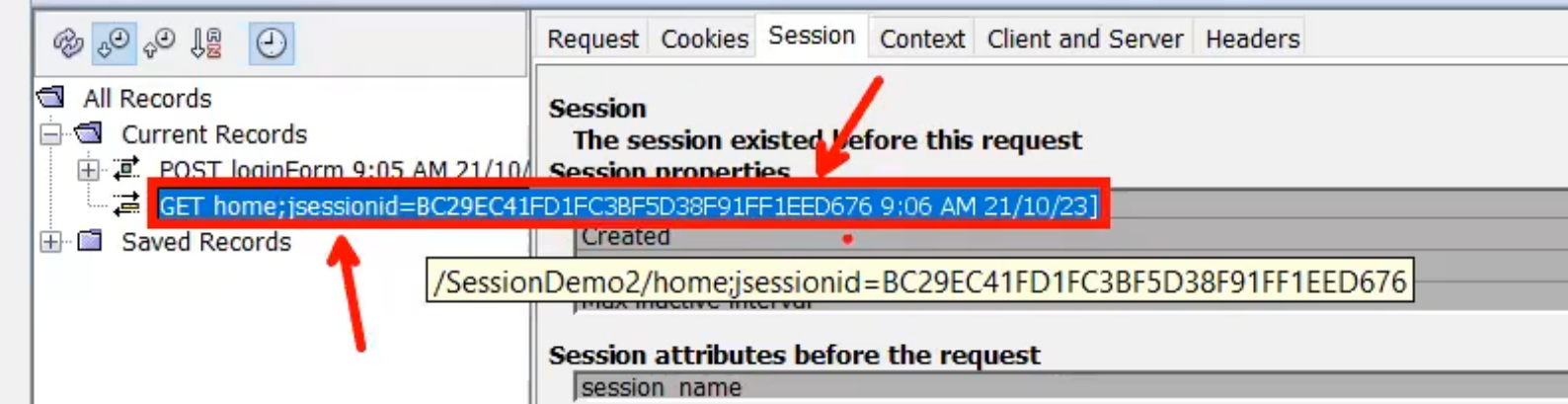
5. Domain name related methods for cookies :-
   = public void setDomain(String domain_name)
   = public String getDomain()

6. To specify path information to cookies :-
   = public void setPath(String path)
   = public String getPath()

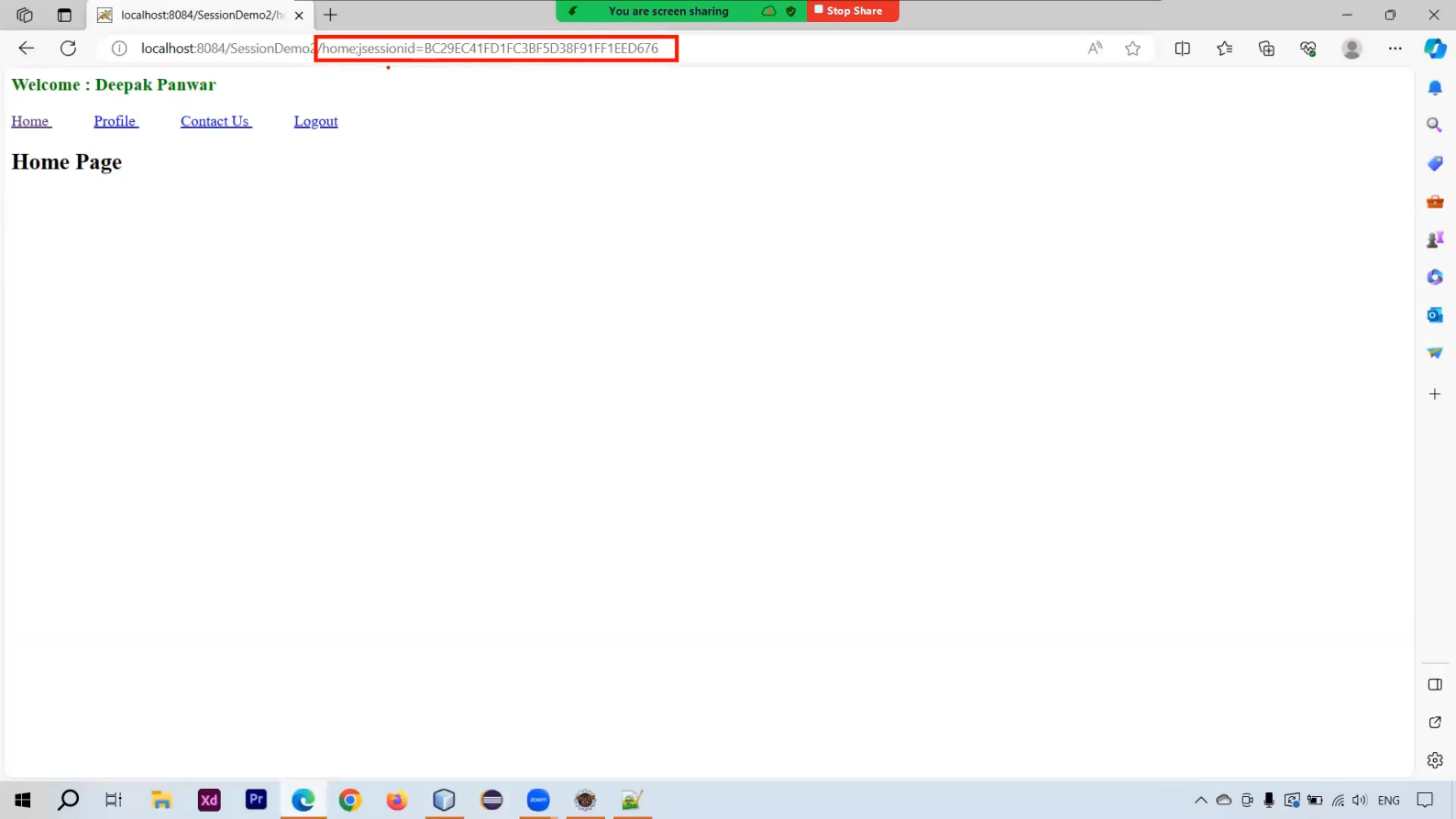

name1 = value1
name2 = value2
cookie
Client
request object
request
Server
request object
Get cookie object
= getCookies()
cookie object
addCookie(Cookie c)
response object


WelcomeUser.java
= Cookies are available or not
= if cookies are available then it will goes to the profile page
= if cookies are not available then it will goes to the login page
login.html
Email
Password
Remeber Me
Login
Login.java
= get email id and password and remember me
= match the email id and password
= if email id and password matched and we have also selected remeber me, then it will store the data in cookie. Also it will goes to the profile page
= if email id and password didnt matched it will provide an error
Profile.java
Welcome : User
Logout
it will delete the cookies


ck.setMaxAge(60*60*24*365);


All Records
Current Records
POST loginForm 9:05 AM 21/10/
GET home;jsessionid=BC29EC41FD1FC3BF5D38F91FF1EED676 9:06 AM 21/10/23]
Saved Records
Request
Cookies
Session
Context
Client and Server
Headers
Session
The session existed before this request
Session properties
Created
/SessionDemo2/home;jsessionid=BC29EC41FD1FC3BF5D38F91FF1EED676
Session attributes before the request
session_name

```
RequestDispatcher rd = req.getRequestDispatcher(resp.encodeURL("/userProfile"));
```


You are screen sharing
Stop Share
localhost:8084/SessionDemo2/home;jsessionid=BC29EC41FD1FC3BF5D38F91FF1EED676
Welcome : Deepak Panwar
Home
Profile
Contact Us
Logout
Home Page

=> URL-Rewritting :-
    -> URL-Rewritting is the technique which is used to modify or enhance the URL's by adding query parameters or any other information to them
    -> This technique allows web developers to pass the data between web pages and maintain state across multiple HTTP requests

    -> Uses of URL-Rewritting :-
        1. Session Tracking Mechanism
        2. Passing Query Parameters

=> Session Tracking Mechanism :-
    -> For session tracking mechanism web containers makes the use of cookies (JSESSIONID). If the client disables the cookies at browser then HttpSession and cookies will not work.
    -> In that case we have to use URL-Rewritting session tracking mechanism

    -> In URL-Rewritting we modify the provided URL by attaching JSESSIONID at the end of URL.
    -> For this HttpServletResponse interface has provided methods :-
        1. public String encodeURL(String url)
            = This method will attach JSESSIONID at the end of specified URL
            = If the cookies are enabled, then it will return the URL unchanged
        2. public String encodeRedirectURL(String url)
            = This method will work same as above method i.e. it will also attach the JSESSIONID at the end of provided URL.
            = This method works with sendRedirect() method.

    -> NOTE :-

-> NOTE :-

1. By default wen container will try to use cookie session tracking mechanism but if the cookies are disabled at the client side (browser) then automatically URL-Rewritting session tracking mechanism will come into work
2. Mostly, we can use URL-Rewritting in case of dynamic web pages (servlets, JSP etc) and not for static web pages (html)
3. URL-Rewritting is automatically handled by web-container, we only need to encode the URL
4. JSESSIONID is automatically added in the URL but we can add the JSESSIONID manually by passing query parameters.

```java
@Override
protected void service(HttpServletRequest req, HttpServletResponse resp) throws ServletExc
{
    resp.setContentType("text/html");

    String name = "deepak";

    PrintWriter out = resp.getWriter();

    out.println("<h2 style='color:red'> This is Servlet 1 Page </h2>");
    out.println("<a href='servlet2?myname="+name+"'> Click Here </a> to go on 'Servlet 2'
}
```

```java
@Override
protected void service(HttpServletRequest req, HttpServletResponse resp) throws ServletExc
{
    resp.setContentType("text/html");

    String name1 = req.getParameter("myname");

    PrintWriter out = resp.getWriter();

    out.println("<h3> Welcome "+name1+"</h3>");
    out.println("<h2 style='color:green'> This is Servlet 2 Page </h2>");
    out.println("<a href='servlet1'> Click Here </a> to go on 'Servlet 1' Page");
}
```

```
--------------------------------------------------------------------------

=> Passing Query Parameters :-
    -> URL-Rewritting is also used to pass query parameters and data within the URL's
    -> This is used to transfer the information or data from one page to another
    -> Syntax :-
        URL ? parameter1=value1 & parameter2=value2 & ____

    -> NOTE :
        = Do not send the sensitive data using query parameters

==========================================================================================
```

<input type="hidden" name="pageid1" value="111" />

```
<h2> Home Page </h2>

<form action="myservlet" method="POST">
    <input type="hidden" name="page1" value="home" />
    <input type="submit" value="Click Me" />
</form>
```

```
<h2> Profile Page </h2>

<form action="myservlet" method="POST">
    <input type="hidden" name="page1" value="profile" />
    <input type="submit" value="Click Me" />
</form>
```

```
    PrintWriter out = resp.getWriter();

    String pageName = req.getParameter("page1");

    if(pageName.equals("home"))
    {
        out.println("Request came from HOME page");
        //home page code
    }
    else if(pageName.equals("profile"))
    {
        out.println("Request came from PROFILE page");
        //profile page code
    }
}
```

=> Hidden Form Fields :-

-> Hidden form field is a specific type of HTML form field that is used to store the data on client side (browser) while keeping the data hidden from user

-> It is used to pass the data from one page to another or from browser to server without displaying the data to the user

-> Syntax :-

```
<input type="hidden" name="----" value="----" />
```

-> Uses of hidden form field :-

1. Session tracking mechanism
2. Passing data between pages
3. Secutiry tokens
4. Storing user preferences
5. URL redirection

   etc

-> NOTE :-

= Hidden form fields are not suitable for storing sensitive and confidential information as data is visible in the page source code and can be manipulated by the user

==========

=> ServletContext :-

-> ServletContext is an interface provided by Java Servlet API

-> Syntax :-

```
public interface ServletContext
{
    //methods
}
```

-> It is used to store and share configuration and resource data among all servlets within the same web application

-> Points to note :-

= When the ServletContext object is created :-

>> When we start the server, then web container also starts and it deploys all the projects and at that time ServletContext object is created

= When the ServletContext object is destroyed :-

>> When we shutdown the server, then web container will also stops and all the projects gets undeployed and at that time ServletContext object will be destroyed

= Lifetime of ServletContext object :-

>> The lifetime of ServletContext object is same as that lifetime of web application

= Scope of ServletContext object :-

>> The scope of ServletContext object is accessaible to the whole web application and to all the resources of that particular web application

= Which type of data we can store in ServletContext object :-

>> It can store parameters data and attributes data

```
-> How to get ServletContext object :-
    1. By using getServletContext() method of Servlet interface
        >> ServletContext context = getServletContext();
    2. By using ServletRequest object
        >> ServletContext context = req.getServletContext();
    3. By using ServletConfig object
        >> ServletContext context = config.getServletContext();

-> How to store data in ServletContext object :-
    1. Using setAttribute() method
        >> context.setAttribute("attributeName", attributeValue);
    2. Using Initialization Parameters (web.xml)
        >> <context-param>
                <param-name> parameterName </param-name>
                <param-value> parameterValue </param-value>
           </context-param>

-> How to get data from ServletContext object :-
    1. Using getAttribute() method
        >> Object value = context.getAttribute("attributeName");
    2. Using Initialization Parameters methods
        >> String parameterValue = context.getInitParameter("parameterName");
```

=> ServletContext :-

-> ServletContext is an interface provided by Java Servlet API

-> Syntax :-

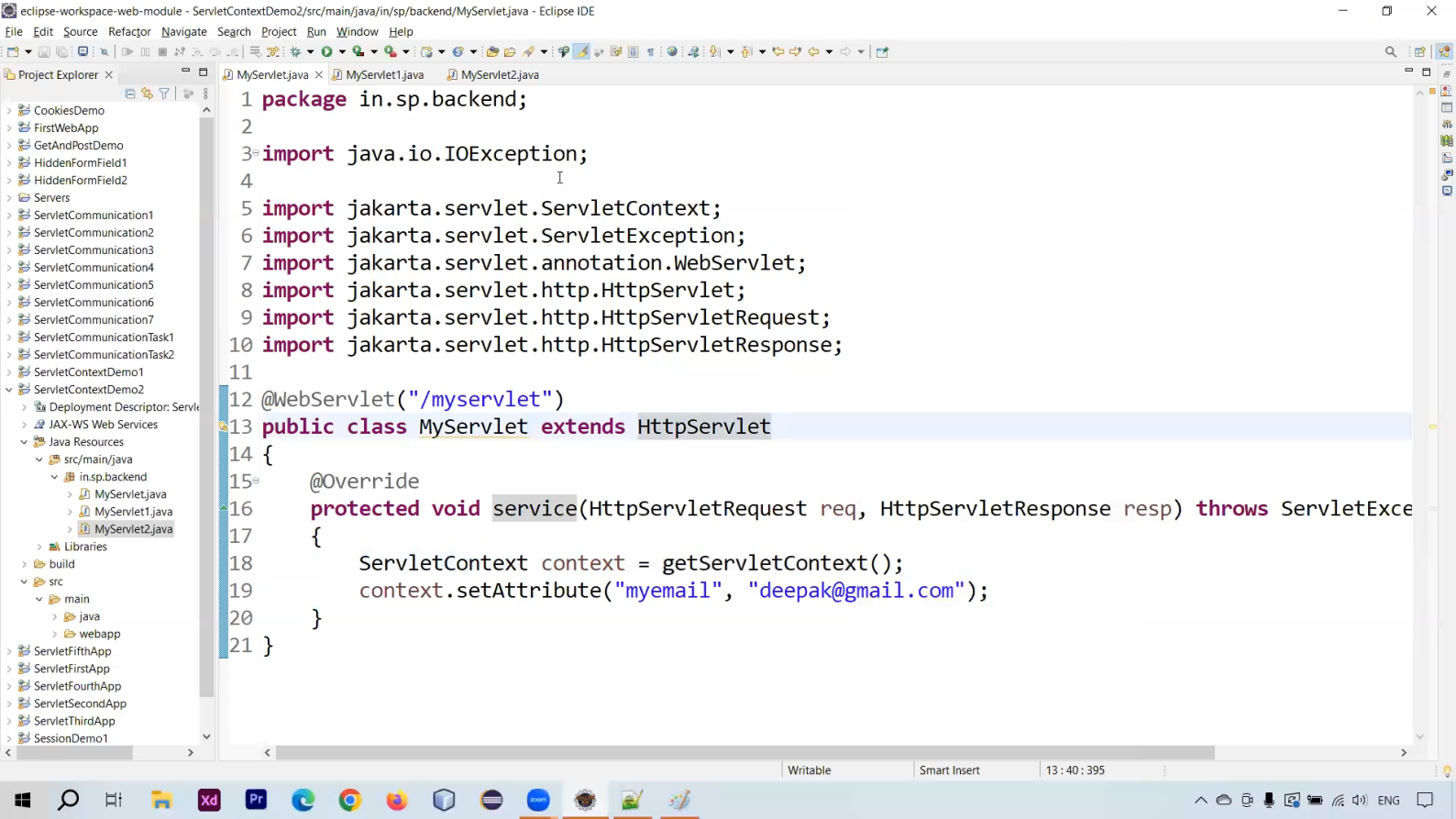
```
public interface ServletContext
{
    //methods
    public void setAttribute(String name, Object value);
    public Object getAttribute(String name);
    public void removeAttribute(String name)

    public String getInitParameter(String name)
    public Enumeration getInitParameterNames()

    etc
}
```

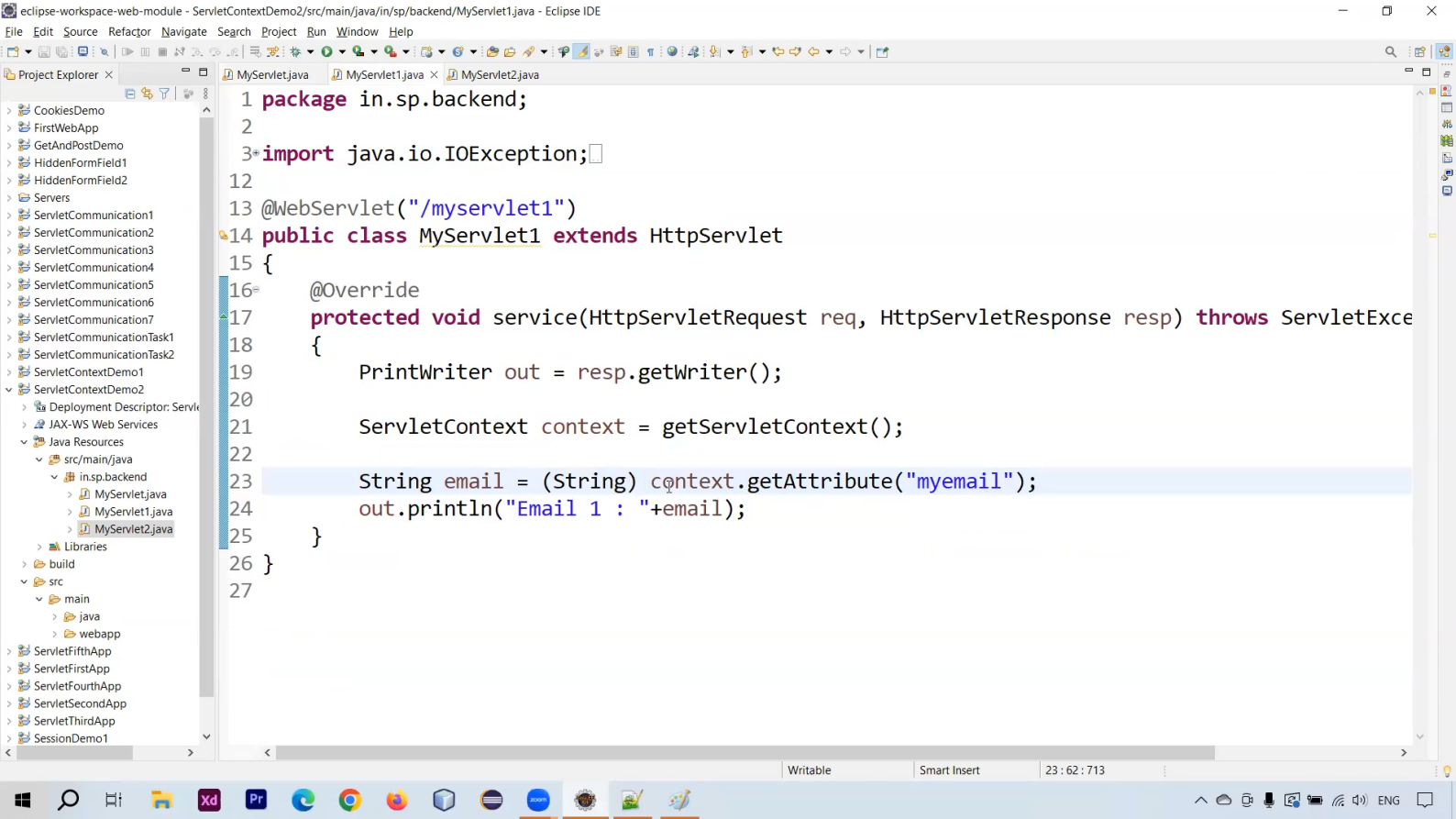
```
{
    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp) throws ServletExce
    {
        PrintWriter out = resp.getWriter();

        ServletContext context = getServletContext();

        String myname = context.getInitParameter("name1");
        out.println("Name : "+myname);
    }
}
```

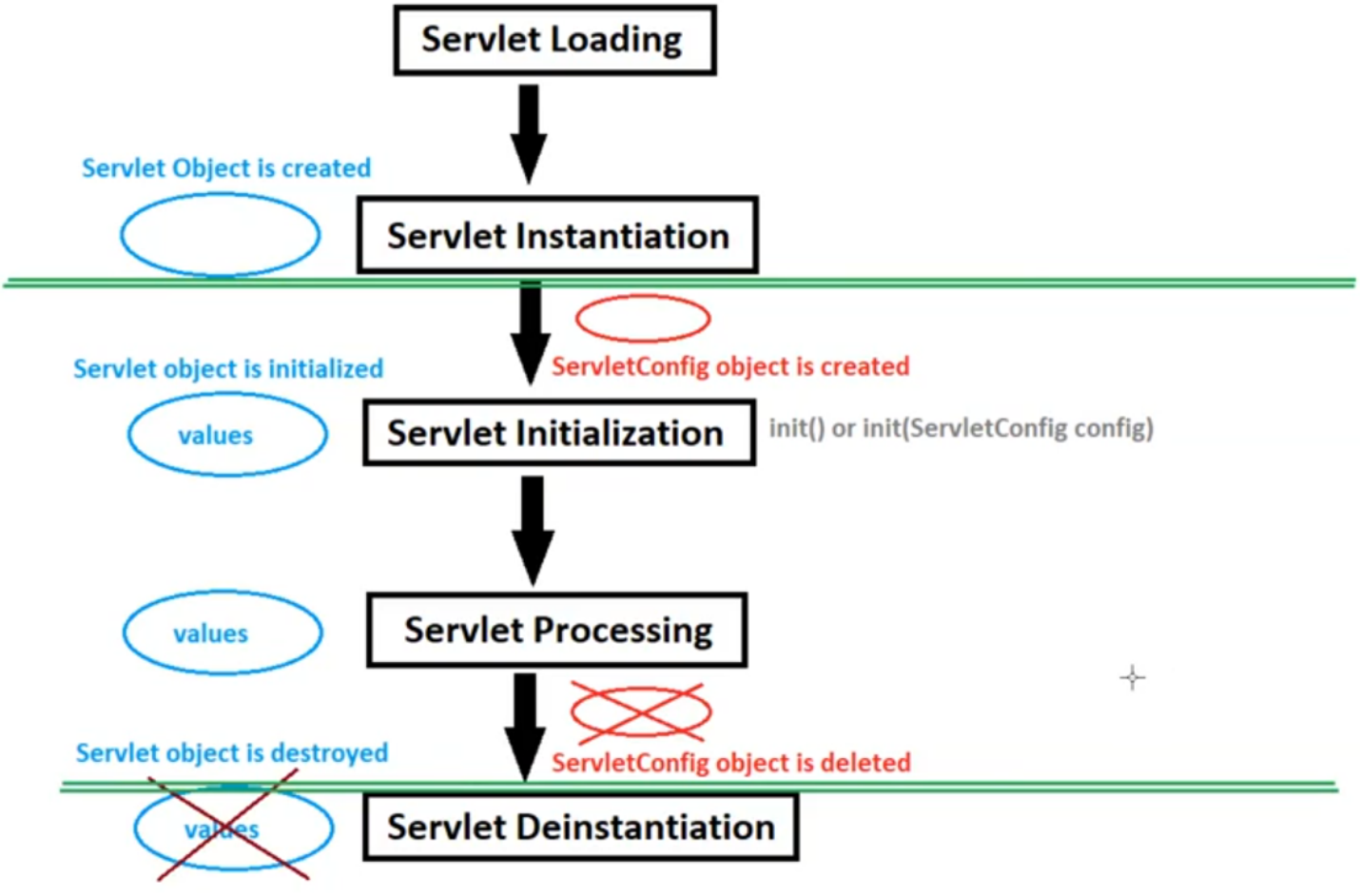
MyServlet1.java | web.xml | MyServlet2.java

Bind to grammar/schema...

```
1 <web-app>
2     <context-param>
3         <param-name> name1 </param-name>
4         <param-value> Deepak Parwar </param-value>
5     </context-param>
6 </web-app>
```


eclipse-workspace-web-module - ServletContextDemo2/src/main/java/in/sp/backend/MyServlet.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
CookiesDemo
FirstWebApp
GetAndPostDemo
HiddenFormField1
HiddenFormField2
Servers
ServletCommunication1
ServletCommunication2
ServletCommunication3
ServletCommunication4
ServletCommunication5
ServletCommunication6
ServletCommunication7
ServletCommunicationTask1
ServletCommunicationTask2
ServletContextDemo1
ServletContextDemo2
Deployment Descriptor: Servl
JAX-WS Web Services
Java Resources
src/main/java
in.sp.backend
MyServlet.java
MyServlet1.java
MyServlet2.java
Libraries
build
src
main
java
webapp
ServletFifthApp
ServletFirstApp
ServletFourthApp
ServletSecondApp
ServletThirdApp
SessionDemo1
MyServlet.java
MyServlet1.java
MyServlet2.java
1 package in.sp.backend;
2
3 import java.io.IOException;
4
5 import jakarta.servlet.ServletContext;
6 import jakarta.servlet.ServletException;
7 import jakarta.servlet.annotation.WebServlet;
8 import jakarta.servlet.http.HttpServlet;
9 import jakarta.servlet.http.HttpServletRequest;
10 import jakarta.servlet.http.HttpServletResponse;
11
12 @WebServlet("/myservlet")
13 public class MyServlet extends HttpServlet
14 {
15 @Override
16 protected void service(HttpServletRequest req, HttpServletResponse resp) throws ServletExce
17 {
18 ServletContext context = getServletContext();
19 context.setAttribute("myemail", "deepak@gmail.com");
20 }
21 }
Writable
Smart Insert
13 : 40 : 395
ENG


eclipse-workspace-web-module - ServletContextDemo2/src/main/java/in/sp/backend/MyServlet1.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
CookiesDemo
FirstWebApp
GetAndPostDemo
HiddenFormField1
HiddenFormField2
Servers
ServletCommunication1
ServletCommunication2
ServletCommunication3
ServletCommunication4
ServletCommunication5
ServletCommunication6
ServletCommunication7
ServletCommunicationTask1
ServletCommunicationTask2
ServletContextDemo1
ServletContextDemo2
Deployment Descriptor: Servl
JAX-WS Web Services
Java Resources
src/main/java
in.sp.backend
MyServlet.java
MyServlet1.java
MyServlet2.java
Libraries
build
src
main
java
webapp
ServletFifthApp
ServletFirstApp
ServletFourthApp
ServletSecondApp
ServletThirdApp
SessionDemo1
MyServlet.java
MyServlet1.java
MyServlet2.java
1 package in.sp.backend;
2
3 import java.io.IOException;
12
13 @WebServlet("/myservlet1")
14 public class MyServlet1 extends HttpServlet
15 {
16 @Override
17 protected void service(HttpServletRequest req, HttpServletResponse resp) throws ServletExce
18 {
19 PrintWriter out = resp.getWriter();
20
21 ServletContext context = getServletContext();
22
23 String email = (String) context.getAttribute("myemail");
24 out.println("Email 1 : "+email);
25 }
26 }
27
Writable
Smart Insert
23 : 62 : 713
ENG


Servlet Loading
Servlet Object is created
Servlet Instantiation
Servlet object is initialized
ServletConfig object is created
values
Servlet Initialization
init() or init(ServletConfig config)
values
Servlet Processing
Servlet object is destroyed
ServletConfig object is deleted
values
Servlet Deinstantiation

=> ServletConfig :-

-> ServletConfig is an interface provided by the Java Servlet API

-> Syntax :-

```
public interface ServletConfig
{
    //methods
    public String getInitParameter(String name);
    public Enumeration getInitParameterNames();

    public ServletContext getServletContext();
    public String getServletName();
}
```

-> ServletConfig is used to provide the configuration information to the servlet.

-> The variable which we are going to change after some time, that we can provide in ServletConfig object so tgar we dont need to recompile our servlet page

-> Points to note :-

= When the ServletConfig object is created :-

>> ServletConfig object is created after Servlet Instantiation and just before the Servlet Initialization phase

= When the ServletConfig object is destroyed :-

>> ServletConfig object is destroyed just before the Servlet Deinstantiation phase

= Lifetime of ServletConfig object :-

>> The lifetime of ServletConfig object is approx same as Servlet Object

= Scope of ServletConfig object :-

>> The scope of ServletConfig is only to one particular servlet

= Which type of data we can store in ServletConfig object :-

>> It can store only parameters data but not the attributes data

-> How to get ServletConfig object :-

1. By using getServletConfig() method of Servlet interface
   >> ServletConfig config = getServletConfig();
2. By overriding the init(ServletConfig config) method
   >> public class Test extends HttpServlet

```
{
    ServletConfig config;

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

-> How to store data in ServletConfig object :-

1. Using web.xml file

```
<web-app>

    <servlet>
        <servlet-name> ---- </servlet-name>
        <servlet-class> ---- </servlet-class>
        <init-param>
            <param-name> ---- </param-name>
            <param-value> ---- </param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name> ---- </servlet-name>
        <url-pattern> ---- </url-pattern>
    </servlet-mapping>

    <servlet>
        <servlet-name> ---- </servlet-name>
        <servlet-class> ---- </servlet-class>
        <init-param>
            <param-name> ---- </param-name>
            <param-value> ---- </param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name> ---- </servlet-name>
        <url-pattern> ---- </url-pattern>
    </servlet-mapping>

    <context-param>
        <param-name> parameterName </param-name>
        <param-value> parameterValue </param-value>
    </context-param>

</web-app>
```

-> How to store data in ServletConfig object :-

1. Using web.xml file

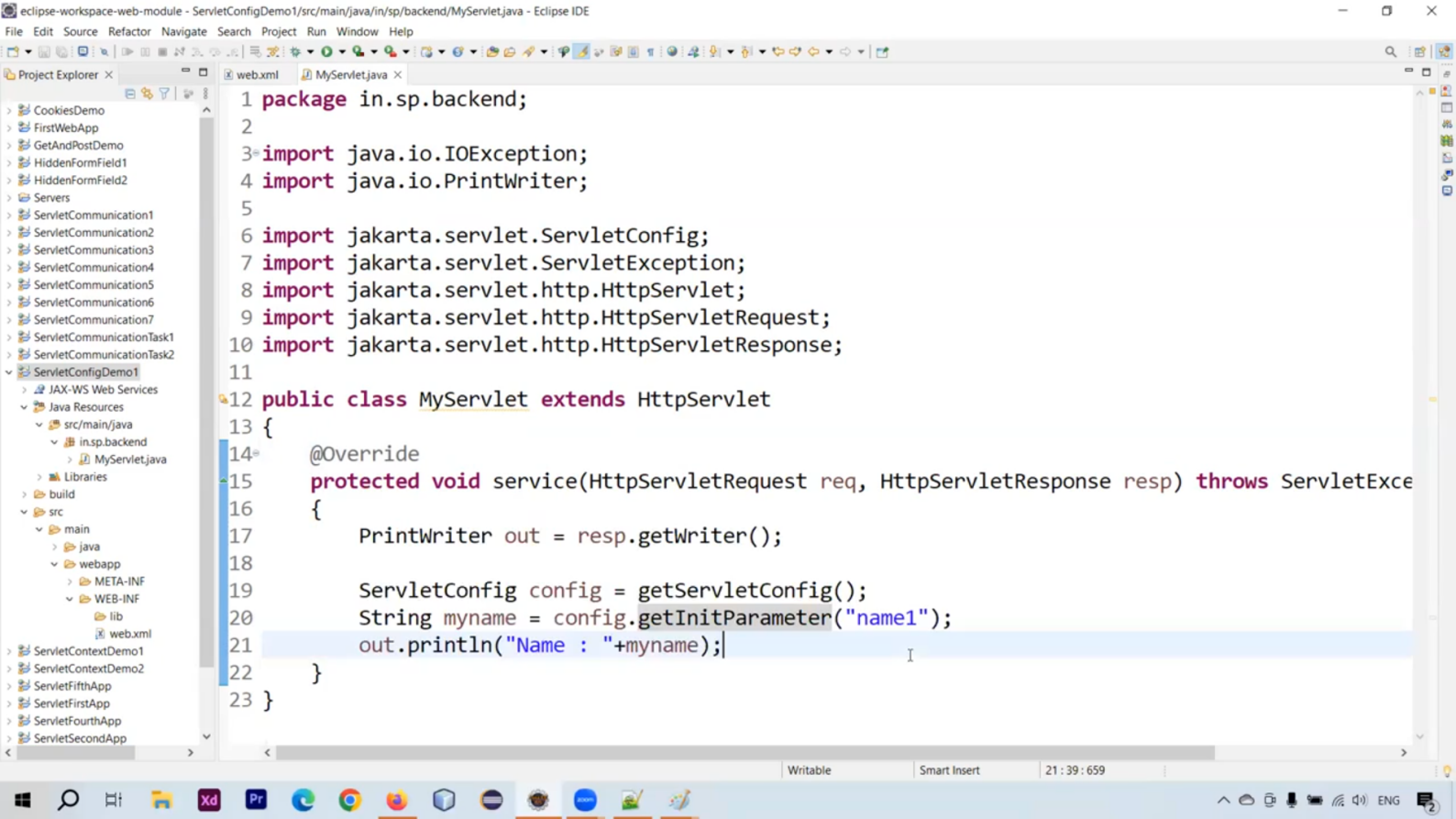
```
<web-app>
    <servlet>
        <servlet-name> ---- </servlet-name>
        <servlet-class> ---- </servlet-class>
        <init-param>
            <param-name> ---- </param-name>
            <param-value> ---- </param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name> ---- </servlet-name>
        <url-pattern> ---- </url-pattern>
    </servlet-mapping>
</web-app>
```

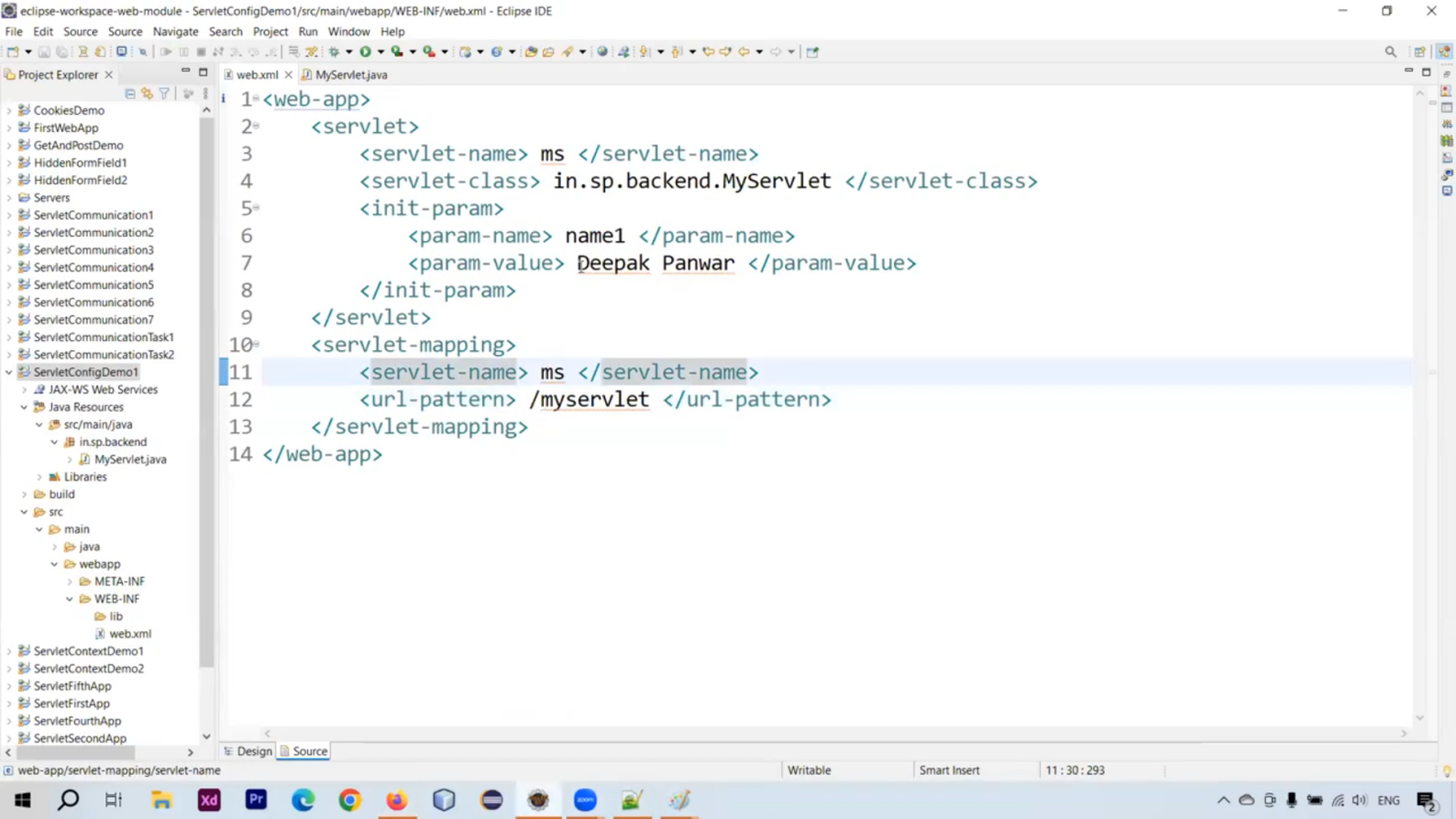
-> How to get data from ServletConfig object :-

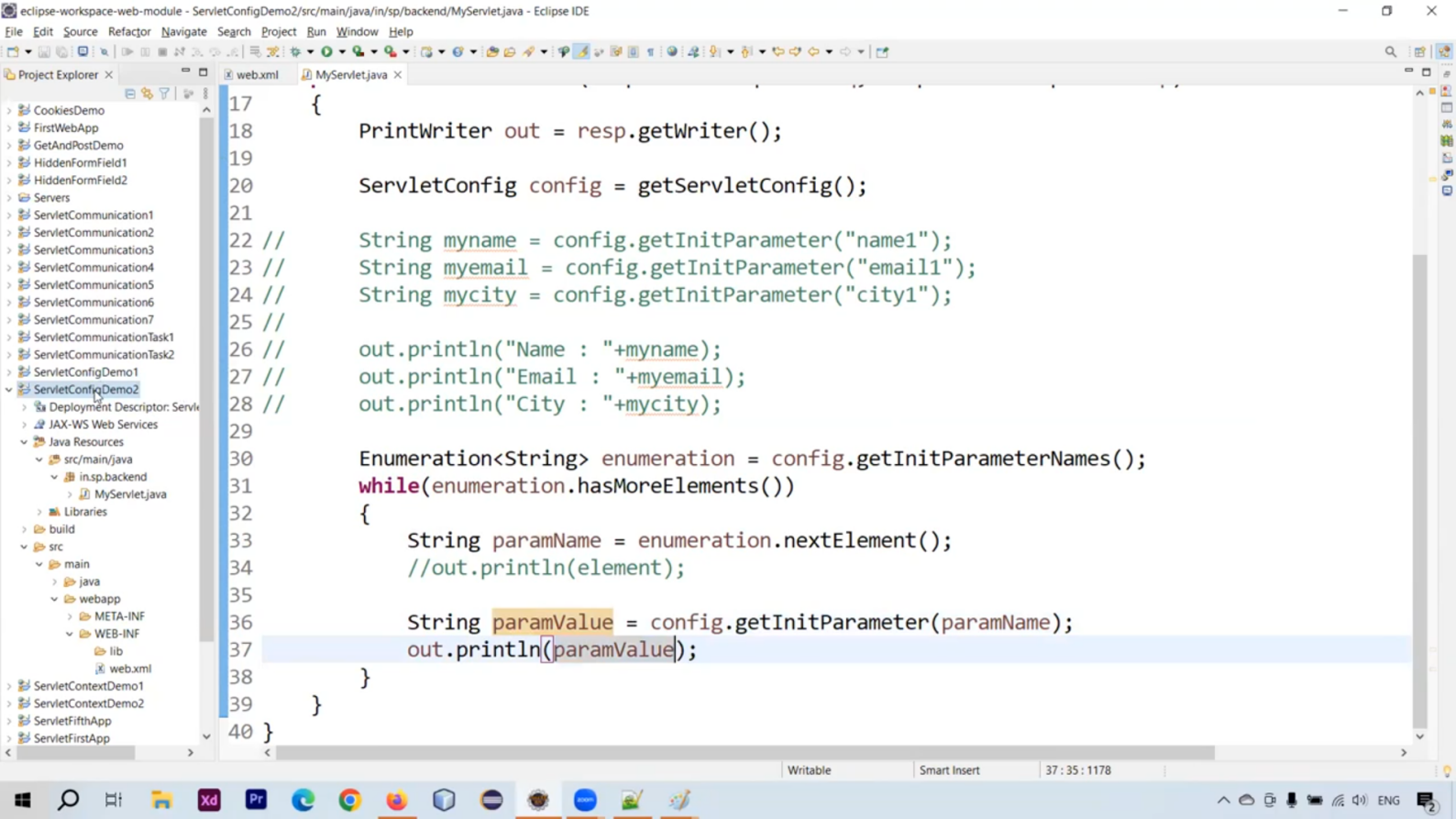
= We can use below methods :-

>> public String getInitParameter(String name);

>> public Enumeration getInitParameterNames();


eclipse-workspace-web-module - ServletConfigDemo1/src/main/java/in/sp/backend/MyServlet.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
CookiesDemo
FirstWebApp
GetAndPostDemo
HiddenFormField1
HiddenFormField2
Servers
ServletCommunication1
ServletCommunication2
ServletCommunication3
ServletCommunication4
ServletCommunication5
ServletCommunication6
ServletCommunication7
ServletCommunicationTask1
ServletCommunicationTask2
ServletConfigDemo1
JAX-WS Web Services
Java Resources
src/main/java
in.sp.backend
MyServlet.java
Libraries
build
src
main
java
webapp
META-INF
WEB-INF
lib
web.xml
ServletContextDemo1
ServletContextDemo2
ServletFifthApp
ServletFirstApp
ServletFourthApp
ServletSecondApp
web.xml
MyServlet.java
1 package in.sp.backend;
2
3 import java.io.IOException;
4 import java.io.PrintWriter;
5
6 import jakarta.servlet.ServletConfig;
7 import jakarta.servlet.ServletException;
8 import jakarta.servlet.http.HttpServlet;
9 import jakarta.servlet.http.HttpServletRequest;
10 import jakarta.servlet.http.HttpServletResponse;
11
12 public class MyServlet extends HttpServlet
13 {
14 @Override
15 protected void service(HttpServletRequest req, HttpServletResponse resp) throws ServletExce
16 {
17 PrintWriter out = resp.getWriter();
18
19 ServletConfig config = getServletConfig();
20 String myname = config.getInitParameter("name1");
21 out.println("Name : "+myname);
22 }
23 }
Writable
Smart Insert
21 : 39 : 659
ENG


eclipse-workspace-web-module - ServletConfigDemo1/src/main/webapp/WEB-INF/web.xml - Eclipse IDE
File Edit Source Source Navigate Search Project Run Window Help
Project Explorer
CookiesDemo
FirstWebApp
GetAndPostDemo
HiddenFormField1
HiddenFormField2
Servers
ServletCommunication1
ServletCommunication2
ServletCommunication3
ServletCommunication4
ServletCommunication5
ServletCommunication6
ServletCommunication7
ServletCommunicationTask1
ServletCommunicationTask2
ServletConfigDemo1
JAX-WS Web Services
Java Resources
src/main/java
in.sp.backend
MyServlet.java
Libraries
build
src
main
java
webapp
META-INF
WEB-INF
lib
web.xml
ServletContextDemo1
ServletContextDemo2
ServletFifthApp
ServletFirstApp
ServletFourthApp
ServletSecondApp
web.xml
MyServlet.java
1 <web-app>
2 <servlet>
3 <servlet-name> ms </servlet-name>
4 <servlet-class> in.sp.backend.MyServlet </servlet-class>
5 <init-param>
6 <param-name> name1 </param-name>
7 <param-value> Deepak Panwar </param-value>
8 </init-param>
9 </servlet>
10 <servlet-mapping>
11 <servlet-name> ms </servlet-name>
12 <url-pattern> /myservlet </url-pattern>
13 </servlet-mapping>
14 </web-app>
Design
Source
web-app/servlet-mapping/servlet-name
Writable
Smart Insert
11 : 30 : 293
ENG


eclipse-workspace-web-module - ServletConfigDemo2/src/main/java/in/sp/backend/MyServlet.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
web.xml
MyServlet.java
CookiesDemo
FirstWebApp
GetAndPostDemo
HiddenFormField1
HiddenFormField2
Servers
ServletCommunication1
ServletCommunication2
ServletCommunication3
ServletCommunication4
ServletCommunication5
ServletCommunication6
ServletCommunication7
ServletCommunicationTask1
ServletCommunicationTask2
ServletConfigDemo1
ServletConfigDemo2
Deployment Descriptor: Servl
JAX-WS Web Services
Java Resources
src/main/java
in.sp.backend
MyServlet.java
Libraries
build
src
main
java
webapp
META-INF
WEB-INF
lib
web.xml
ServletContextDemo1
ServletContextDemo2
ServletFifthApp
ServletFirstApp
17 {
18 PrintWriter out = resp.getWriter();
19
20 ServletConfig config = getServletConfig();
21
22 // String myname = config.getInitParameter("name1");
23 // String myemail = config.getInitParameter("email1");
24 // String mycity = config.getInitParameter("city1");
25 //
26 // out.println("Name : "+myname);
27 // out.println("Email : "+myemail);
28 // out.println("City : "+mycity);
29
30 Enumeration<String> enumeration = config.getInitParameterNames();
31 while(enumeration.hasMoreElements())
32 {
33 String paramName = enumeration.nextElement();
34 //out.println(element);
35
36 String paramValue = config.getInitParameter(paramName);
37 out.println(paramValue);
38 }
39 }
40 }
Writable
Smart Insert
37 : 35 : 1178
ENG

Event Overview:

- **Hackfest-Advaita** is an exciting opportunity for students from various colleges to collaborate, share knowledge, and work on innovative projects in the field of technology.
- This hackathon will cover a wide range of domains, including:
  - **HealthCare**
  - **Sustainability**
  - **Education**
  - **Fintech**
  - **Entertainment**
  - **Internet of Things (IoT)**
  - **Blockchain**
  - **Open Innovation**

AI-Powered Financial Insights Platform

PROBLEM STATEMENT

India possesses vast amounts of financial and economic data across sectors, but professionals, researchers, policymakers, and citizens struggle to access, analyze, and interpret this information effectively. The current landscape lacks comprehensive tools that transform complex economic data into accessible insights, hindering informed decision-making and financial literacy.

CHALLENGE

Develop an innovative platform that democratizes financial information in India by leveraging AI to transform raw economic data into meaningful insights and visualizations. Your solution should bridge the gap between complex financial information and actionable intelligence.

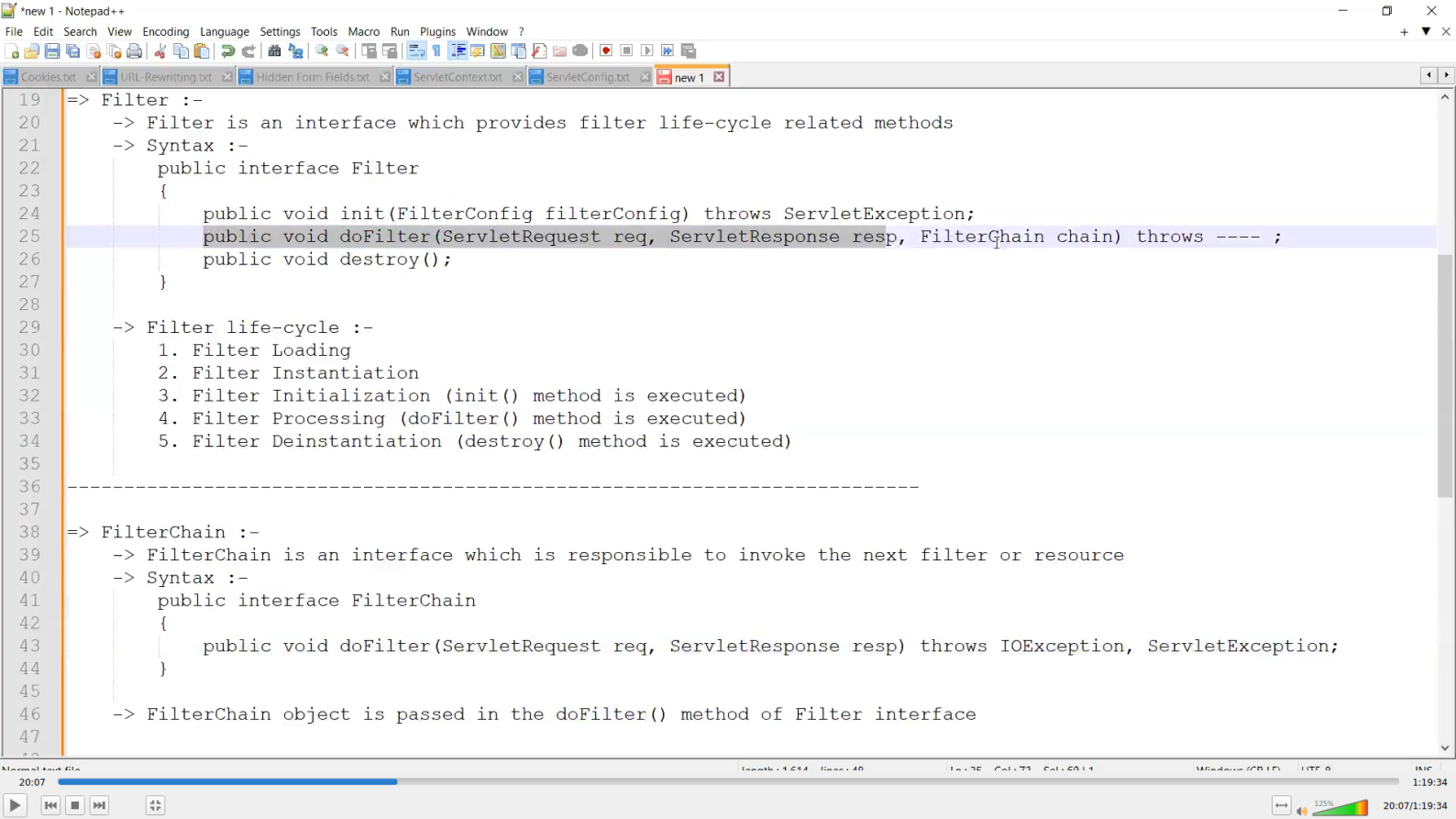
OBJECTIVES

Create a solution that allows users to:

- Discover and explore India's financial ecosystem through consolidated data
- Uncover hidden patterns and trends in economic performance
- Generate research-quality analysis and reports
- Visualize complex financial relationships in intuitive ways

=> Filters :-
    -> A filter is a server side component which is executed at pre-processing and post-processing of the request
    -> Use of filters :-
        1. Input Validations
        2. Encryption & Decryption
        3. Session Validations
        4. Logging Operations
        5. Internationalization
           etc

    -> Filter API :-
        = There are 3 main interfaces in Filter API :-
            1. Filter
            2. FilterChain
            3. FilterConfig


*new 1 - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Cookies.txt URL-Rewritting.txt Hidden Form Fields.txt ServletContext.txt ServletConfig.txt new 1
=> Filter :-
-> Filter is an interface which provides filter life-cycle related methods
-> Syntax :-
public interface Filter
{
public void init(FilterConfig filterConfig) throws ServletException;
public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) throws ---- ;
public void destroy();
}
-> Filter life-cycle :-
1. Filter Loading
2. Filter Instantiation
3. Filter Initialization (init() method is executed)
4. Filter Processing (doFilter() method is executed)
5. Filter Deinstantiation (destroy() method is executed)

=> FilterChain :-
-> FilterChain is an interface which is responsible to invoke the next filter or resource
-> Syntax :-
public interface FilterChain
{
public void doFilter(ServletRequest req, ServletResponse resp) throws IOException, ServletException;
}
-> FilterChain object is passed in the doFilter() method of Filter interface
20:07
1:19:34
20:07/1:19:34

```
=> FilterConfig :-
    -> FilterConfig is an interface which is used to provide any configuration data to the filters
    -> Syntax :-
        public interface FilterConfig
        {
            public String getFilterName();
            public ServletContext getServletContext();
            public String getInitParameter(String name);
            public Enumeration<String> getInitParameterNames();
        }

    -> For every Filter an object is created by web container which is known as FilterConfig object
    -> The FilterConfig object gets the configuration information from web.xml file or annotations
    -> The configuration information is provided only for particular filter
```

```xml
<web-app>
    <servlet>
        <servlet-name>reg</servlet-name>
        <servlet-class>in.sp.servlets.Register</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>reg</servlet-name>
        <url-pattern>/regForm</url-pattern>
    </servlet-mapping>
</web-app>
```

```xml
<web-app>
    <filter>
        <filter-name>validationFilter</filter-name>
        <filter-class>in.sp.filters.ValidationFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>validationFilter</filter-name>
        <url-pattern>/regForm</url-pattern>
    </filter-mapping>

    <servlet>
        <servlet-name>reg</servlet-name>
        <servlet-class>in.sp.servlets.Register</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>reg</servlet-name>
        <url-pattern>/regForm</url-pattern>
    </servlet-mapping>
</web-app>
```

```xml
<web-app>
    <filter>
        <filter-name>validationFilter</filter-name>
        <filter-class>in.sp.filters.ValidationFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>validationFilter</filter-name>
        <url-pattern>/regForm</url-pattern>
    </filter-mapping>

    <servlet>
        <servlet-name>reg</servlet-name>
        <servlet-class>in.sp.servlets.Register</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>reg</servlet-name>
        <url-pattern>/regForm</url-pattern>
    </servlet-mapping>
</web-app>
```

```
@WebFilter(urlPatterns = {"/regForm"}, filterName = "validationFilter")
```

=> Static Web Pages :-
    -> Static web pages are pre-built and do not change the content unless manually editted by web developer
    -> Static web pages are created in HTML, CSS etc
    -> Static pages are fast to load because there is no server-side processing involved
    -> Examples :-
        - home pages
        - login and register pages
        - portfolio pages
        - website information pages
          etc

=> Dynamic Web Pages :-
    -> Dynamic web pages change their content based on the user-input or database updates or external factors
    -> Dynamic web pages are created in Servlets, JSP, Spring, PHP, Python etc
    -> Dynamic web pages are more versatile but may take longer time to load due to server-side processing
    -> Examples :-
        - Profile pages
        - Social networking websites home pages
        - Shopping web pages
          etc

=> JSP (Java Server Pages) :-
    -> JSP is server side technology which is used to create dynamic web pages
    -> Using JSP technology, we can embed Java code into HTML code

    -> JSP is an advanced version of servlet technology
        = In servlets we embed HTML code within Java code
        = In JSP we embed Java code within HTML code

    -> What is difference between Servlet & JSP :-
        1. Servlet is a java code
           JSP is the HTML-based code

        2. It is hard to write code in servlets
           It's easier to write code in JSP

        3. Servlets are fast as compared to JSP
           JSP are slower as compared to servlets because it takes time to convert into servlet and compile the servlet

        4. In MVC architecture, servlets work as a controller
           In MVC architecture, JSP works as a view to display the output

        5. Modification in servlet file is time consuming due to reloading, recompilation and restarting the server
           Modification in JSP is fast as just we need to click refresh button

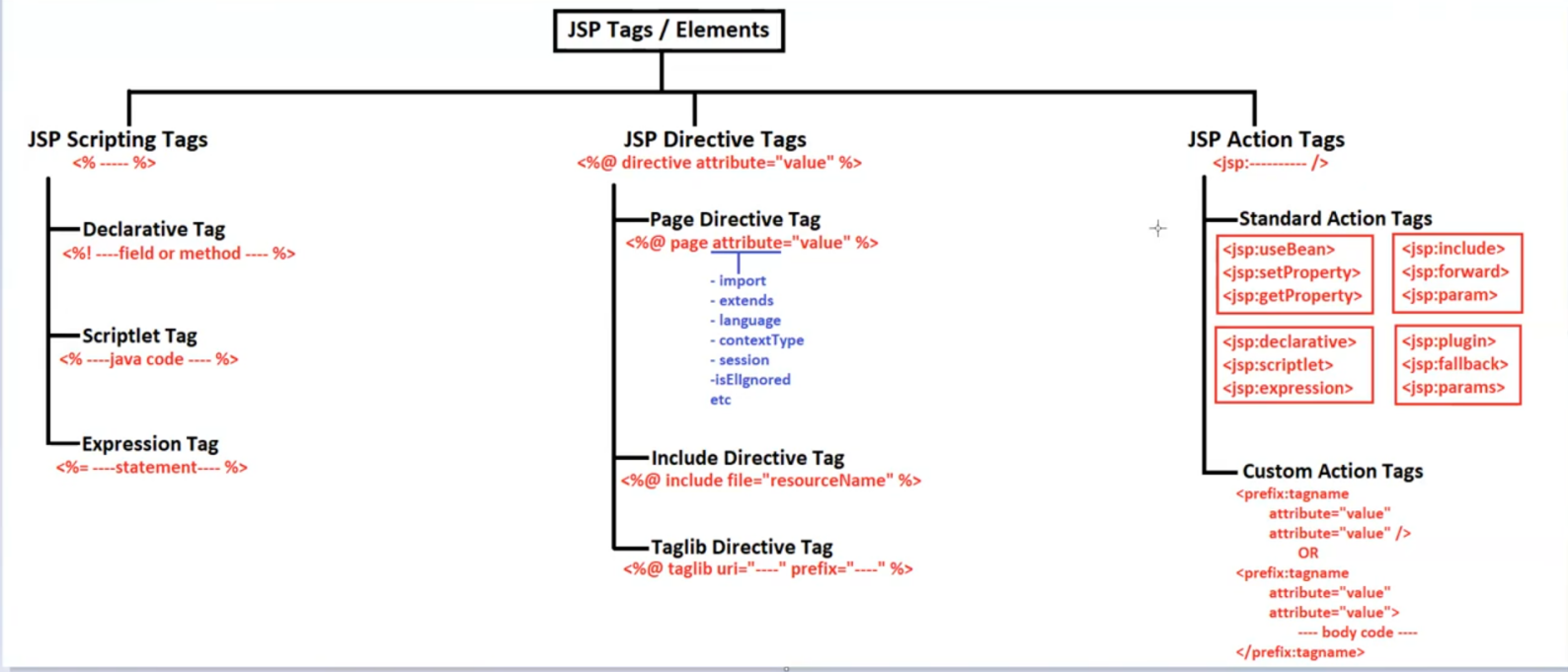
-> Features of JSP :-
1. Simplified Development
= JSP simplifies web development by combining Java and HTML for dynamic web pages
2. Built-in Tags :-
= JSP includes built-in tags like scriptlet tag, action tags, custom tags to streamline coding
3. Easy Maintenance :-
= JSP promotes maintability, flexibility and portability
4. No Recompilation :-
= Changes to JSP page doesnt require recompilation and redeployment
5. Extention to servlet :-
= JSP is an extention of servlet technology for easier web app development

-> How to create JSP pages :-
1. Create a text file with .jsp extension
2. Write HTML code for page structure
3. Embed Java code to generate dynamic content by using tags i.e. <%----=%> or <jsp:----> etc
-> NOTE :-
1. JSP pages should be created in webapp folder.
2. We can also create jsp pages in WEB-INF folder but when we want to access jsp page then we have to provide the mapping in web.xml file because WEB-INF folder is private

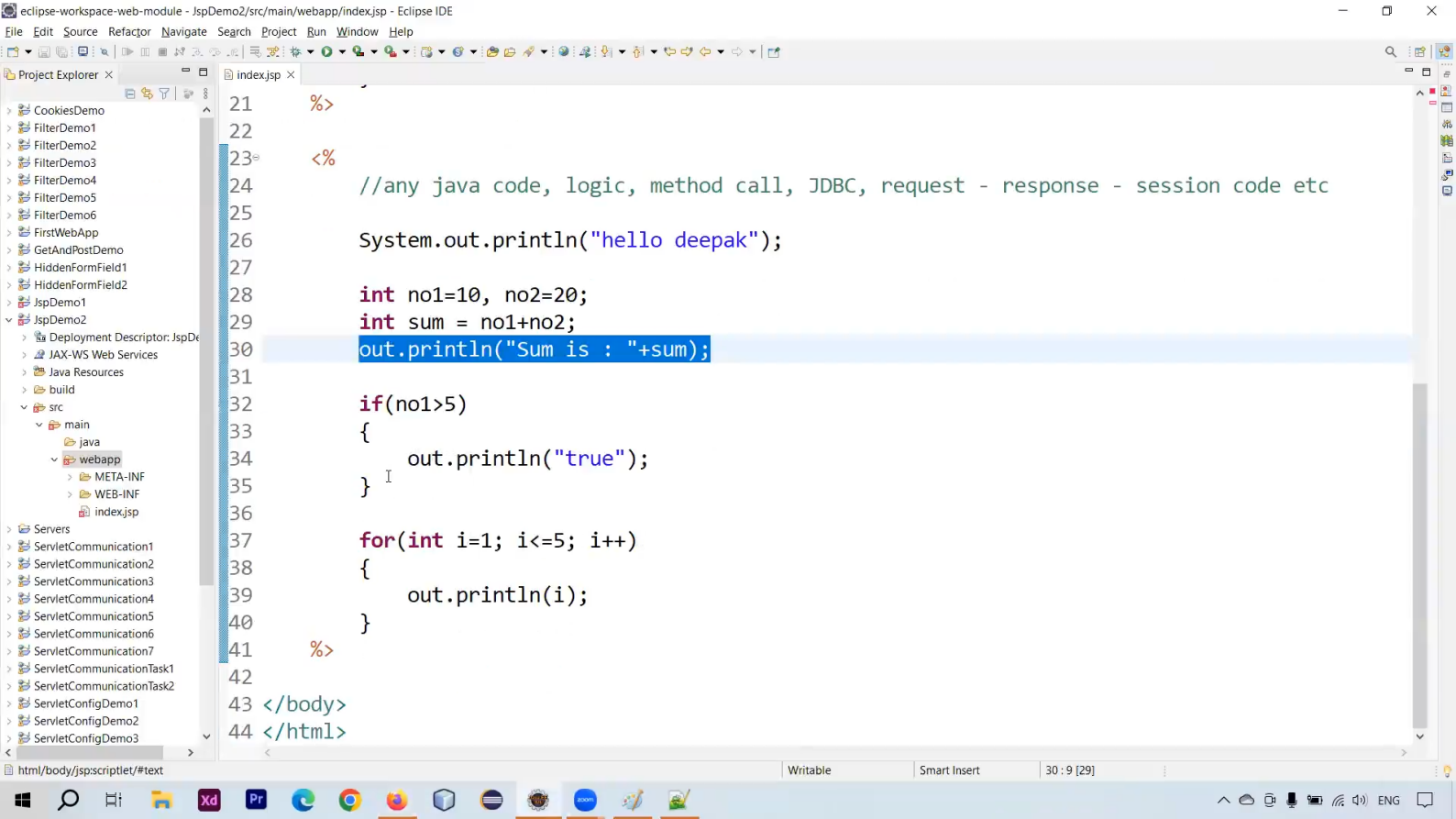
```jsp
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
    <h3> Welcome to Smart Programming </h3>

    <%
        String name = "Deepak";
    %>
    Welcome : <%= name %>
</body>
</html>
```

=> JSP Life Cycle :-

1. JSP Loading:
   = The JSP page is loaded into the server
2. JSP Parsing:
   = The JSP container parses the JSP page to identify Java code
3. JSP Translation:
   = The JSP page is translated into servlet page
4. Servlet Compilation:
   = The java servlet source code is compiled
5. Servlet Loading:
   = The compiled servlet class is loaded
6. Servlet Instantiation:
   = An instance of servlet is create
7. Servlet Initialization:
   = The servlet object is initialized
8. Servlet Processing:
   = The servlet handles the client requests
9. Servlet Destruction:
   = The servlet is destroyed dusring undeployment or server shutdown


JSP Tags / Elements
JSP Scripting Tags
<% ----- %>
Declarative Tag
<%! ----field or method ---- %>
Scriptlet Tag
<% ----java code ---- %>
Expression Tag
<%= ----statement---- %>
JSP Directive Tags
<%@ directive attribute="value" %>
Page Directive Tag
<%@ page attribute="value" %>
- import
- extends
- language
- contextType
- session
-isEllgnored
etc
Include Directive Tag
<%@ include file="resourceName" %>
Taglib Directive Tag
<%@ taglib uri="----" prefix="----" %>
JSP Action Tags
<jsp:---------- />
Standard Action Tags
<jsp:useBean>
<jsp:setProperty>
<jsp:getProperty>
<jsp:include>
<jsp:forward>
<jsp:param>
<jsp:declarative>
<jsp:scriptlet>
<jsp:expression>
<jsp:plugin>
<jsp:fallback>
<jsp:params>
Custom Action Tags
<prefix:tagname
attribute="value"
attribute="value" />
OR
<prefix:tagname
attribute="value"
attribute="value">
---- body code ----
</prefix:tagname>

```
=> JSP Tags :-
    -> JSP tags are special elements or directives used within the JSP document to embed Java code, perform actions or
    define custom behaviours
    -> JSP tags allow us to mix dynamic java code with static HTML content making it easier to create dynamic web pages

    -> There are 3 types of JSP tags :-
        1. JSP Scripting Tags
        2. JSP Directive Tags
        3. JSP Action Tags

    -> NOTE :-
        = There are many other libraries and tags that we can use in JSP to provide some additional functionalities i.e.
            - JSTL (JSP Standard Tag Library)
            - Spring MVC Tags
            - Struts Tags
            - JavaServer Faces (JSF) Tags
              etc
        = These tags can be used within the JSP using Taglib Directive Tag or Custom Action Tag

================================================================================================================================
```


eclipse-workspace-web-module - JspDemo2/src/main/webapp/index.jsp - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
CookiesDemo
FilterDemo1
FilterDemo2
FilterDemo3
FilterDemo4
FilterDemo5
FilterDemo6
FirstWebApp
GetAndPostDemo
HiddenFormField1
HiddenFormField2
JspDemo1
JspDemo2
Deployment Descriptor: JspDe
JAX-WS Web Services
Java Resources
build
src
main
java
webapp
META-INF
WEB-INF
index.jsp
Servers
ServletCommunication1
ServletCommunication2
ServletCommunication3
ServletCommunication4
ServletCommunication5
ServletCommunication6
ServletCommunication7
ServletCommunicationTask1
ServletCommunicationTask2
ServletConfigDemo1
ServletConfigDemo2
ServletConfigDemo3
index.jsp
21 %>
22
23 <%
24 //any java code, logic, method call, JDBC, request - response - session code etc
25
26 System.out.println("hello deepak");
27
28 int no1=10, no2=20;
29 int sum = no1+no2;
30 out.println("Sum is : "+sum);
31
32 if(no1>5)
33 {
34 out.println("true");
35 }
36
37 for(int i=1; i<=5; i++)
38 {
39 out.println(i);
40 }
41 %>
42
43 </body>
44 </html>
html/body/jsp:scriptlet/#text
Writable
Smart Insert
30 : 9 [29]
ENG

```
<%!
    int rollno=101;
    String name = "deepak";

    public int add(int a, int b)
    {
        return a+b;
    }
%>

<%= rollno %>
<%= name %>
<%= add(100, 200) %>
```

=> JSP Scripting Tags :-
    -> These tags are used to embed Java Code into JSP Page
    -> There are 3 types of JSP Scripting Tags :-
        1. Declaration Tag
        2. Scriptlet Tag
        3. Expression Tag

=> Declaration Tag :-
    -> This tag is used to declare variables or methods or classes or interfaces
    -> Syntax :-
        <%! ----variables, methods, classes, interfaces declaration---- %>
    -> If we declare variables or methods then they will be of instance or class level

=> Scriptlet Tag :-
    -> It is used to execute java source code in JSP
    -> Syntax :-
        <% ----java code---- %>
    -> Whatever code we provide in scriptlet tag will be provided in _jspService() method
    -> NOTE :
        = It's important to note that embedding too much java code in JSP can lead to less maintanable and less readable code. It's often recommended to use JavaBean or Servlets for complex business logic and keep JSP page for presentation (view)

=> Expression Tag :-
    -> It is used to print the values of variables or method calls directly to the output stream
    -> Syntax :-
        <%= ----variables or method calls---- %>
    -> The code placed within JSP expression tag will be written to the output stream of response and thus we dont need to use out.println() to write the data

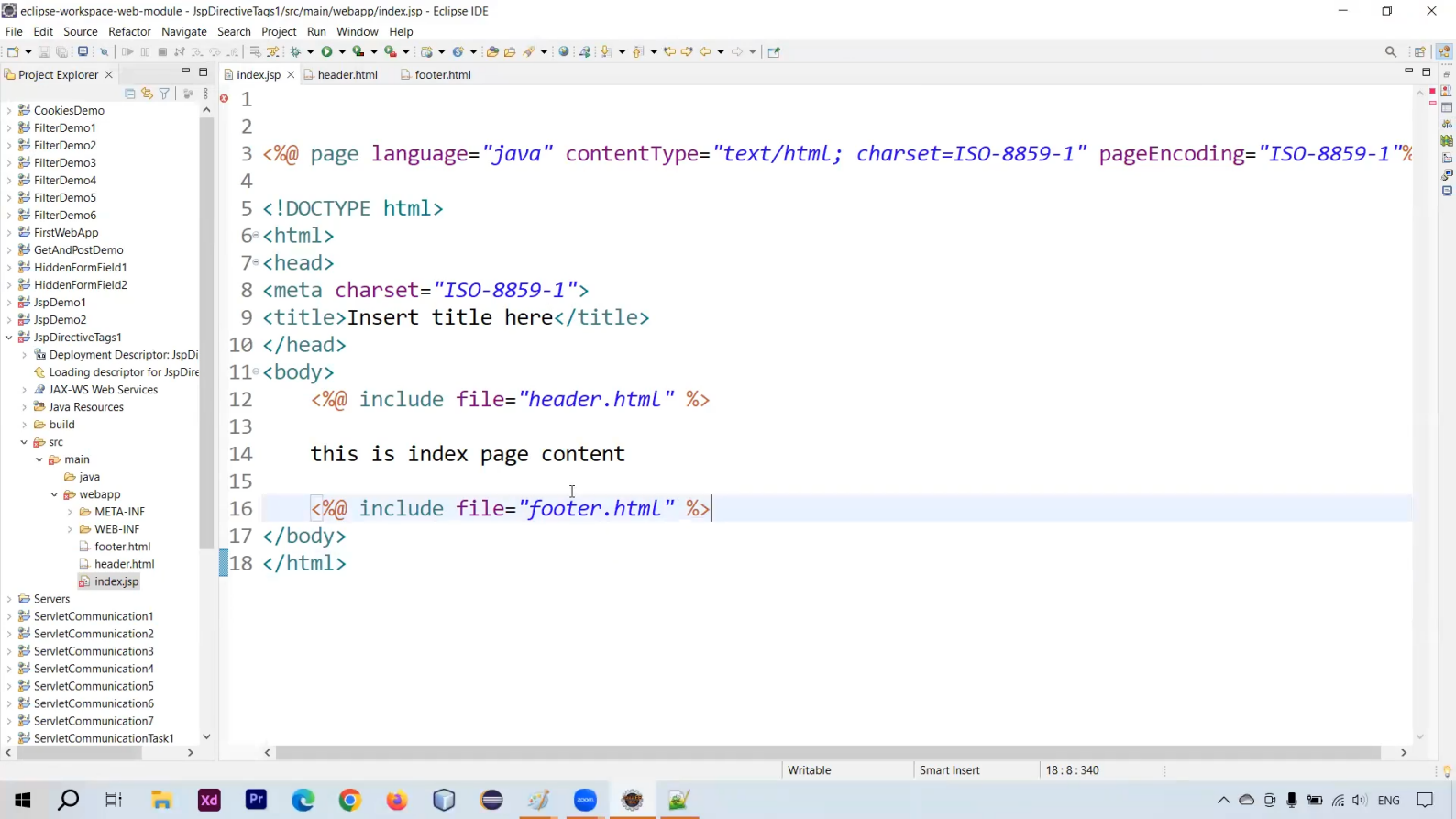
=> JSP Directive Tags :-
    -> JSP directive tags are used to provide the instructions or directives to the JSP container on how to handle the JSP page during compilation and execution
    -> Syntax :-
        <%@ directive attribute="value" %>

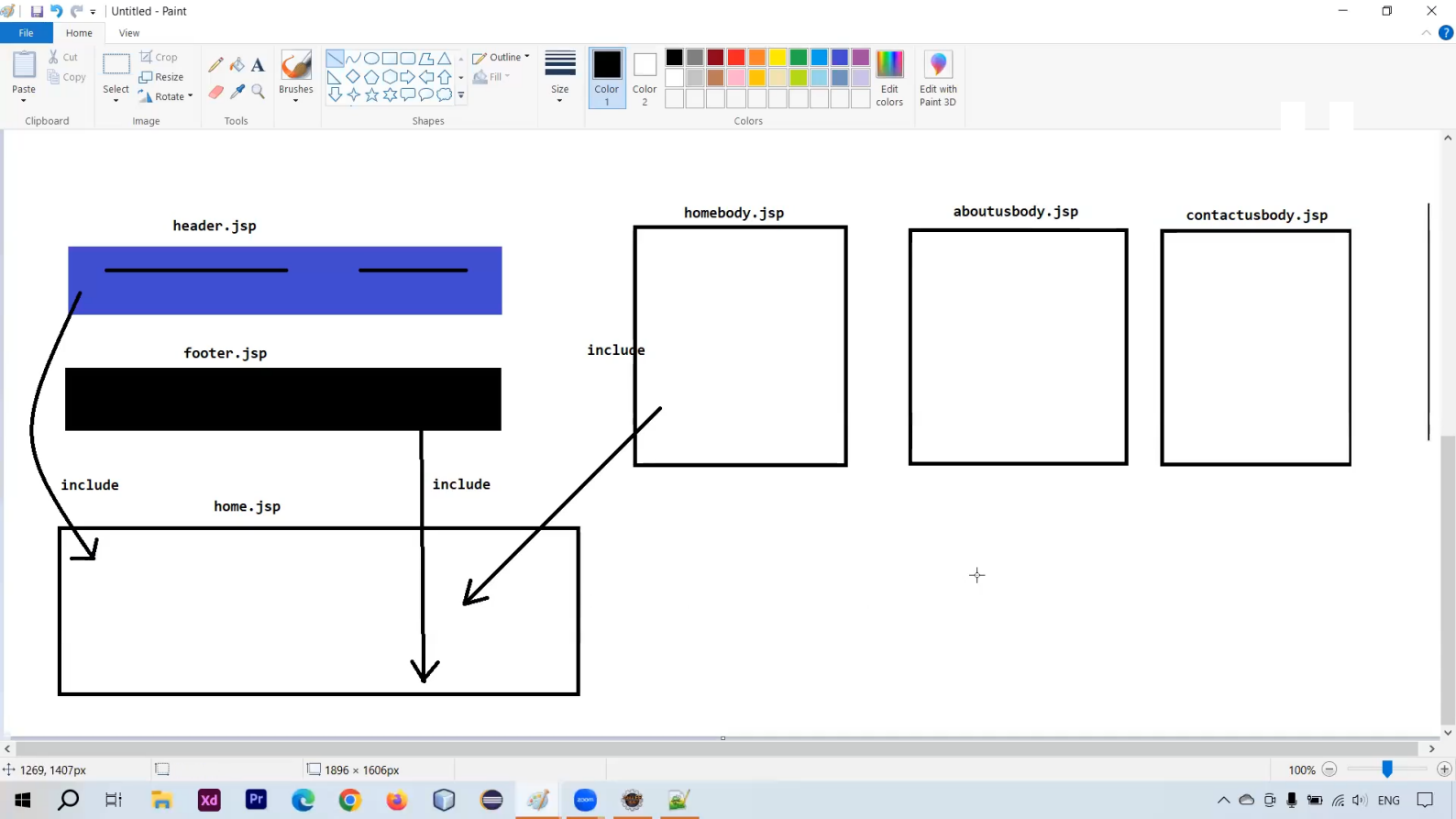
    -> There are 3 types of JSP directive tags :-
        1. Page Directive Tag
        2. Include Directive Tag
        3. Taglib Directive Tag

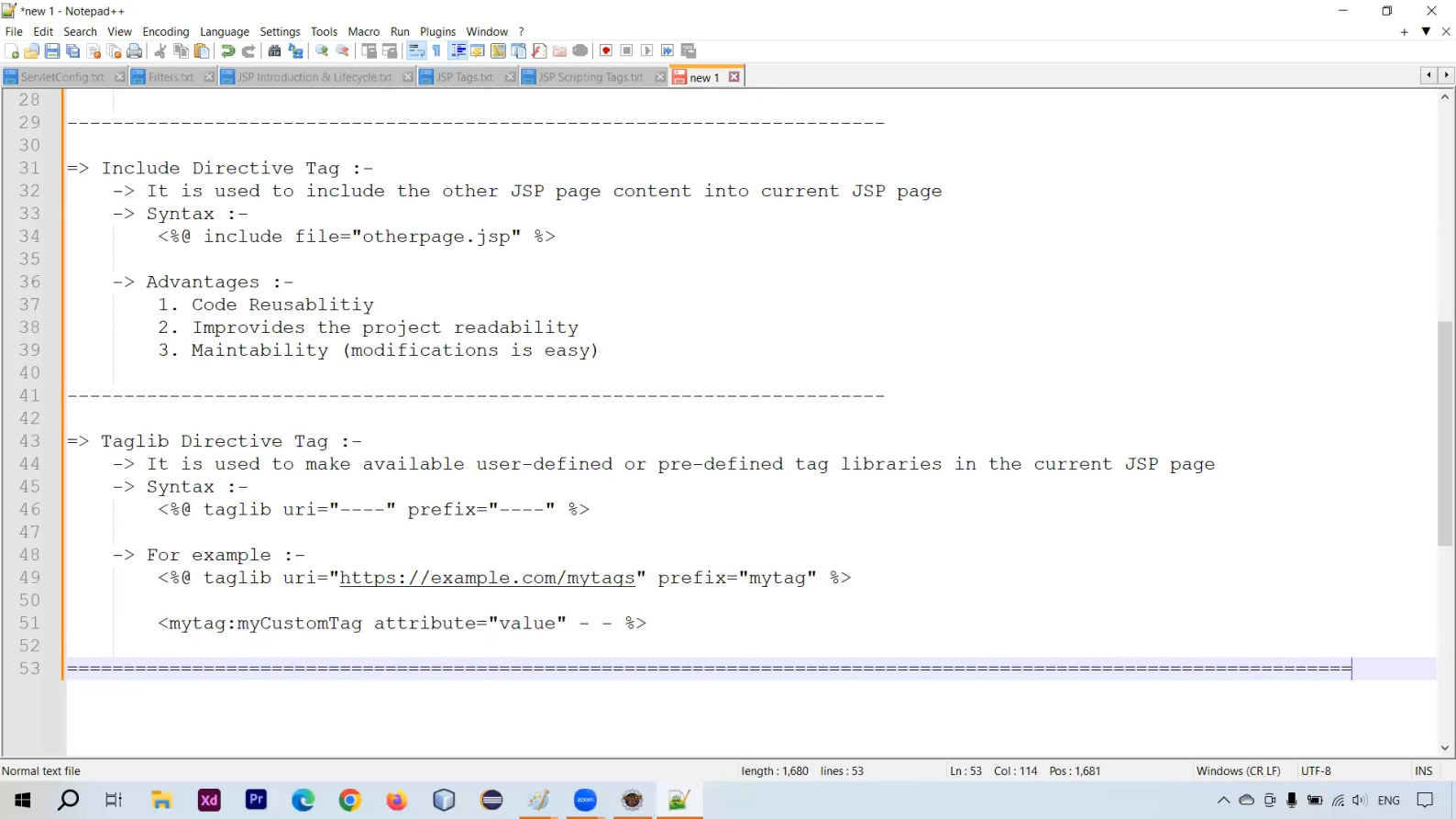
=> Page Directive Tag :-
    -> It is used to define various properties and settings for the current JSP page
    -> Syntax :-
        <%@ page attribute="value" - - %>

    -> Attributes in page directive tag :-
        1. import
        2. extends
        3. langauge
        4. contextType
        5. session
        6. isElIgnored
        7. errorPage
        8. isErrorPage
        etc

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


eclipse-workspace-web-module - JspDirectiveTags1/src/main/webapp/index.jsp - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
index.jsp
header.html
footer.html
CookiesDemo
FilterDemo1
FilterDemo2
FilterDemo3
FilterDemo4
FilterDemo5
FilterDemo6
FirstWebApp
GetAndPostDemo
HiddenFormField1
HiddenFormField2
JspDemo1
JspDemo2
JspDirectiveTags1
Deployment Descriptor: JspDi
Loading descriptor for JspDire
JAX-WS Web Services
Java Resources
build
src
main
java
webapp
META-INF
WEB-INF
footer.html
header.html
index.jsp
Servers
ServletCommunication1
ServletCommunication2
ServletCommunication3
ServletCommunication4
ServletCommunication5
ServletCommunication6
ServletCommunication7
ServletCommunicationTask1
1
2
3 <%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%
4
5 <!DOCTYPE html>
6 <html>
7 <head>
8 <meta charset="ISO-8859-1">
9 <title>Insert title here</title>
10 </head>
11 <body>
12 <%@ include file="header.html" %>
13
14 this is index page content
15
16 <%@ include file="footer.html" %>
17 </body>
18 </html>
Writable
Smart Insert
18 : 8 : 340
ENG


Untitled - Paint
File
Home
View
Cut
Copy
Paste
Crop
Resize
Rotate
Select
Brushes
Outline
Fill
Size
Color 1
Color 2
Edit colors
Edit with Paint 3D
Clipboard
Image
Tools
Shapes
Colors
header.jsp
footer.jsp
include
include
home.jsp
include
homebody.jsp
aboutusbody.jsp
contactusbody.jsp
1269, 1407px
1896 × 1606px
100%
ENG


*new 1 - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
ServletConfig.txt
Filters.txt
JSP Introduction & Lifecycle.txt
JSP Tags.txt
JSP Scripting Tags.txt
new 1
28
29 ----------
30
31 => Include Directive Tag :-
32 -> It is used to include the other JSP page content into current JSP page
33 -> Syntax :-
34 <%@ include file="otherpage.jsp" %>
35
36 -> Advantages :-
37 1. Code Reusablitiy
38 2. Improvides the project readability
39 3. Maintability (modifications is easy)
40
41 ----------
42
43 => Taglib Directive Tag :-
44 -> It is used to make available user-defined or pre-defined tag libraries in the current JSP page
45 -> Syntax :-
46 <%@ taglib uri="----" prefix="----" %>
47
48 -> For example :-
49 <%@ taglib uri="https://example.com/mytags" prefix="mytag" %>
50
51 <mytag:myCustomTag attribute="value" - - %>
52
53 ==========
Normal text file
length : 1,680 lines : 53
Ln : 53 Col : 114 Pos : 1,681
Windows (CR LF)
UTF-8
INS
ENG

=> JSP Action Tags :-

-> The main purpose of JSP Action tags is to reduce the java code from JSP page

-> They are often used to perform dynamic actions i.e. making decisions, iterating over collections etc without embedding the java code directly into JSP page

-> NOTE :-

= JSP Action Tags are mostly used in place of "Scriptlet Tag"

= It helps to separate the business layer from presentation layer in JSP page improving code organization and readability

-> Syntax :-

<jsp:actionName ---- >

-> Types of Action tags :-

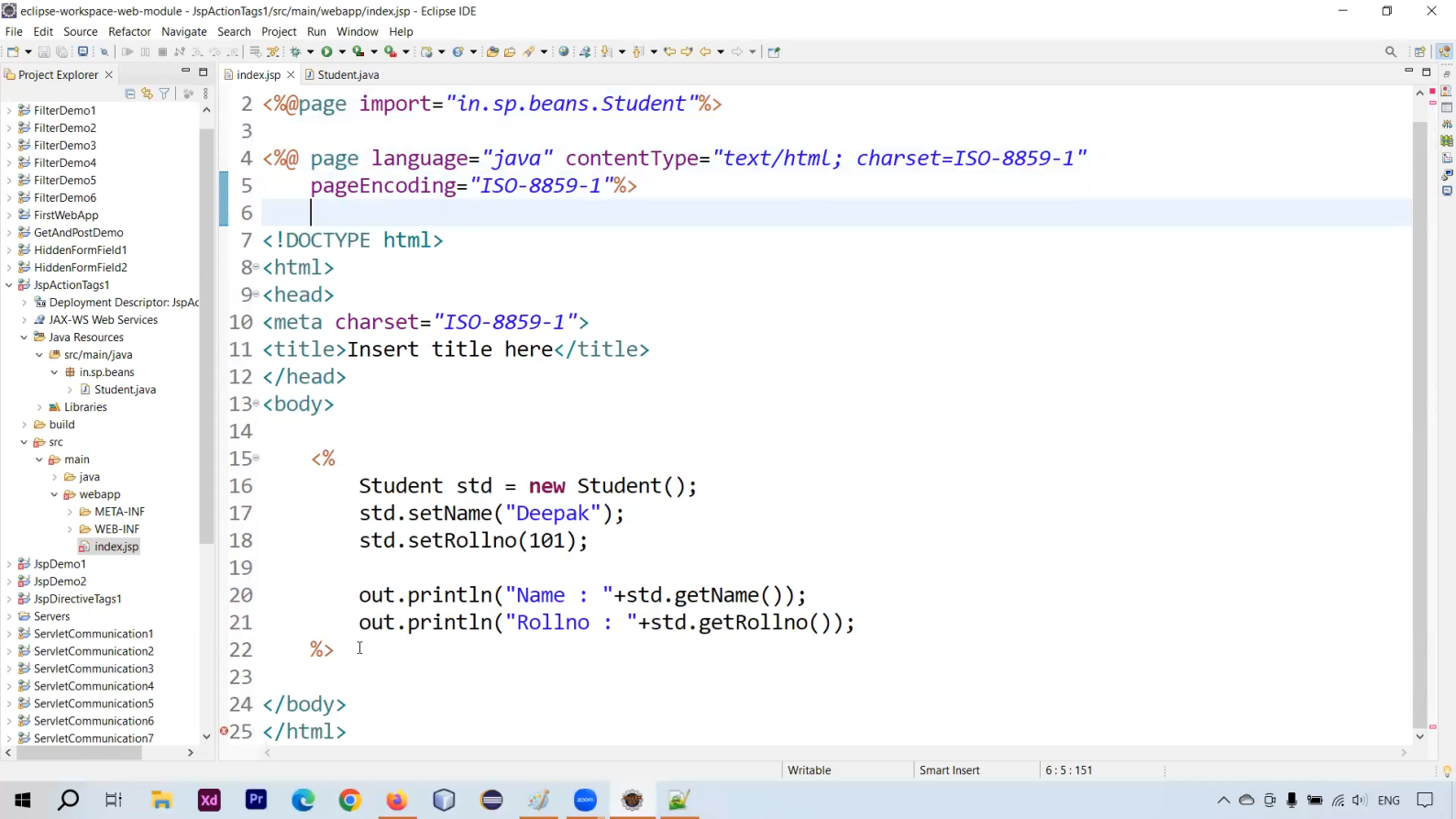
1. Standard Action Tag
2. Custom Action Tag

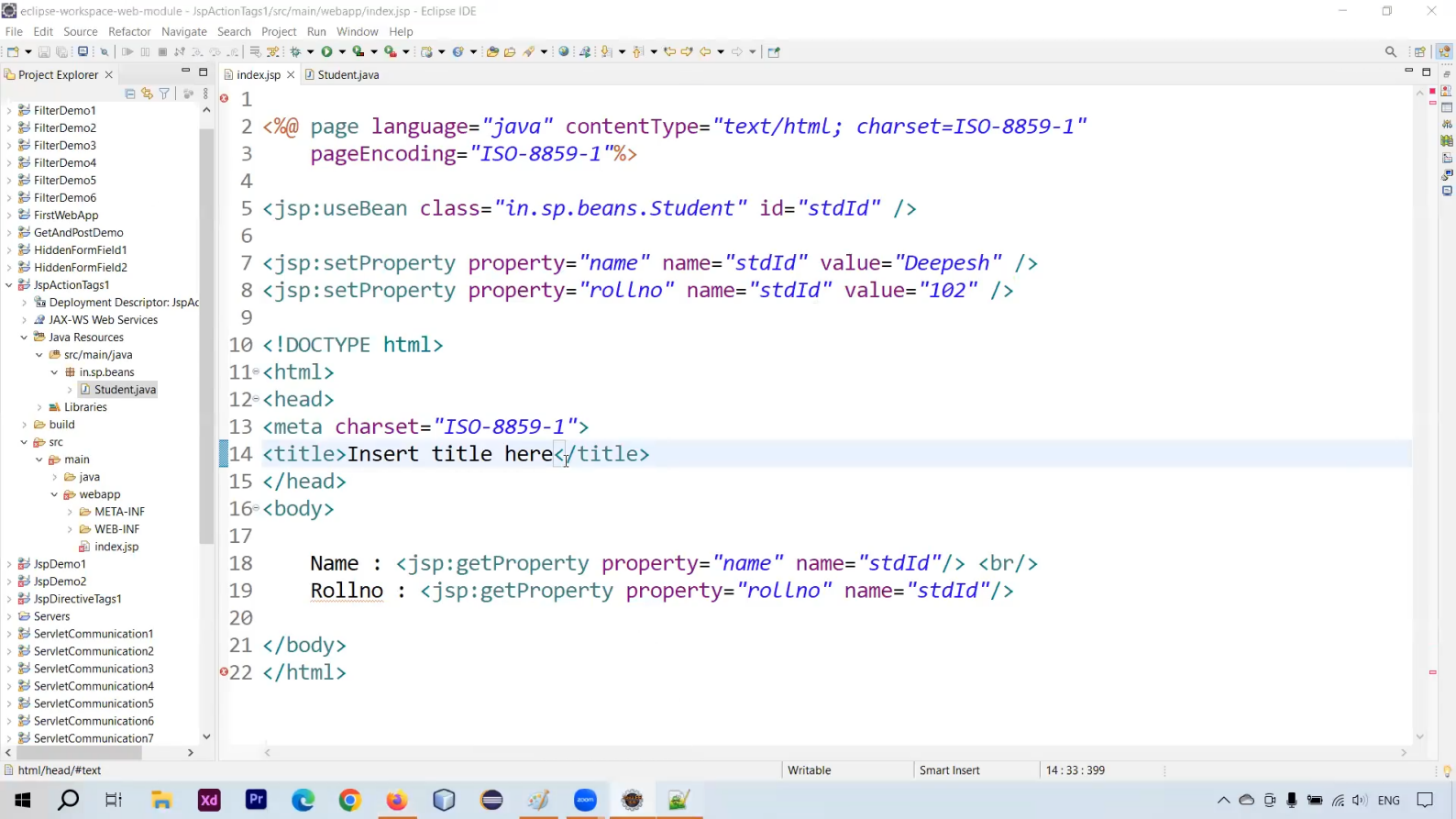
=> Standard Action Tag :-
    -> These are built-in JSP action tags provided by JSP specifications
    -> These tags are used to perform common tasks without writin java code directly in our JSP page
    -> Examples :-
        (Bean related tags)
        1. <jsp:useBean>
        2. <jsp:setProperty>
        3. <jsp:getProperty>

        (Control flow tags)
        4. <jsp:include>
        5. <jsp:forward>
        6. <jsp:param>

        (Scripting elements/tags)
        7. <jsp:declaration>
        8. <jsp:scriptlet>
        9. <jsp:expression>

        (Applets related tags)
        10. <jsp:plugin>
        11. <jsp:fallback>
        12. <jsp:params>


eclipse-workspace-web-module - JspActionTags1/src/main/webapp/index.jsp - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
index.jsp
Student.java
FilterDemo1
FilterDemo2
FilterDemo3
FilterDemo4
FilterDemo5
FilterDemo6
FirstWebApp
GetAndPostDemo
HiddenFormField1
HiddenFormField2
JspActionTags1
Deployment Descriptor: JspAc
JAX-WS Web Services
Java Resources
src/main/java
in.sp.beans
Student.java
Libraries
build
src
main
java
webapp
META-INF
WEB-INF
index.jsp
JspDemo1
JspDemo2
JspDirectiveTags1
Servers
ServletCommunication1
ServletCommunication2
ServletCommunication3
ServletCommunication4
ServletCommunication5
ServletCommunication6
ServletCommunication7
2 <%@page import="in.sp.beans.Student"%>
3
4 <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
5 pageEncoding="ISO-8859-1"%>
6
7 <!DOCTYPE html>
8 <html>
9 <head>
10 <meta charset="ISO-8859-1">
11 <title>Insert title here</title>
12 </head>
13 <body>
14
15 <%
16 Student std = new Student();
17 std.setName("Deepak");
18 std.setRollno(101);
19
20 out.println("Name : "+std.getName());
21 out.println("Rollno : "+std.getRollno());
22 %>
23
24 </body>
25 </html>
Writable
Smart Insert
6 : 5 : 151
ENG


eclipse-workspace-web-module - JspActionTags1/src/main/webapp/index.jsp - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
index.jsp
Student.java
FilterDemo1
FilterDemo2
FilterDemo3
FilterDemo4
FilterDemo5
FilterDemo6
FirstWebApp
GetAndPostDemo
HiddenFormField1
HiddenFormField2
JspActionTags1
Deployment Descriptor: JspAc
JAX-WS Web Services
Java Resources
src/main/java
in.sp.beans
Student.java
Libraries
build
src
main
java
webapp
META-INF
WEB-INF
index.jsp
JspDemo1
JspDemo2
JspDirectiveTags1
Servers
ServletCommunication1
ServletCommunication2
ServletCommunication3
ServletCommunication4
ServletCommunication5
ServletCommunication6
ServletCommunication7
1
2 <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
3 pageEncoding="ISO-8859-1"%>
4
5 <jsp:useBean class="in.sp.beans.Student" id="stdId" />
6
7 <jsp:setProperty property="name" name="stdId" value="Deepesh" />
8 <jsp:setProperty property="rollno" name="stdId" value="102" />
9
10 <!DOCTYPE html>
11 <html>
12 <head>
13 <meta charset="ISO-8859-1">
14 <title>Insert title here</title>
15 </head>
16 <body>
17
18 Name : <jsp:getProperty property="name" name="stdId"/>
19 Rollno : <jsp:getProperty property="rollno" name="stdId"/>
20
21 </body>
22 </html>
html/head/#text
Writable
Smart Insert
14 : 33 : 399
ENG

=> What is difference between Include Directive Tag and JSP Action include Tag (<jsp:include>)

1. Include Directive Tag is used for static pages
   Action include Tag is used for dynamic pages

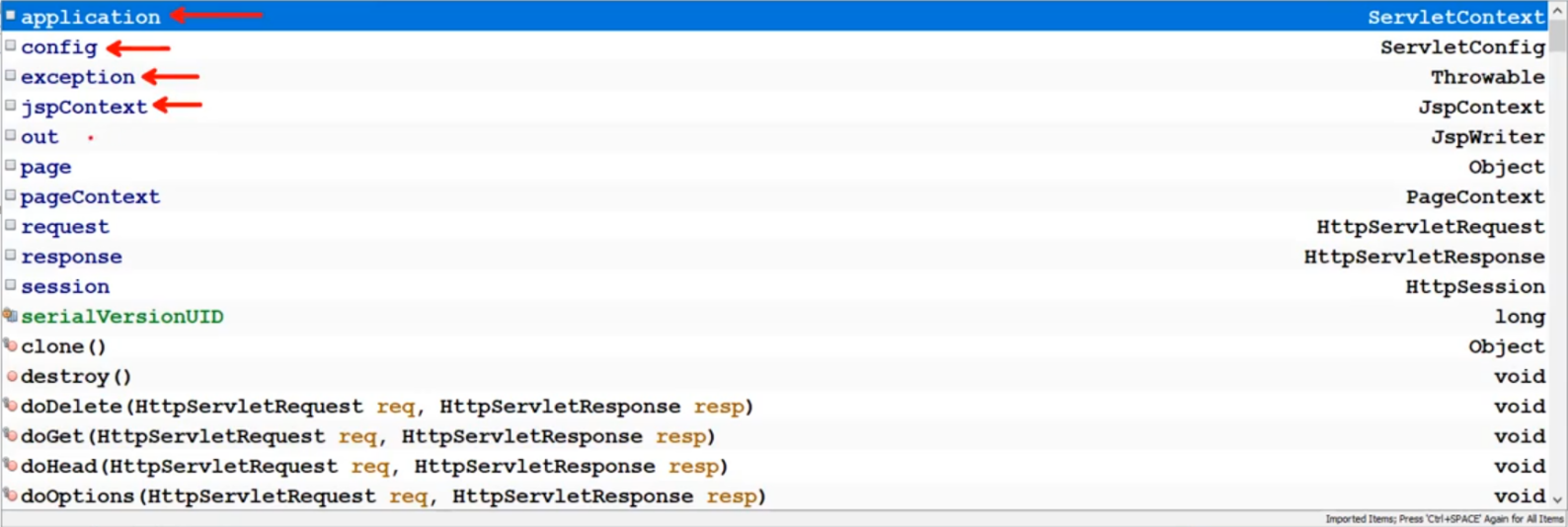
2. Include Directive Tag includes the content of the target resource
   Action include Tag includes the target response in the present JSP page

3. Include Directive Tag is evaluated at translation time
   Action include Tag is evaluated at request processing time

=> Custom Action Tag :-
-> Already many standard action tags are created but sometimes we need to create our own action tags, then we can use custom action tag
-> Advantages of custom action tag :-
1. It eliminates the need of scriptlet tag
2. It separates the business layer from presentation layer
3. Reusablity of the code increase

-> Syntax :-

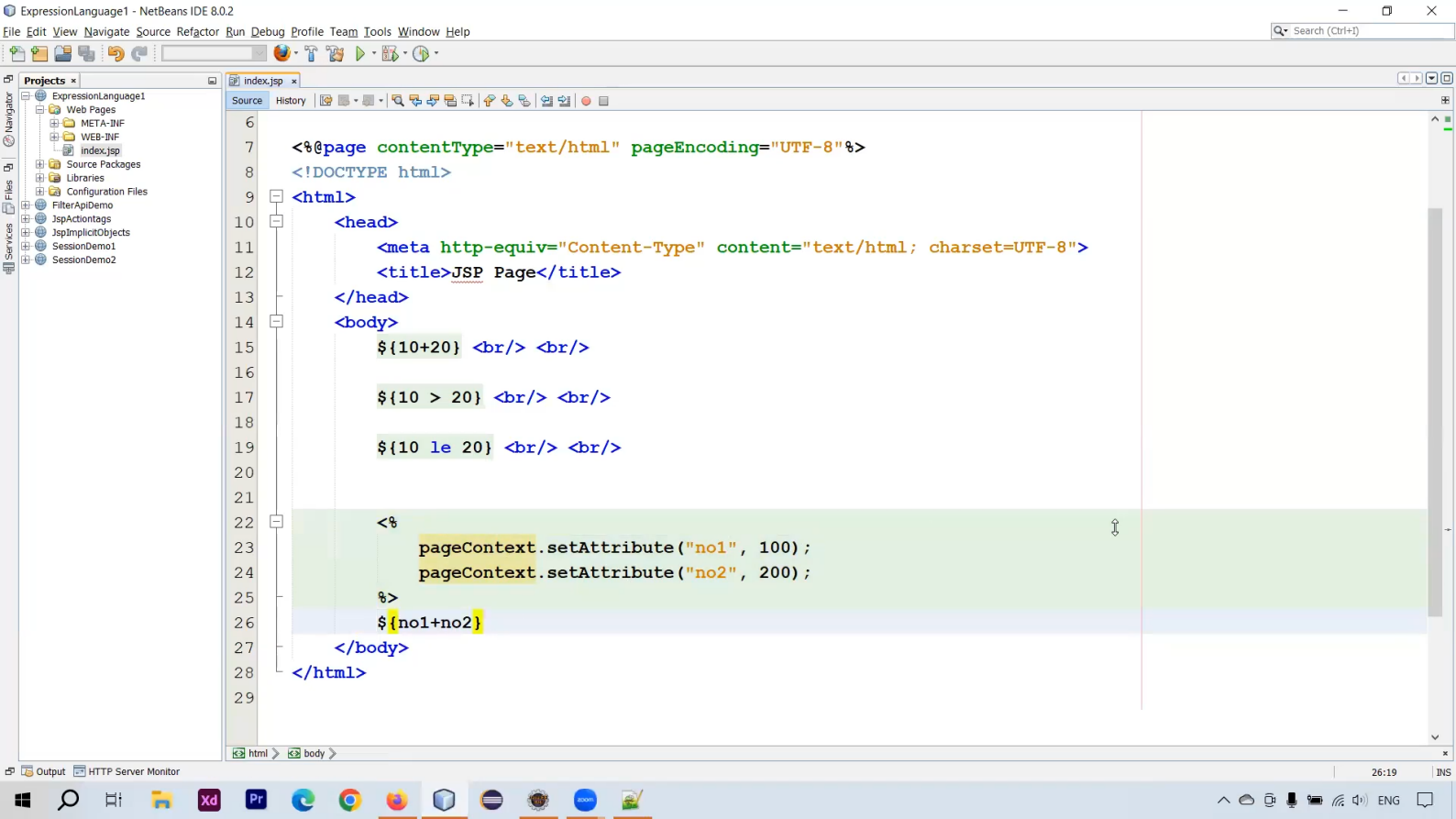
```
<prefix:tagname attribute="value" - - />
            OR
<prefix:tagname attribute="value" - ->
    //body
</prefix:tagname>
```


application ServletContext
config ServletConfig
exception Throwable
jspContext JspContext
out JspWriter
page Object
pageContext PageContext
request HttpServletRequest
response HttpServletResponse
session HttpSession
serialVersionUID long
clone() Object
destroy() void
doDelete(HttpServletRequest req, HttpServletResponse resp) void
doGet(HttpServletRequest req, HttpServletResponse resp) void
doHead(HttpServletRequest req, HttpServletResponse resp) void
doOptions(HttpServletRequest req, HttpServletResponse resp) void
Imported Items; Press 'Ctrl+SPACE' Again for All Items

=> JSP Implicit Objects :-
-> JSP implicit objects are pre-defined objects that are automatically available for use in our JSP pages without the need to explicitly declare or instantiate them
-> These objects are created by JSP container
-> There are total 9 implicit objects in JSP :-

1. out - JspWriter <class>
2. request - HttpServletRequest <interface>
3. response - HttpServletResponse <interface>
4. session - HttpSession <interface>
5. application - ServletContext <interface>
6. config - ServletConfig <interface>
7. pageContext - PageContext <class>
8. page - Object <refers to the JSP page itself>
9. exception - Throwable <class>

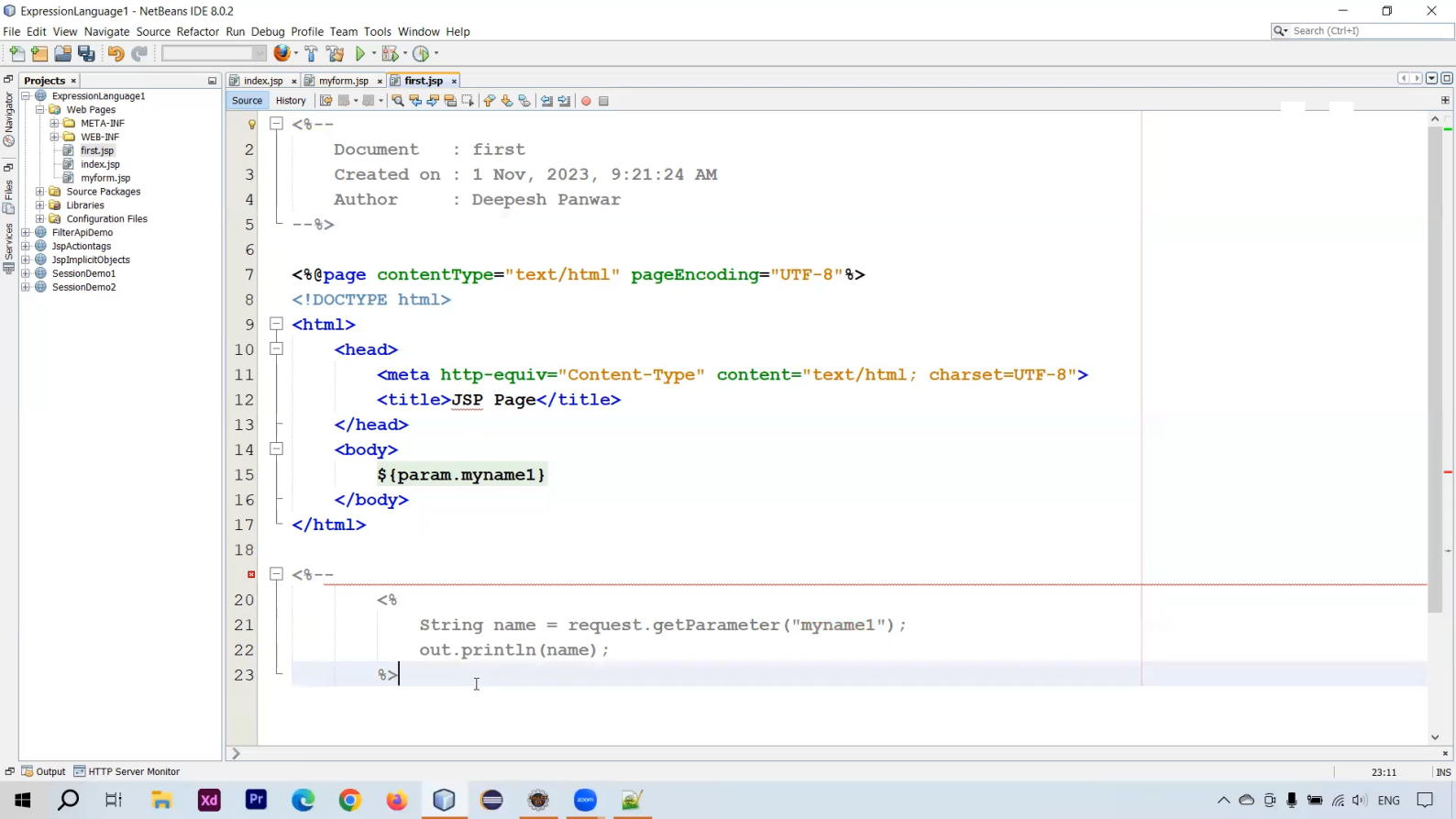
```
=> Expression Language (EL) :-
    -> It is a scripting language used in web development commonly associated with JSP and Java EE applications
    -> It is used to access and manipulate data stored in java objects i.e. JavaBeans, request, session, application etc
    -> It is designed to simplify the process of integrating dynamic data into our web page
    -> It was introduced in JSP 2.0 version

    -> Syntax :-
        ${expression}

    -> EL contains various elements which are as follows :-
        = Operators
            +, -, *, /, ==, !=, &&, ||, [], () etc

        = Reserved Words
            true, false, null, empty, eq (equal), ne (not equal), lt (less than), le (less than or equal to), gt
            (greater than), ge (greater than or equal to) etc

        = Implicit Objects
            - pageScope
            - requestScope
            - sessionScope
            - applicationScope
            - param
            - paramValues
            - pageContext
            etc

==========================================================================================================================
```

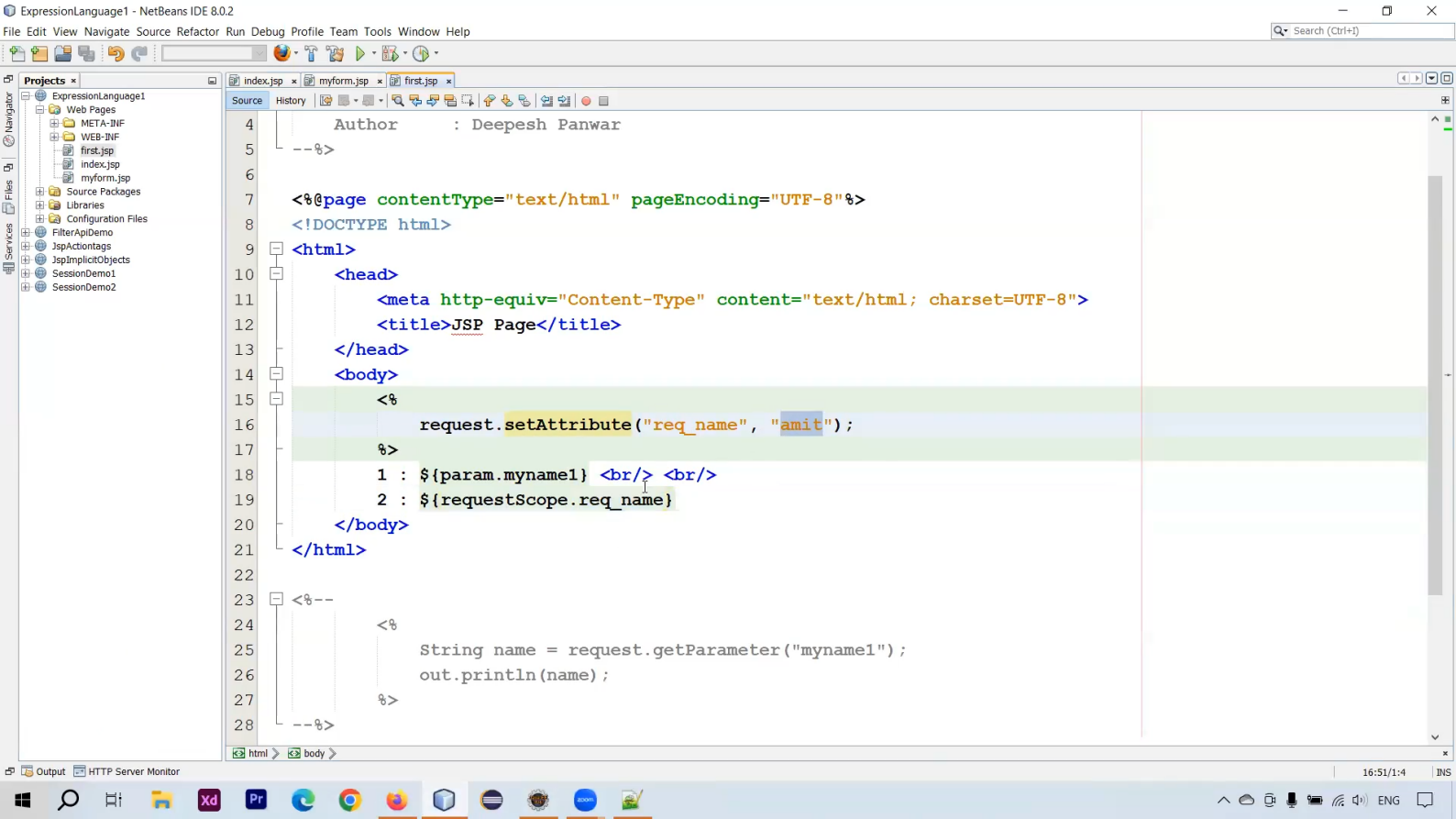

ExpressionLanguage1 - NetBeans IDE 8.0.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
Projects
ExpressionLanguage1
Web Pages
META-INF
WEB-INF
index.jsp
Source Packages
Libraries
Configuration Files
FilterApiDemo
JspActiontags
JspImplicitObjects
SessionDemo1
SessionDemo2
index.jsp
Source
History
6
7 <%@page contentType="text/html" pageEncoding="UTF-8"%>
8 <!DOCTYPE html>
9 <html>
10 <head>
11 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
12 <title>JSP Page</title>
13 </head>
14 <body>
15 ${10+20}

16
17 ${10 > 20}

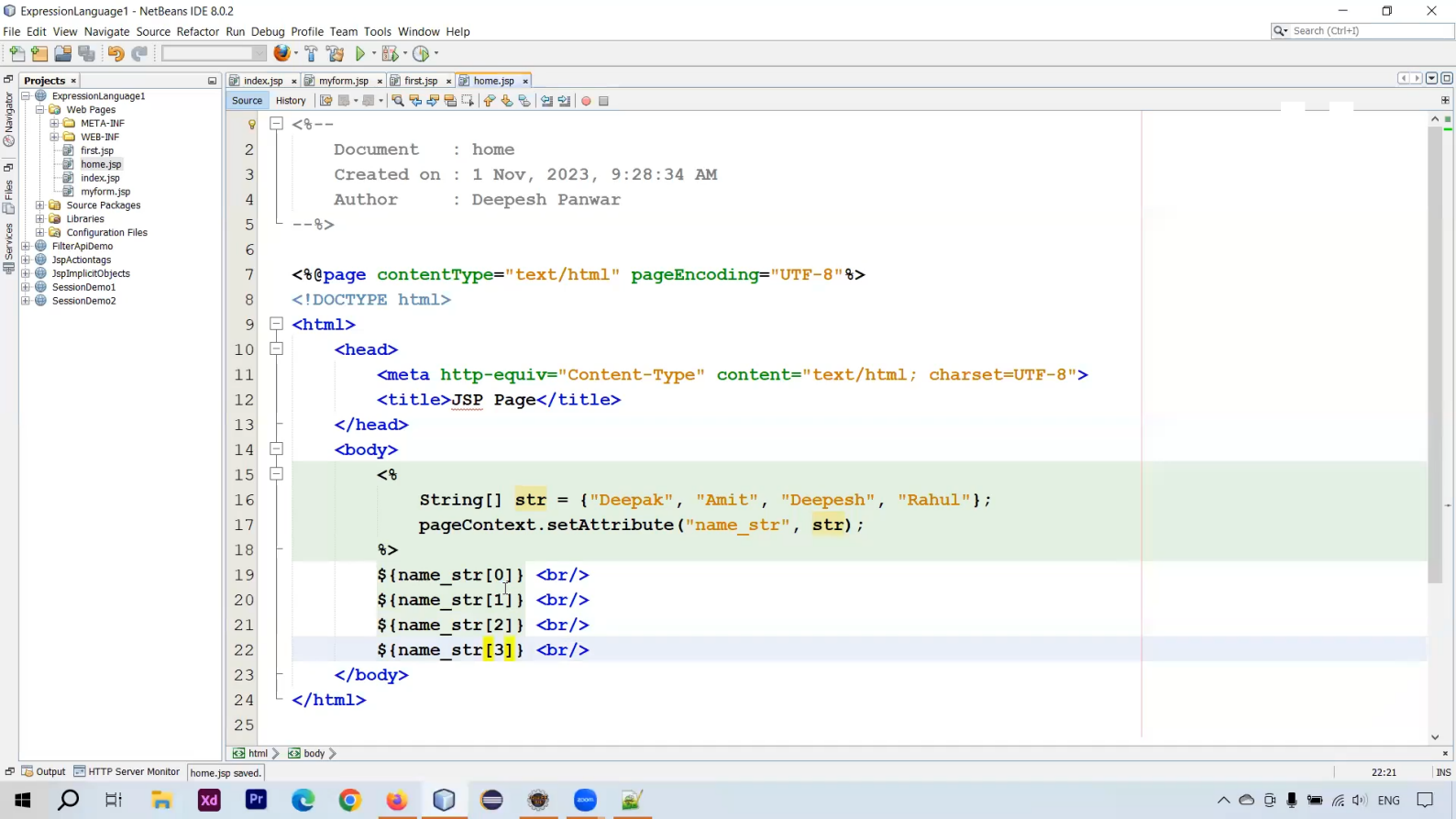
18
19 ${10 le 20}

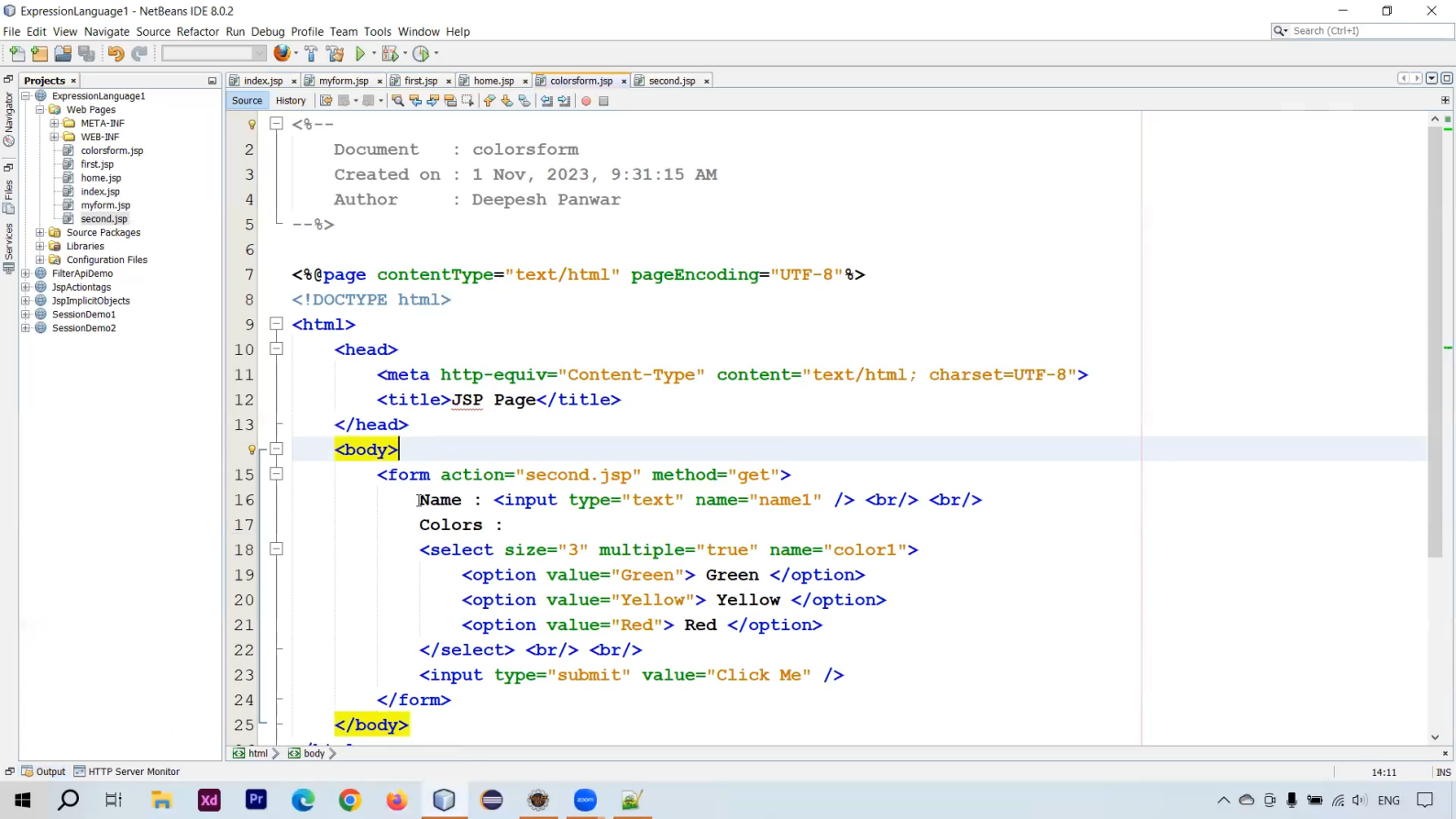
20
21
22 <%
23 pageContext.setAttribute("no1", 100);
24 pageContext.setAttribute("no2", 200);
25 %>
26 ${no1+no2}
27 </body>
28 </html>
29
html
body
Output
HTTP Server Monitor
26:19
INS
ENG


ExpressionLanguage1 - NetBeans IDE 8.0.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
Projects
ExpressionLanguage1
Web Pages
META-INF
WEB-INF
first.jsp
index.jsp
myform.jsp
Source Packages
Libraries
Configuration Files
FilterApiDemo
JspActiontags
JspImplicitObjects
SessionDemo1
SessionDemo2
index.jsp
myform.jsp
first.jsp
Source
History
<%--
Document : first
Created on : 1 Nov, 2023, 9:21:24 AM
Author : Deepesh Panwar
--%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP Page</title>
</head>
<body>
${param.myname1}
</body>
</html>
<%--
<%
String name = request.getParameter("myname1");
out.println(name);
%>
Output
HTTP Server Monitor
23:11
INS
ENG


ExpressionLanguage1 - NetBeans IDE 8.0.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
Projects
ExpressionLanguage1
Web Pages
META-INF
WEB-INF
first.jsp
index.jsp
myform.jsp
Source Packages
Libraries
Configuration Files
FilterApiDemo
JspActiontags
JspImplicitObjects
SessionDemo1
SessionDemo2
index.jsp
myform.jsp
first.jsp
Source
History
Author : Deepesh Panwar
--%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP Page</title>
</head>
<body>
<%
request.setAttribute("req_name", "amit");
%>
1 : ${param.myname1}

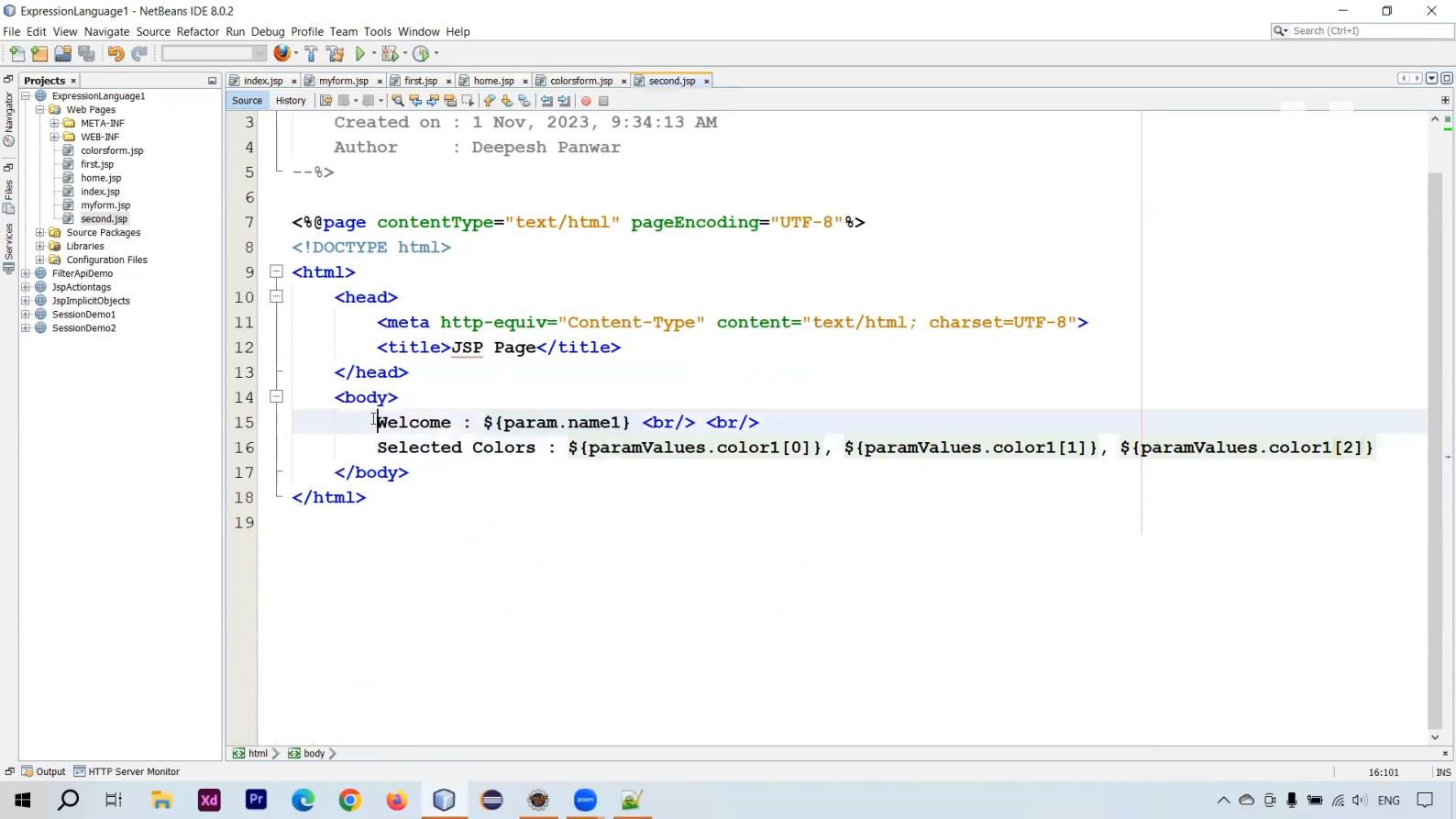
2 : ${requestScope.req_name}
</body>
</html>
<%--
<%
String name = request.getParameter("myname1");
out.println(name);
%>
--%>
html
body
Output
HTTP Server Monitor
16:51/1:4
INS
ENG


ExpressionLanguage1 - NetBeans IDE 8.0.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
Projects
ExpressionLanguage1
Web Pages
META-INF
WEB-INF
first.jsp
home.jsp
index.jsp
myform.jsp
Source Packages
Libraries
Configuration Files
FilterApiDemo
JspActiontags
JspImplicitObjects
SessionDemo1
SessionDemo2
index.jsp
myform.jsp
first.jsp
home.jsp
Source
History
<%--
Document : home
Created on : 1 Nov, 2023, 9:28:34 AM
Author : Deepesh Panwar
--%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP Page</title>
</head>
<body>
<%
String[] str = {"Deepak", "Amit", "Deepesh", "Rahul"};
pageContext.setAttribute("name_str", str);
%>
${name_str[0]}
${name_str[1]}
${name_str[2]}
${name_str[3]}
</body>
</html>
html
body
Output
HTTP Server Monitor
home.jsp saved.
22:21
INS
ENG


ExpressionLanguage1 - NetBeans IDE 8.0.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
Projects
ExpressionLanguage1
Web Pages
META-INF
WEB-INF
colorsform.jsp
first.jsp
home.jsp
index.jsp
myform.jsp
second.jsp
Source Packages
Libraries
Configuration Files
FilterApiDemo
JspActiontags
JspImplicitObjects
SessionDemo1
SessionDemo2
index.jsp
myform.jsp
first.jsp
home.jsp
colorsform.jsp
second.jsp
Source
History
<%--
Document : colorsform
Created on : 1 Nov, 2023, 9:31:15 AM
Author : Deepesh Panwar
--%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP Page</title>
</head>
<body>
<form action="second.jsp" method="get">
Name : <input type="text" name="name1" />

Colors :
<select size="3" multiple="true" name="color1">
<option value="Green"> Green </option>
<option value="Yellow"> Yellow </option>
<option value="Red"> Red </option>
</select>

<input type="submit" value="Click Me" />
</form>
</body>
html
body
Output
HTTP Server Monitor
14:11
INS
ENG


ExpressionLanguage1 - NetBeans IDE 8.0.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
Projects
ExpressionLanguage1
Web Pages
META-INF
WEB-INF
colorsform.jsp
first.jsp
home.jsp
index.jsp
myform.jsp
second.jsp
Source Packages
Libraries
Configuration Files
FilterApiDemo
JspActiontags
JspImplicitObjects
SessionDemo1
SessionDemo2
index.jsp
myform.jsp
first.jsp
home.jsp
colorsform.jsp
second.jsp
Source
History
Created on : 1 Nov, 2023, 9:34:13 AM
Author : Deepesh Panwar
--%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP Page</title>
</head>
<body>
Welcome : ${param.name1}

Selected Colors : ${paramValues.color1[0]}, ${paramValues.color1[1]}, ${paramValues.color1[2]}
</body>
</html>
html
body
Output
HTTP Server Monitor
16:101
INS
ENG

=> JSTL (JSP Standard Tag Library) :-
   -> JSTL is a standard library for JSP which was developed by Java Community Process (JCP)
   -> JSTL is a collection of tags designed to simplify and enhance the JSP development

   -> There are total 5 types of JSTL tags :-
      1. Core Tags
      2. XML tags
      3. Formatted Tags
      4. SQL Tags
      5. Function Tags

   -> NOTE :-
      = Most commonly used JSTL tags are "Core Tags"
      = If we want to use JSTL tags in our project then we have to add one jar file i.e. jstl.jar

   -> To use JSTL tags, we have to use taglib directive tags
      <%@ taglib uri="----" prefix="----" %>

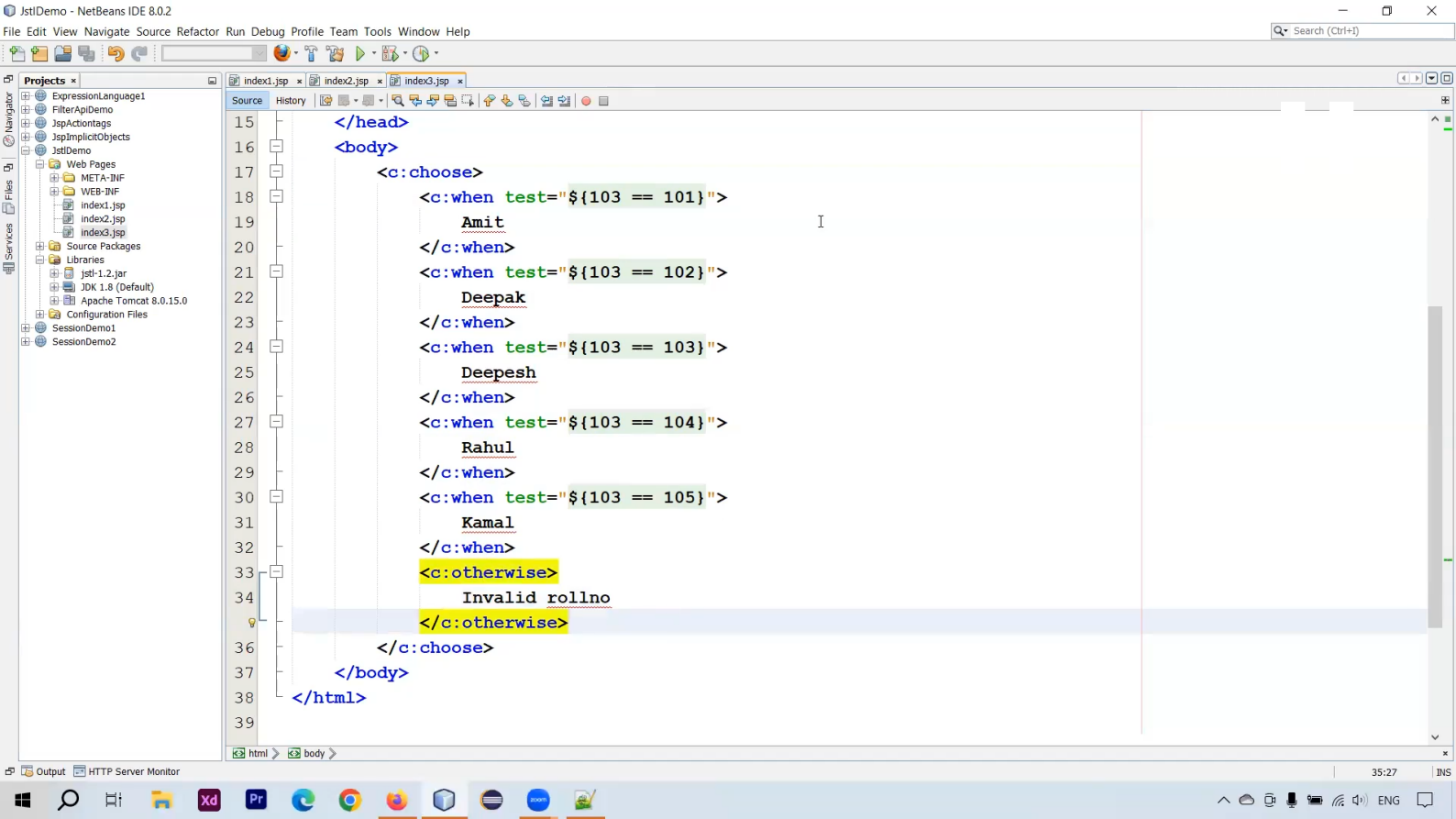
=> Core Tags :-
    -> These tags are commonly used for tasks like control statements, looping, variable manipulation & outputting content
    -> To use core tags in JSTP we have to include taglib directive tag as below :-
        <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

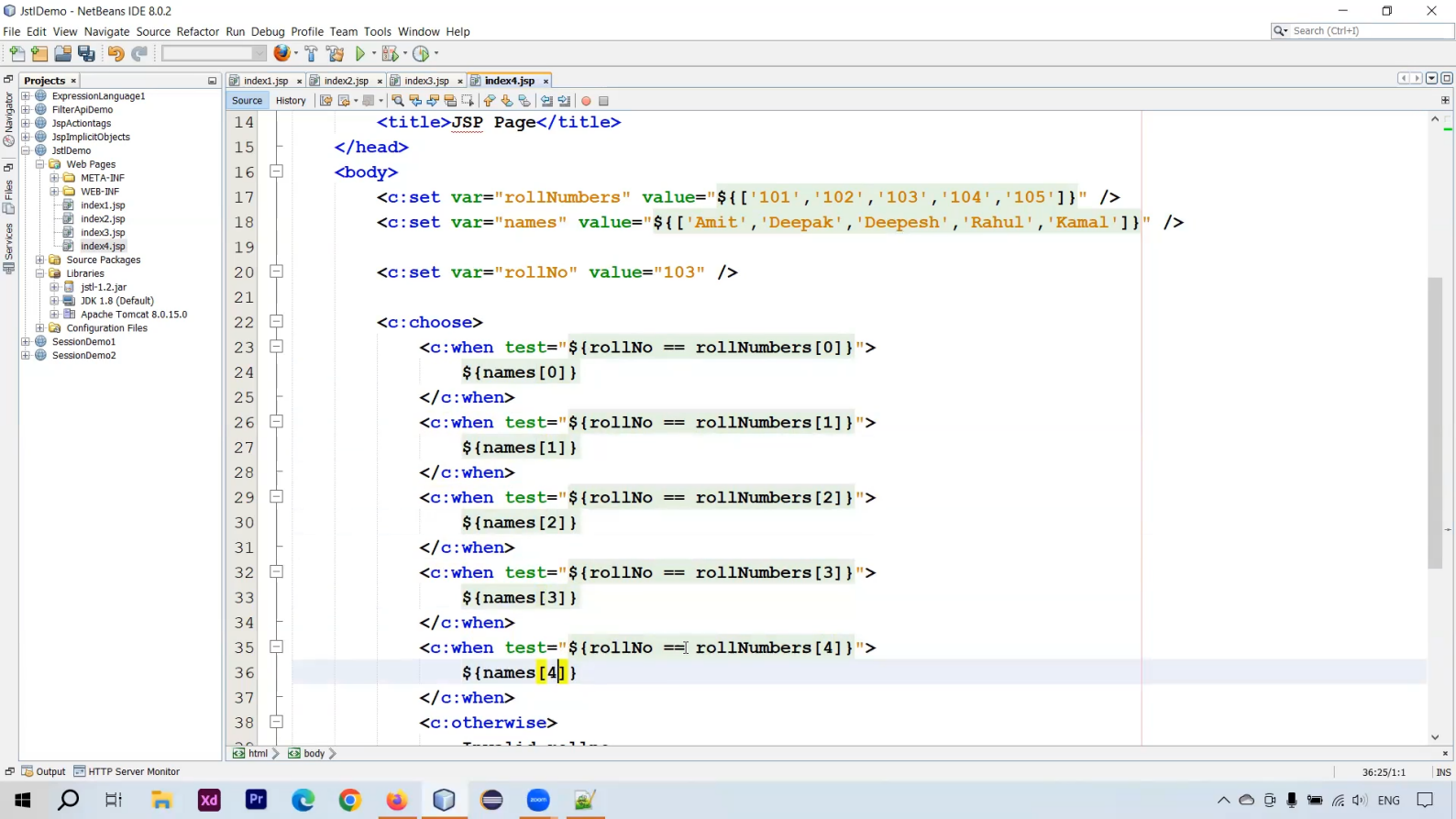
    -> In core tags we have approx 14 tags :-
        1. General Purpose Tags :-
            <c:set>, <c:out>, <c:remove>, <c:catch>
        2. Conditional Tags :-
            <c:if>, <c:choose>, <c:when>, <c:otherwise> (last 3 tags are similar to switch statement)
        3. Iterative Tags :-
            <c:forEach>, <c:forTokens>
        4. URL Based Tags :-
            <c:url>, <c:import>, <c:redirect>

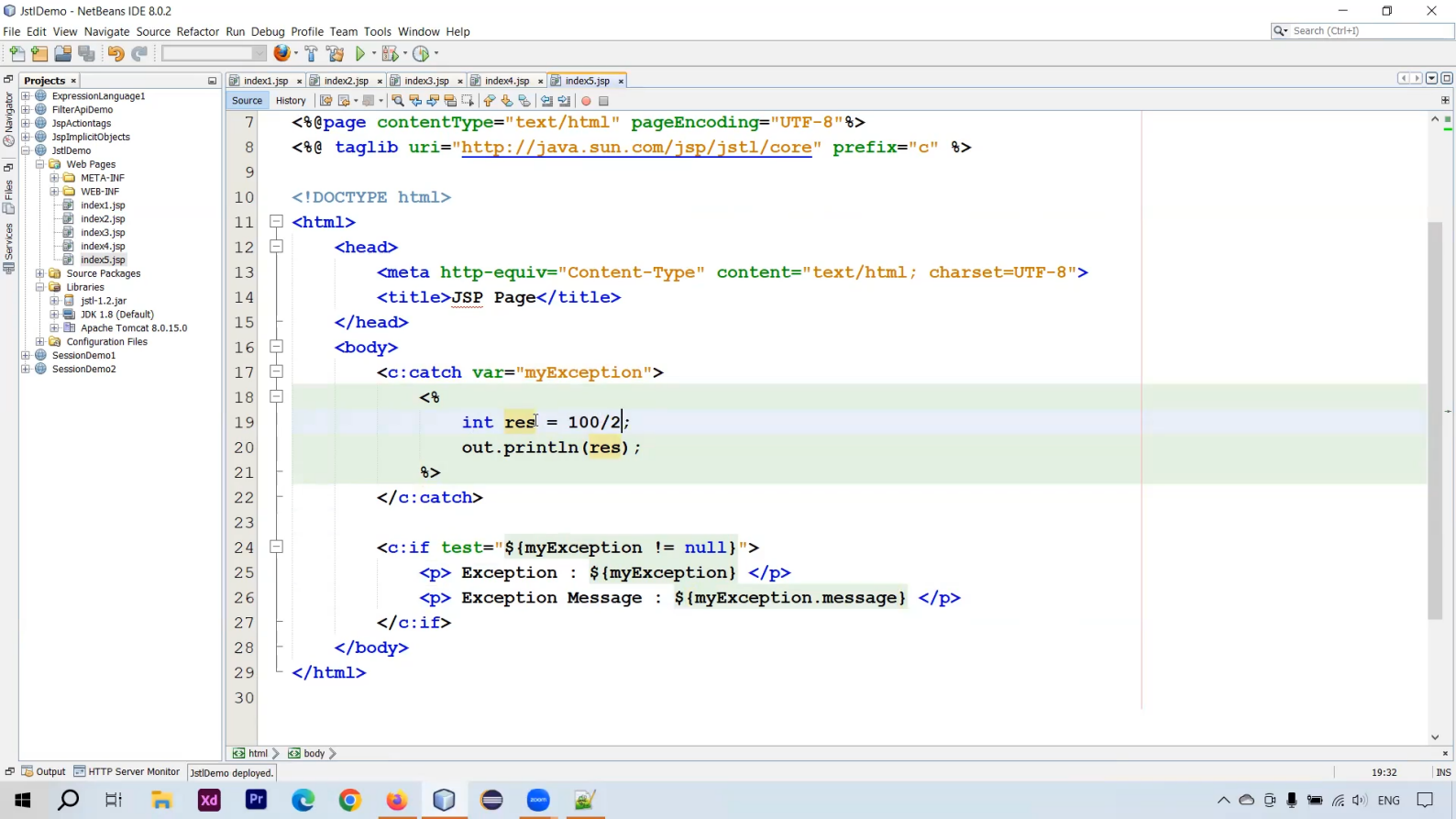
```jsp
<%@page contentType="text/html" pageEncoding="UTF-8"%>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <c:set var="name1" value="Deepak Panwar" />

        <c:out value="${name1}" />
    </body>
</html>
```

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <c:set var="name1" value="Deepak Panwar" />

        1 : <c:out value="${name1}" /> <br/>
        2 : ${name1} <br/> <br/>


        <c:remove var="name1" />


        1 : <c:out value="${name1}" /> <br/>
        2 : ${name1}
    </body>
</html>
```

```jsp
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <c:if test="${10 < 20}">
            hello 1
        </c:if>

        <c:if test="${10 gt 20}">
            hello 2
        </c:if>
    </body>
</html>
```


JstlDemo - NetBeans IDE 8.0.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
Projects
ExpressionLanguage1
FilterApiDemo
JspActiontags
JspImplicitObjects
JstlDemo
Web Pages
META-INF
WEB-INF
index1.jsp
index2.jsp
index3.jsp
Source Packages
Libraries
jstl-1.2.jar
JDK 1.8 (Default)
Apache Tomcat 8.0.15.0
Configuration Files
SessionDemo1
SessionDemo2
index1.jsp
index2.jsp
index3.jsp
Source
History
15 </head>
16 <body>
17 <c:choose>
18 <c:when test="${103 == 101}">
19 Amit
20 </c:when>
21 <c:when test="${103 == 102}">
22 Deepak
23 </c:when>
24 <c:when test="${103 == 103}">
25 Deepesh
26 </c:when>
27 <c:when test="${103 == 104}">
28 Rahul
29 </c:when>
30 <c:when test="${103 == 105}">
31 Kamal
32 </c:when>
33 <c:otherwise>
34 Invalid rollno
</c:otherwise>
36 </c:choose>
37 </body>
38 </html>
39
html
body
Output
HTTP Server Monitor
35:27
INS
ENG


JstlDemo - NetBeans IDE 8.0.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
Projects
ExpressionLanguage1
FilterApiDemo
JspActiontags
JspImplicitObjects
JstlDemo
Web Pages
META-INF
WEB-INF
index1.jsp
index2.jsp
index3.jsp
index4.jsp
Source Packages
Libraries
jstl-1.2.jar
JDK 1.8 (Default)
Apache Tomcat 8.0.15.0
Configuration Files
SessionDemo1
SessionDemo2
index1.jsp
index2.jsp
index3.jsp
index4.jsp
Source
History
14 <title>JSP Page</title>
15 </head>
16 <body>
17 <c:set var="rollNumbers" value="${['101','102','103','104','105']}" />
18 <c:set var="names" value="${['Amit','Deepak','Deepesh','Rahul','Kamal']}" />
19
20 <c:set var="rollNo" value="103" />
21
22 <c:choose>
23 <c:when test="${rollNo == rollNumbers[0]}">
24 ${names[0]}
25 </c:when>
26 <c:when test="${rollNo == rollNumbers[1]}">
27 ${names[1]}
28 </c:when>
29 <c:when test="${rollNo == rollNumbers[2]}">
30 ${names[2]}
31 </c:when>
32 <c:when test="${rollNo == rollNumbers[3]}">
33 ${names[3]}
34 </c:when>
35 <c:when test="${rollNo == rollNumbers[4]}">
36 ${names[4]}
37 </c:when>
38 <c:otherwise>
html
body
Output
HTTP Server Monitor
36:25/1:1
INS
ENG


JstlDemo - NetBeans IDE 8.0.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
Projects
ExpressionLanguage1
FilterApiDemo
JspActiontags
JspImplicitObjects
JstlDemo
Web Pages
META-INF
WEB-INF
index1.jsp
index2.jsp
index3.jsp
index4.jsp
index5.jsp
Source Packages
Libraries
jstl-1.2.jar
JDK 1.8 (Default)
Apache Tomcat 8.0.15.0
Configuration Files
SessionDemo1
SessionDemo2
index1.jsp
index2.jsp
index3.jsp
index4.jsp
index5.jsp
Source
History
7 <%@page contentType="text/html" pageEncoding="UTF-8"%>
8 <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
9
10 <!DOCTYPE html>
11 <html>
12 <head>
13 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
14 <title>JSP Page</title>
15 </head>
16 <body>
17 <c:catch var="myException">
18 <%
19 int res = 100/2;
20 out.println(res);
21 %>
22 </c:catch>
23
24 <c:if test="${myException != null}">
25 Exception : ${myException}
26 Exception Message : ${myException.message}
27 </c:if>
28 </body>
29 </html>
30
html
body
Output
HTTP Server Monitor
JstlDemo deployed.
19:32
INS
ENG

```
<body>
    <c:forEach begin="1" end="10" var="i">
        ${i} <br/>
    </c:forEach>
</body>
```

```
<c:set var="names" value="${['Amit','Deepak','Deepesh','Rahul','Kamal']}" />

${names}
<c:forEach begin="0" end="4" var="i">
    ${names[i]}
</c:forEach>
```

```
<c:forEach items="${names}" var="name1">
    ${name1} <br/>
</c:forEach>
```

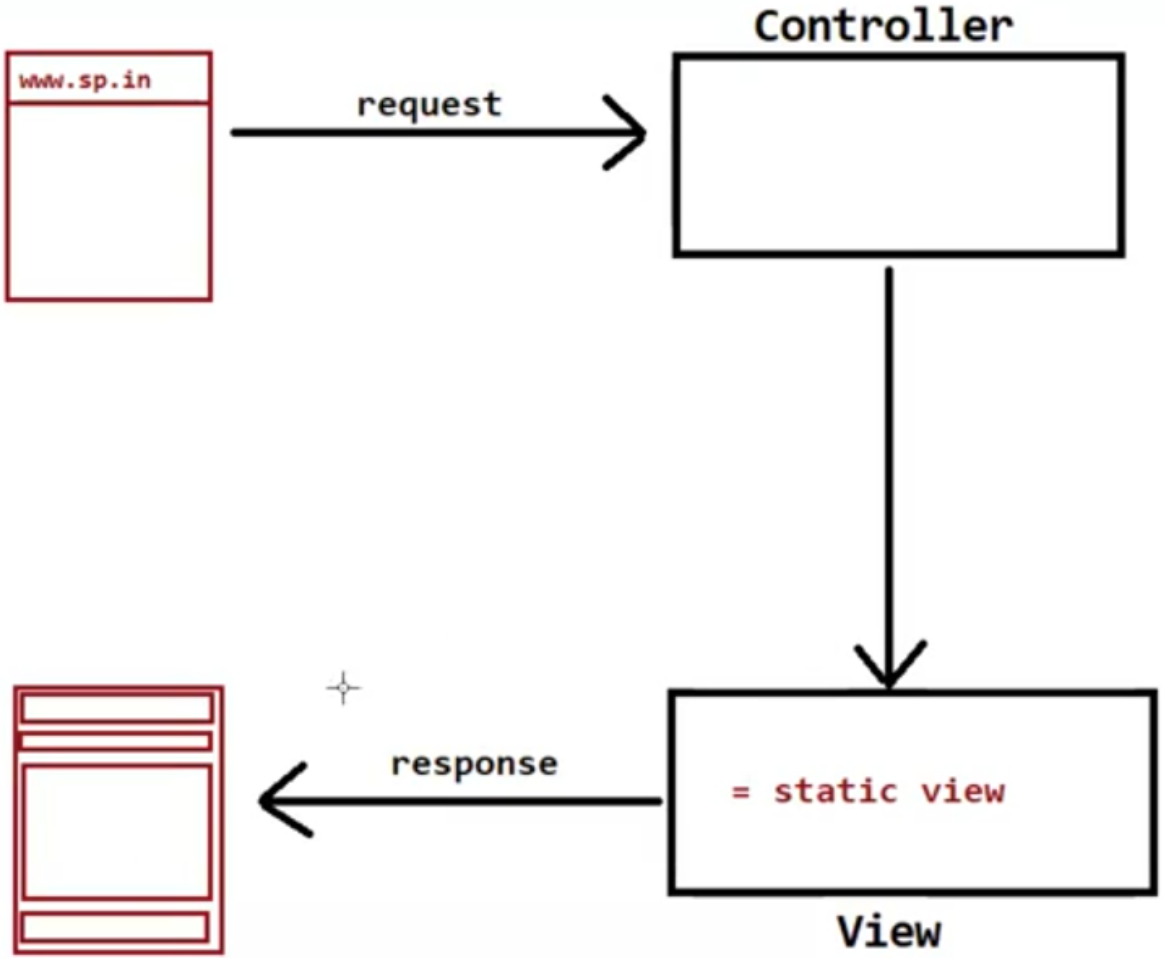
```
<body>
    <c:set var="names" value="Amit,Deepak,Deepesh,Kamal,Rahul" />

    <c:forTokens items="${names}" delims="," var="myname">
        ${myname}
    </c:forTokens>
</body>
```

=> XML tags :-
    -> These tags are used to work with XML data within JSP page
    -> To use XML tags in JSP we have to include taglib directive tag as below :-
        <%@ taglib uri="http://java.sun.com/jsp/jstl/xml" prefix="x" %>

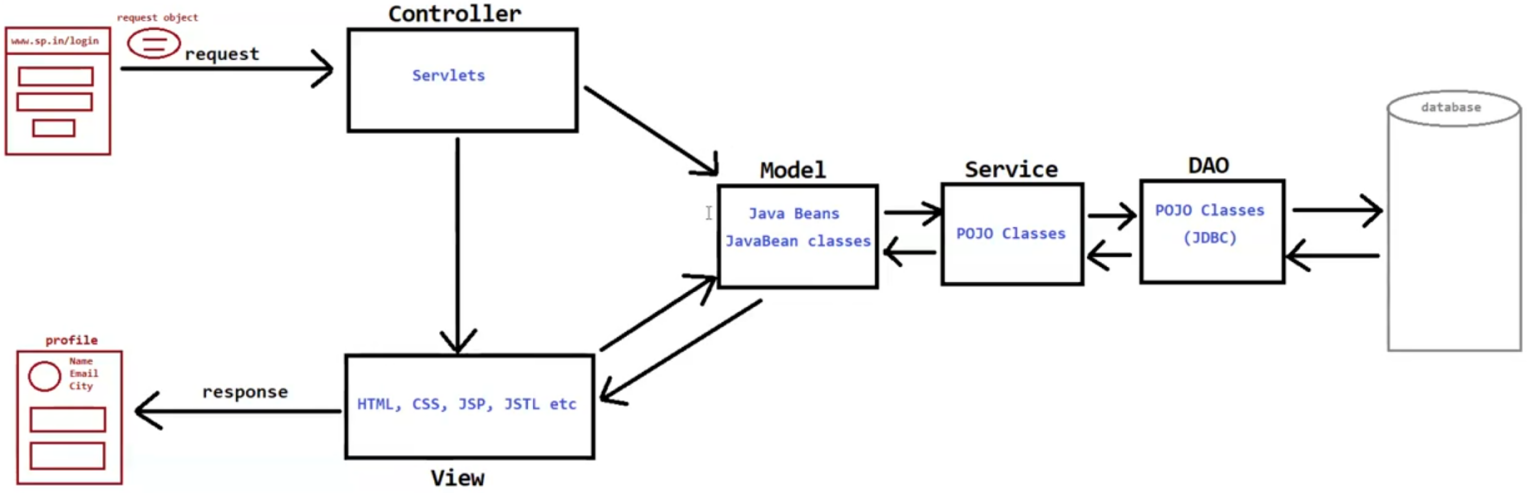
=> Formatted Tags :-
    -> These tags are used for internationalization and localization including formatting dates, numbers and currencies
    -> To use Formatted tags in JSP we have to include taglib directive tag as below :-
        <%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>

=> SQL Tags :-
    -> These tags are used for database operations but it is recommended to use other technologies like JDBC for
    database access due to security concerns
    -> To use SQL tags in JSP we have to include taglib directive tag as below :-
        <%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>

=> Function Tags :-
    -> These tags are used to call functions for various tasks in JSTL
    -> To use Function tags in JSP we have to include taglib directive tag as below :-
        <%@ taglib uri="http://java.sun.com/jsp/jstl/function" prefix="fn" %>


Controller
www.sp.in
request
response
= static view
View

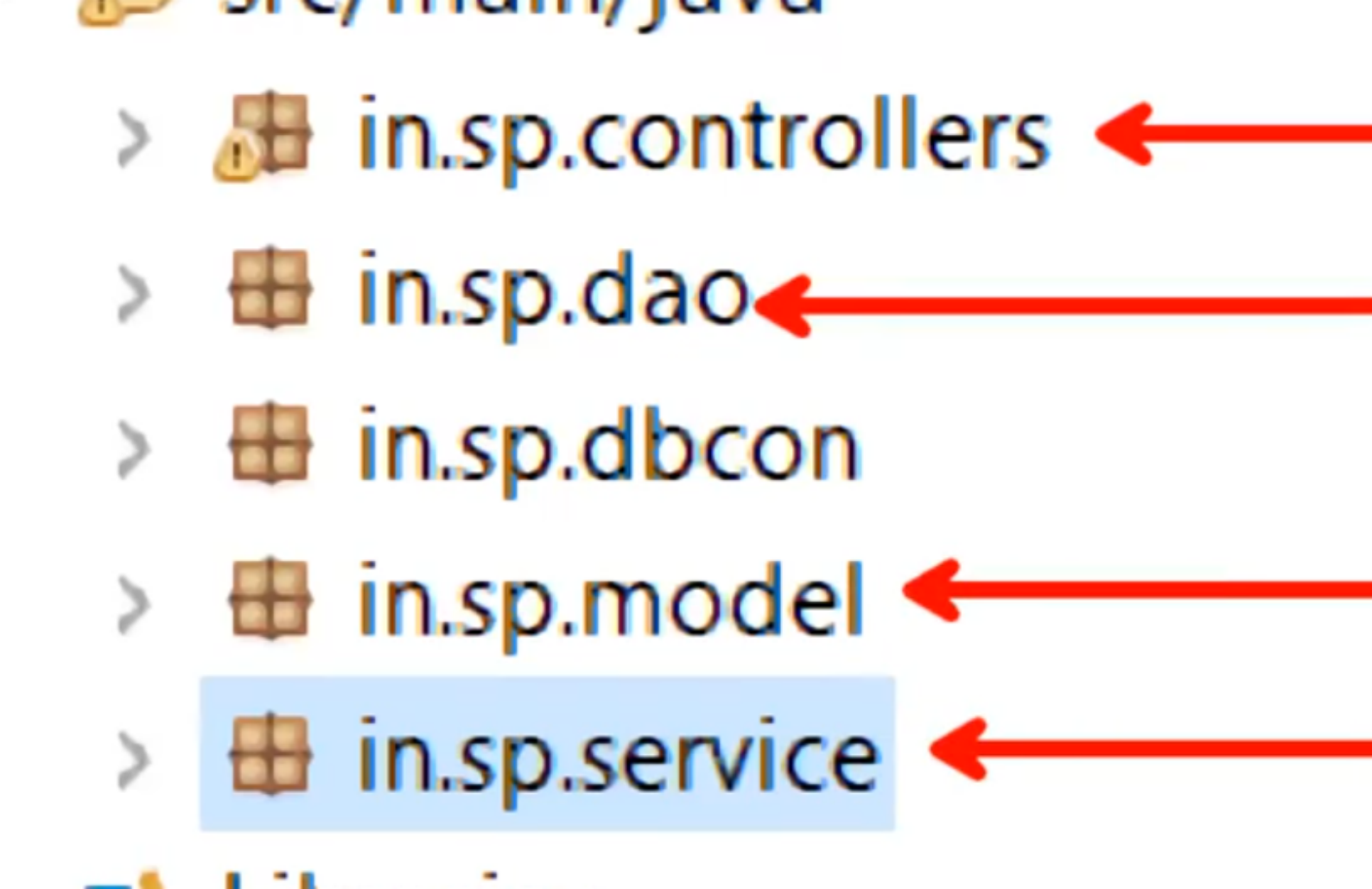
request object
www.sp.in/login
request
Controller
Model
profile
Name
Email
City
response
= dynamic view
View


www.sp.in/login
request object
request
Controller
Servlets
Model
Java Beans
JavaBean classes
Service
POJO Classes
DAO
POJO Classes
(JDBC)
database
profile
Name
Email
City
response
HTML, CSS, JSP, JSTL etc
View

```
out.println("<h3 style='color=red'> Email id and password didnt matched </h3>");
```

```java
public class DbConnection
{
    public static Connection getConnection()
    {
        Connection con = null;
        try
        {
            Class.forName("com.mysql.cj.jdbc.Driver");
            con = DriverManager.getConnection("jdbc:mysql://localhost:3306/mvc_db", "root", "rc
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }

        return con;
    }
}
```

```
=> MVC (Model View Controller) :-
   -> MVC is a design pattern which is used to separate the different layers of an application i.e. we are able to
   separate the model layer, UI/presentation layer and controller layer

   -> Architecture of MVC (for simple applications) :-

   -> MVC have 3 components :-
      1. M (Model) :-
         = Model is used to manages the application's data (handles the data storage, retrival and manipulation) and
         business logic
         = "POJO (Plain Old Java Object) Classes" acts as model
      2. V (View) :-
         = View represent the UI/Presentation layer which is transfered to the client as a response
         = There are 2 types of views:-
            - static view (HTML, CSS etc)
            - dynamic view (JSP, JSTL etc)
      3. C (Controller) :-
         = Controller is used to handle the client requests and process user inputs and events
         = Controller acts as an intermediary between the model and the view
         = "Servlets" acts as controller

   -> Architecture of MVC (for complex applications) :-
```


in.sp.controllers
in.sp.dao
in.sp.dbcon
in.sp.model
in.sp.service