AKTU

B.Tech 5th Sem

GATEWAY CLASSES

CS IT & CS Allied

Web Technology

ONE SHOT Revision

(Crash Course)

Unit-5

Servlets & Java Server Pages (JSP)

By Amol Sharma sir
- Pursuing Ph.D. from IIT (BHU)
- Wipro Certified Faculty (WCF)

AKTU 5th Sem: Web Technology

UNIT – V:

Servlets: Servlet Overview and Architecture, Interface Servlet and the Servlet Life Cycle, Handling HTTP get Requests, Handling HTTP post Requests, Redirecting Requests to Other Resources, Session Tracking, Cookies, Session Tracking with Http Session.

Java Server Pages (JSP): Introduction, Java Server Pages Overview, A First Java Server Page Example, Implicit Objects, Scripting, Standard Actions, Directives, Custom Tag Libraries.

AKTU PYQs

Web Technology (AKTU 2022 – 23)

1. Explain the implicit and explicit objects in JSP. Describe request, response and session objects in detail.
2. Describe Java Servlets. Write Index.html to input the some details (name, age, message) from user. Also write a servlet to fetch and display the same data upon the submission of request.
3. Compare Servlet and JSP. Justify why Servlets perform faster in comparison to JSP. Explain the life cycle of JSP with suitable diagram.

Web Technology (AKTU 2021 – 22)

1. Outline Implicit Objects in JSP.
2. Explain the Directory structure of web application.
3. Define the advantages of servlet over CGI? Explain with an example.
4. Illustrate why the session is required in web applications. Mention all the session tracking techniques in servlet with their pros and cons. Also give example of each.
5. Describe Servlet. Explain types of Servlet. Give an example of HttpServlet. Write steps to execute the servlet.

Web Technology (AKTU 2020 – 21)

1. Explain various JSP implicit objects in detail.
2. Describe Session Tracking. How Session Tracking using Http Session is performed?

AKTU PYQs

Web Technology (AKTU 2019 – 20)

1. Discuss about tomcat server. How to set the classpath for servlet in tomcat server?
2. Explain Servlets with its life cyle. How its lifecycle is different from the life cycle of JSP? Explain with an example.
3. Discuss JSP in details. What are JSP directives? Explain various types of directives with suitable example.

Web Technology (AKTU 2018 – 19)

1. Compare JSP and Servlet. Explain the lifecycle of a JSP page with a suitable diagram. Also list any five action tags used in JSP.
2. What is difference between Session and Cookies? Write a servlet program for servlet login and logout using cookies.

AKTU

B.Tech 5th Sem

GATEWAY CLASSES

CS IT & CS Allied

Web Technology

UNIT-5 Lecture-1

Today's Target

- Servlets
  - Servlet Overview and Architecture,
  - Interface Servlet
  - Servlet Life Cycle,
  - Handling HTTP get Requests,
  - Handling HTTP post Requests.
- AKTU PYQs

By Amol Sharma sir
- Pursuing Ph.D. from IIT (BHU)
- Wipro Certified Faculty (WCF)

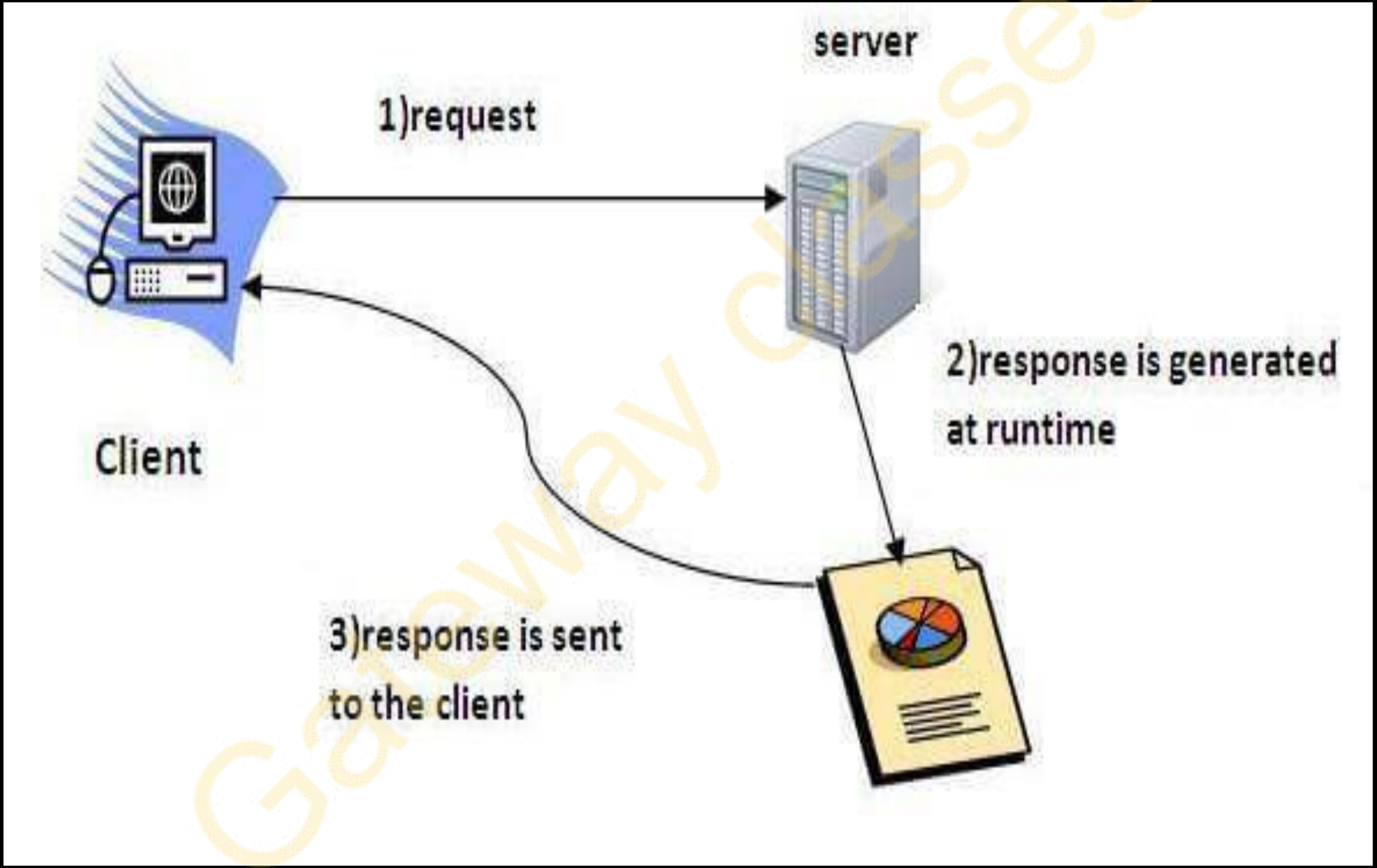
Servlets

"A servlet is a java program that executes on the server side of a web connection."

Servlet Overview

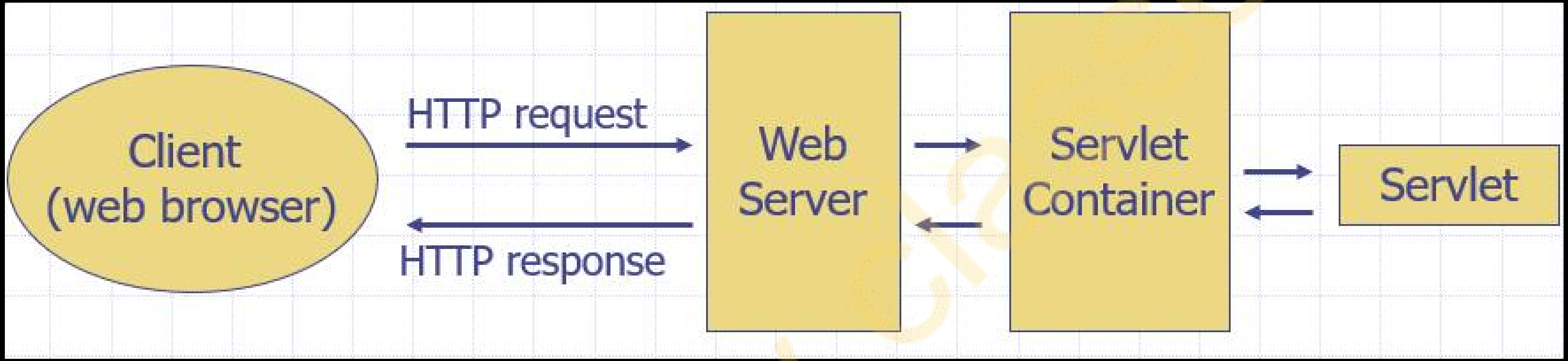
- Servlet is a class that extends the capabilities of the servers and responds to the incoming requests.
- A servlet can collect input from users through web page forms, present records from a database/ other sources, and create web pages dynamically.
- Executes within the address space of web server.
- Platform Independent.
- Java Security Manager on the server enforces a set of restrictions to protect the resources on a server machine.
- Full functionality of the Java class libraries is available to a servlet.
- Server Development Environment is needed for the servlet creation. One recommended is *Apache Tomcat Server.*

Servlets


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

Servlets

Servlet Architecture



- ➢ The client makes a request via HTTP.
- ➢ The web server receives the request and forwards it to the servlet container for processing.
- ➢ The servlet container checks if the servlet is already loaded into the JVM. If the servlet is not yet loaded, the servlet container loads it into the JVM and initializes it.
- ➢ The servlet container invokes the servlet, passing the HTTP request to it.
- ➢ The servlet processes the request, generates a response, and sends it back to the servlet container.
- ➢ The servlet container returns the response to the web server.
- ➢ The web server forwards the response to the client.

Servlets

Servlet Life Cycle

Loading:

When the servlet is accessed for the first time, the servlet container loads the servlet class into memory. This step ensures the servlet is ready for use.

Initialization:

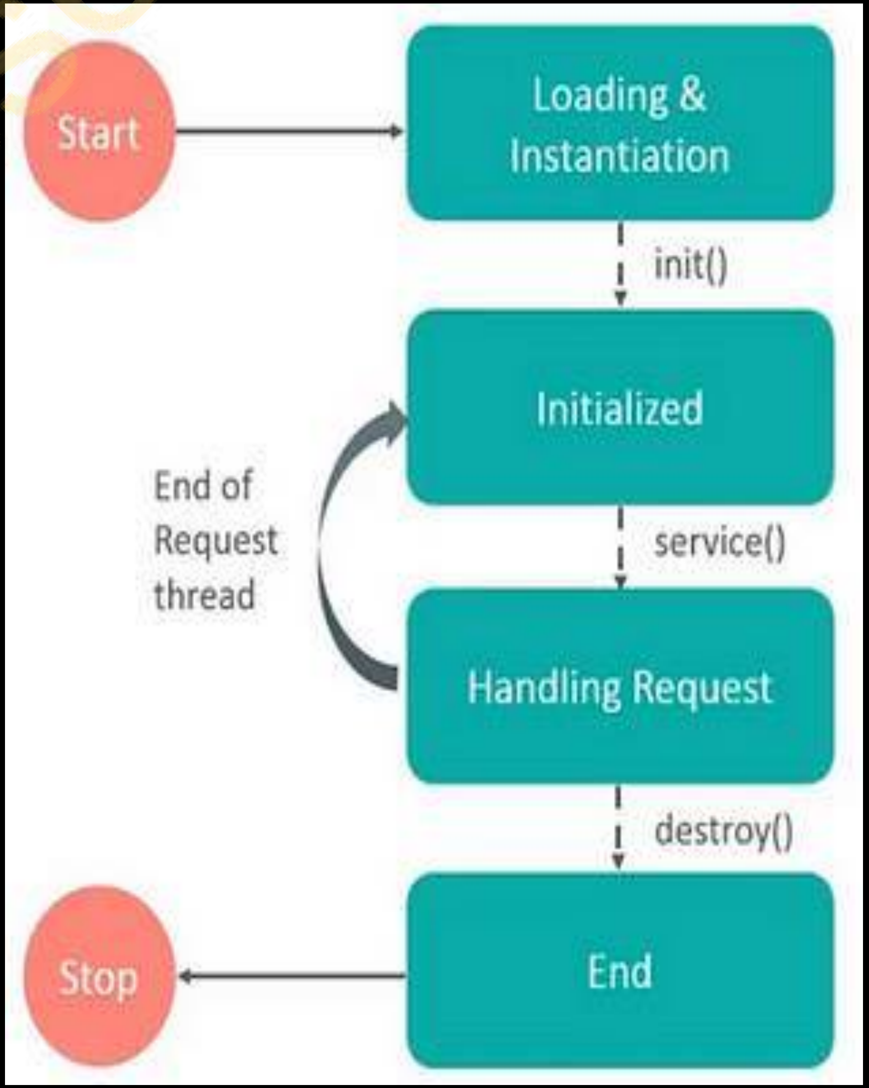
After loading, the servlet container creates an instance of the servlet and calls its init() method. This method is executed only once during the servlet's lifecycle and is used for initialization tasks, such as establishing a database connection or reading configuration settings.

Request Processing:

Each time the servlet receives a client request, the servlet container invokes the service() method. This method processes the request and generates an appropriate response for the client.

Destruction:

When the servlet is removed from the container, the servlet container calls the destroy() method. This method performs cleanup tasks, such as closing database connections or releasing system resources.



Servlets

Servlet API

The javax.servlet package contains a number of interfaces and classes that are used in developing servlets.

‘Servlet’ interface

It’s the foundation of the Servlet API. Every servlet must implement this interface either directly or indirectly.

This interface has following methods to manage the servlet lifecycle:

- void init(ServletConfig config)
- void service (ServletRequest request, ServletResponse response)
- void destroy()

‘GenericServlet’ class

GenericServlet is an abstract class that simplifies servlet development by handling non-HTTP-specific tasks.

- It provides default implementations for lifecycle methods.
- Developers only need to override the service() method to process requests.

Servlets

Example: (Servlet Life Cycle)

```java
import jakarta.servlet.*; // or import javax.servlet.* for tomcat 9 or earlier
import jakarta.servlet.annotation.WebServlet; // or import javax.servlet.annotation.WebServlet for tomcat 9 or earlier
import java.io.*;

@WebServlet("/servletlifecycle")
public class ServletLifeCycle extends GenericServlet {
    // Called only once when the servlet is initialized
    @Override
    public void init() throws ServletException {
        System.out.println("Servlet is being initialized...");
    }
    // Called for every request to the servlet
    @Override
    public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException {
        System.out.println("Servlet is servicing a request...");
        // Send a simple response back to the client
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html><body>");
        out.println("<p>The <strong>service()</strong> method was invoked to handle this request.</p>");
        out.println("</body></html>");
        out.close();
    }
    // Called only once when the servlet is being removed
    @Override
    public void destroy() {
        System.out.println("Servlet is being destroyed...");
    }
}
```

Servlets

Servlet API

When working with HTTP, we use the interfaces and classes in javax.servlet.http/ jakarta.servlet.http.

'HttpServlet' class

HttpServlet is a subclass of GenericServlet designed specifically for handling HTTP requests.
It simplifies development by providing HTTP-specific methods:

- doGet() - Handles HTTP GET requests.
- doPost() - Handles HTTP POST requests.

'HttpServletRequest' interface Request object passed to doGet and doPost. Extends ServletRequest.

Frequently used methods

String getParameter (String name)	Returns value of parameter name.
Enumeration getParameterNames ()	Returns names of parameters.
String[] getParameterValues (String name)	Returns array of strings containing values of a parameter.

'HttpServletResponse' interface Response object passed to doGet and doPost. Extends ServletResponse.

Frequently used methods

PrintWriter getWriter()	Gets character-based output stream, send text to client.
void setContentType(String type)	Specify MIME (Multipurpose Internet Mail Extensions) type of the response. MIME type "text/html" indicates that response is HTML document. Helps to display data properly.

Servlets

Handling HTTP GET Requests

- GET Request is used to request data from the server.
- Parameters are appended to the URL as a query string (?key=value).
- It is used for actions like searching or retrieving resources.

Method to Handle GET Requests:

The doGet() method in the HttpServlet class is overridden to handle GET requests.

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

HttpServletRequest object is used to retrieve client-provided data (e.g., query parameters).

HttpServletResponse object is used to send responses (e.g., HTML content) back to the client.

Servlets

Handling HTTP POST Requests

- POST request is used to send data to the server, typically through form submissions.
- Data is sent in the body of the HTTP request, not appended to the URL.
- Commonly used for secure or sensitive data, like login credentials.

Method to Handle GET Requests:

The doPost() method in the HttpServlet class is overridden to handle POST requests.

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

HttpServletRequest object is used to retrieve form data or uploaded files.

HttpServletResponse object is used to send responses (e.g., acknowledgment or error messages).

Servlets

Example: (Handling HTTP GET Requests)

```
import java.io.*;
import jakarta.servlet.*;
import jakarta.servlet.http.*;
import jakarta.servlet.annotation.WebServlet;

@WebServlet("/getmessage")
public class GetMessageServlet extends HttpServlet {

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

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

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

        out.println("<html>");
        out.println("<body>");
        out.println("<h1>Details sent by the user:</h1>");
        out.println("<p><b>Name: </b>" + name + "</p>");
        out.println("<p><b>Age: </b>" + age + "</p>");
        out.println("<p><b>Message: </b>" + message + "</p>");
        out.println("</body>");
        out.println("</html>");

    }
}
```

Servlets

Example: (Handling HTTP POST Requests)

```
import java.io.*;
import jakarta.servlet.*;
import jakarta.servlet.http.*;
import jakarta.servlet.annotation.WebServlet;

@WebServlet("/postmessage")
public class PostMessageServlet extends HttpServlet {

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

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

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

        out.println("<html>");
        out.println("<body>");
        out.println("<h1>Details sent by the user:</h1>");
        out.println("<p><b>Name: </b>" + name + "</p>");
        out.println("<p><b>Age: </b>" + age + "</p>");
        out.println("<p><b>Message: </b>" + message + "</p>");
        out.println("</body>");
        out.println("</html>");

    }
}
```

Servlets

Executing a Servlet

Steps:

To run the servlet from the command line, follow these steps in Windows.

[*Befor following the steps, ensure you have the Java Development Kit (JDK) installed, Apache Tomcat properly configured, and the servlet-api.jar located in C:\apache-tomcat-11.0.2\lib.*]

1. Save Your Servlet

   Save your servlet program in a .java file. For our examples, name it GetMessageServlet.java/ PostMessageServlet.java

2. Set Environment Variables

   Ensure the following environment variables are set:

JAVA_HOME:	Path to your JDK installation (e.g., C:\Program Files\Java\jdk-22).
PATH:	Path to JAVA_HOME\bin (e.g., C:\Program Files\Java\jdk-22\bin).

3. Compile the Servlet

   Open Command Prompt and navigate to the directory containing Java source code file say GetMessageServlet.java. Use the following command to compile the servlet, ensuring you include the servlet-api.jar in the classpath:

   ```
   javac -cp C:\apache-tomcat-11.0.2\lib\servlet-api.jar GetMessageServlet.java
   ```

   This will generate a GetMessageServlet.class file.

Servlets

Executing a Servlet

Steps:

4. Create the Directory Structure

For Tomcat to recognize the servlet, you need to create the appropriate directory structure. Navigate to the webapps folder of your Tomcat installation (e.g., C:\apache-tomcat-11.0.2\webapps) and do the following:

i. Create a new directory for your application, e.g., MyServlets.
ii. Inside MyServlets, create the following:
   - A folder named WEB-INF.
   - Inside WEB-INF, create:
     - A classes folder to store the .class file.
     - A web.xml file for servlet mapping.
       (If annotation @WebServlet("/getmessage") is not used)

5. Move the Class File

Copy the GetMessageServlet.class file to C:\apache-tomcat-11.0.2\webapps\MyServlets\WEB-INF\classes.

Servlets

Executing a Servlet

Steps:

6. Create the web.xml File (If annotation @WebServlet("/getmessage") is not used)

In C:\apache-tomcat-11.0.2\webapps\MyServlets\WEB-INF, create a web.xml file with the following content:

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
version="3.0">
  <servlet>
    <servlet-name>GetMessageServlet</servlet-name>
    <servlet-class> GetMessageServlet </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name> GetMessageServlet </servlet-name>
    <url-pattern>/getmessage</url-pattern>
  </servlet-mapping>
</web-app>
```

The *<servlet>* element declares the servlet, including a name used to refer to the servlet by other elements in the file, the class to use for the servlet, and initialization parameters.

The *<servlet-mapping>* element specifies a URL pattern and the name of a declared servlet to use for requests whose URL matches the pattern.

The file named web.xml is called deployment descriptor

- A web application's deployment descriptor describes the classes, resources and configuration of the application and how the web server uses them to serve web requests.
- When the web server receives a request for the application, it uses the deployment descriptor to map the URL of the request to the code that ought to handle the request.

Servlets

Executing a Servlet

Steps:

Instead of defining servlet mapping in the web.xml file, we can use the @WebServlet annotation in the servlet class to specify the URL pattern directly.

```
@WebServlet("/getmessage")
```

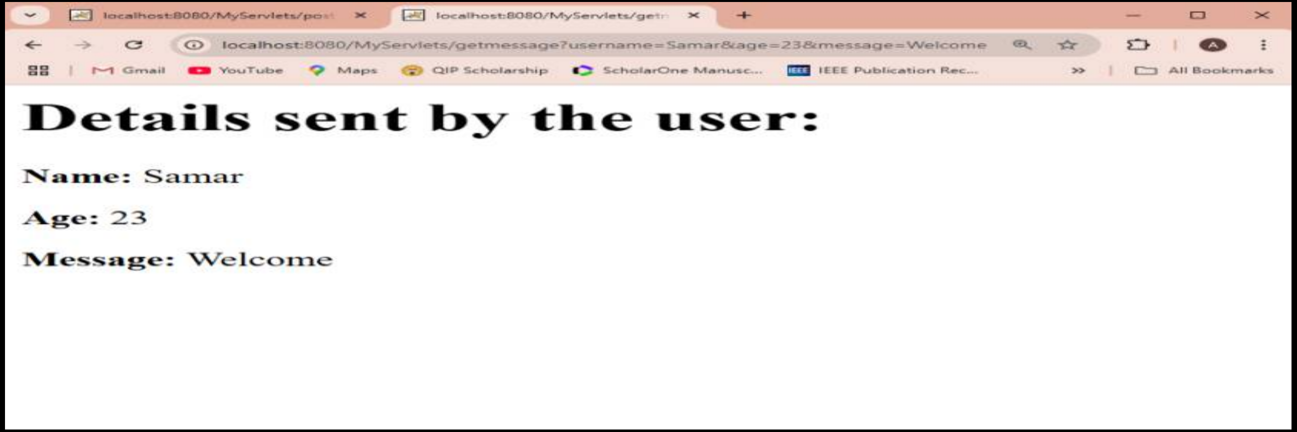
7. Start Tomcat

   Start the Tomcat server by running startup.bat located in the bin directory of your Tomcat installation: C:\apache-tomcat-11.0.2\bin\startup.bat

8. Access the Servlet

   Open a web browser and navigate to:
   http://localhost:8080/MyServlets/getmessage?username=Samar&age=23&message=Welcome

   The servlet will display the details:


localhost:8080/MyServlets/post
localhost:8080/MyServlets/getm
localhost:8080/MyServlets/getmessage?username=Samar&age=23&message=Welcome
Gmail
YouTube
Maps
QIP Scholarship
ScholarOne Manusc...
IEEE Publication Rec...
All Bookmarks

Details sent by the user:

Name: Samar

Age: 23

Message: Welcome

Servlets

Executing a Servlet

Steps:

If you have used the doPost method (Follow steps 1- 7 for the PostMessageServlet.java program), you need to submit the request using an HTML form.

9. Create an HTML Form

   To test the servlet, create an index.html file in the C:\apache-tomcat-11.0.2\webapps\MyServlets directory with the following content:

```
<html>
    <head> <title>AKTU PYQ 2022 - 23 Q.2</title> </head>
    <body> <h1>AKTUP PYQ 2022 - 23 Q.2</h1>
        <p>
            Write Index.html to input the some details (name, age, message) from user.
            Also write a servlet to fetch and display the same data upon the submission of request.
        </p>
        <form name="message" method="post" action="postmessage">
            <input type="text" name="username" placeholder="Enter your name here..."><br><br>
            <input type="text" name="age" placeholder="Enter your age here..."><br><br>
            <input type="text" name="message" placeholder="Enter your message here..."><br><br>
            <input type="submit" name="submit">
        </form>
    </body>
</html>
```

Servlets

Executing a Servlet

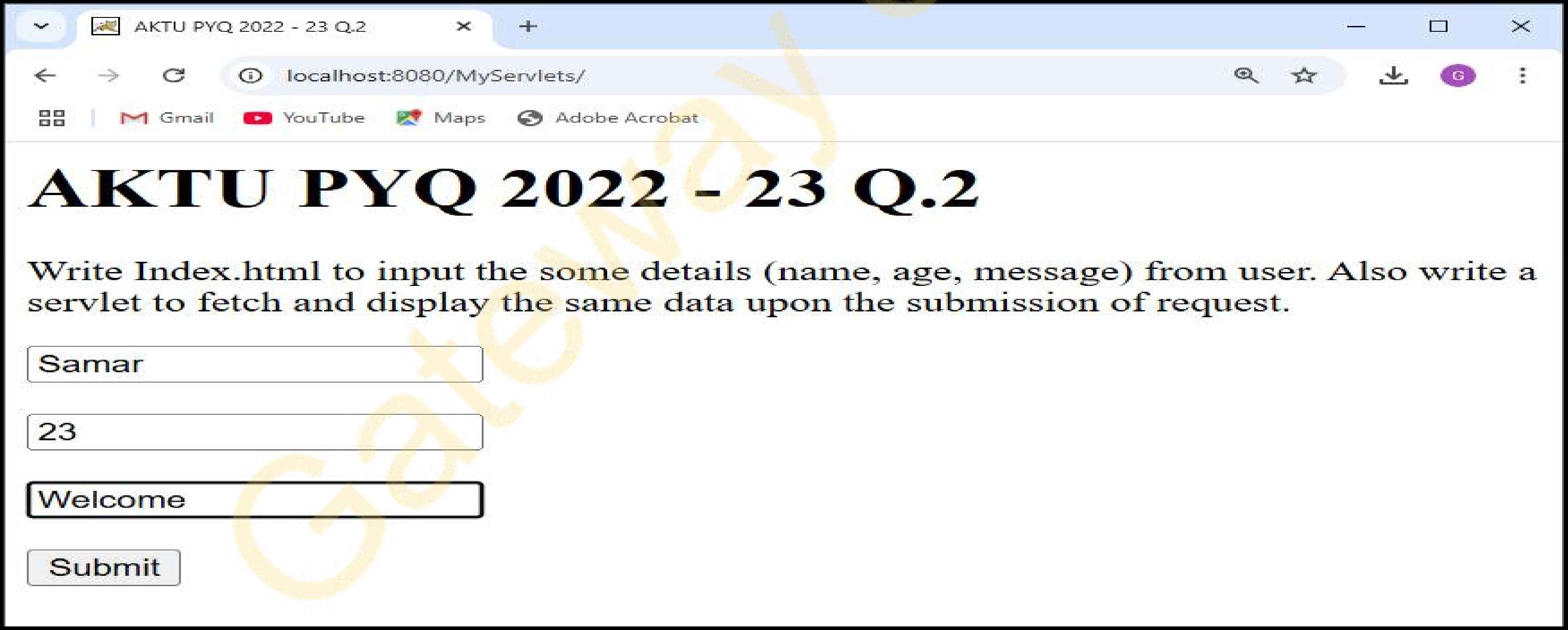
Steps:

Access the form at:

http://localhost:8080/MyServlets/index.html

or

simply, http://localhost:8080/MyServlets/



AKTU PYQ 2022 - 23 Q.2

Write Index.html to input the some details (name, age, message) from user. Also write a servlet to fetch and display the same data upon the submission of request.

Samar

23

Welcome

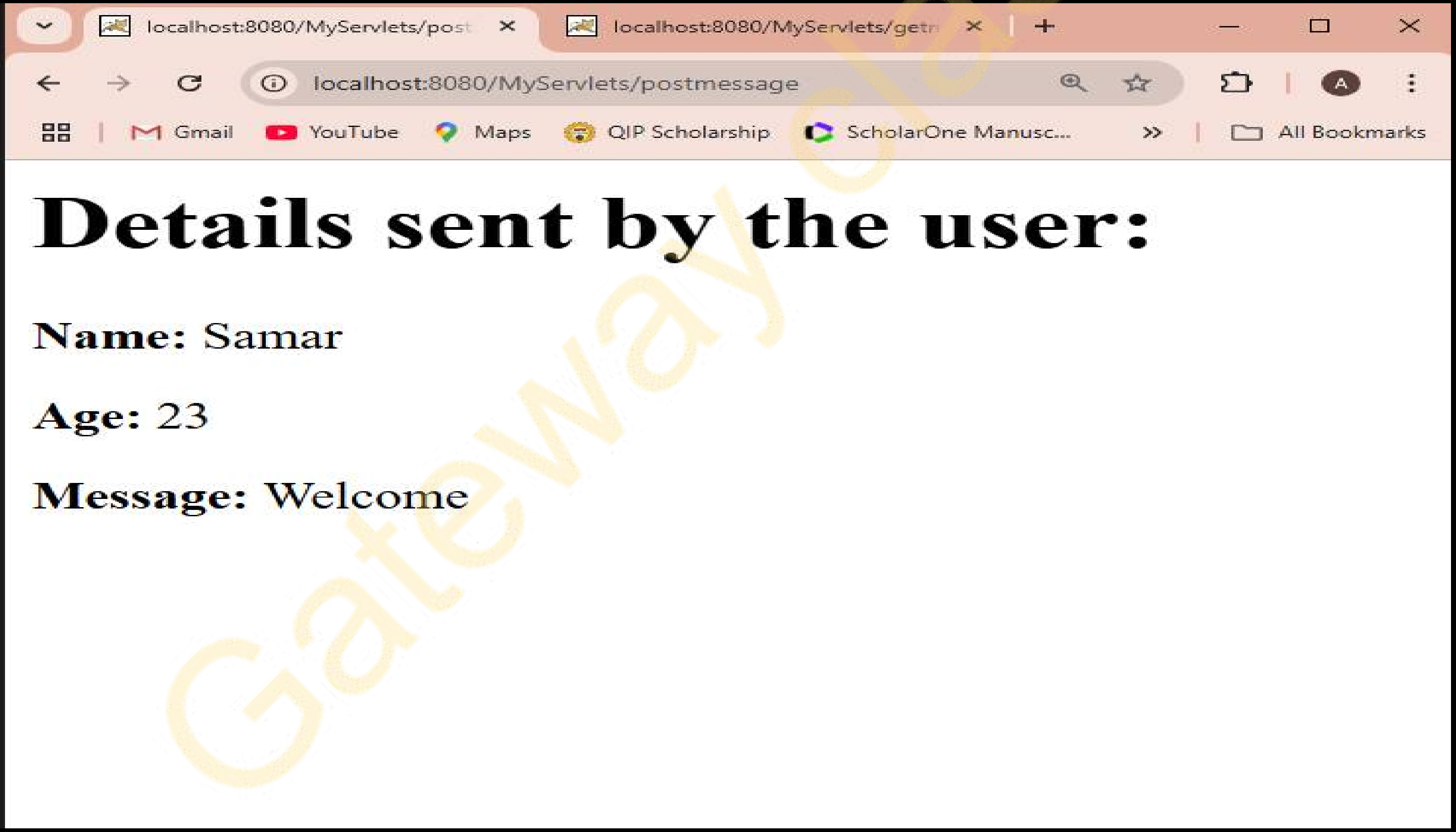
Submit

Servlets

Executing a Servlet

Steps:

Submit the form, and the servlet will display the details:


localhost:8080/MyServlets/post
localhost:8080/MyServlets/getm
localhost:8080/MyServlets/postmessage
Gmail
YouTube
Maps
QIP Scholarship
ScholarOne Manusc...
All Bookmarks

Details sent by the user:

Name: Samar

Age: 23

Message: Welcome

AKTU

B.Tech 5th Sem

GATEWAY CLASSES

CS IT & CS Allied

Web Technology

UNIT-5 Lecture-2

Today's Target

- Servlets
  - Redirecting Requests to Other Resources,
    - Client-side Redirection
      - sendRedirect()
    - Server-side Redirection
      - RequestDispatcher
        - forward()
        - include()
- AKTU PYQs

By Amol Sharma sir

- Pursuing Ph.D. from IIT (BHU)
- Wipro Certified Faculty (WCF)

Servlets

Redirecting Requests to Other Resources

"Redirecting requests in servlets allows the server to forward a request to another resource or instruct the client's browser to make a new request to a different URL."

For this purpose we may either use server-side forwarding with RequestDispatcher or client-side redirection with sendRedirect().

- Client-side Redirection

  Using the sendRedirect() method of HttpServletResponse.

  void sendRedirect(String location) throws IOException

  The client's browser is involved in the redirection. It receives a new URL and issues a new HTTP request.

- Server-side Redirection

  Using the RequestDispatcher interface.

  The request is forwarded to another resource within the server without involving the client's browser. The client does not know about the redirection.

Servlets

Redirecting Requests to Other Resources

RequestDispatcher interface provides the option of dispatching the client’s request to another web resource, which could be an HTML page, another servlet, JSP etc.
It provides the following two methods:

forward()

The forward() method is used to transfer the client request to another resource (HTML file, servlet, jsp etc). When this method is called, the control is transferred to the next resource called.

```
public void forward(ServletRequest request, ServletResponse response) throws ServletException, IOException
```

include()

The include() method is used to include the contents of the calling resource into the called one. When this method is called, the control still remains with the calling resource. It simply includes the processed output of the calling resource into the called one.

```
public void include(ServletRequest request, ServletResponse response)throws ServletException, java.io.IOException
```

Servlets

Redirecting Requests to Other Resources

Example:

RequestRedirectionDemo.html

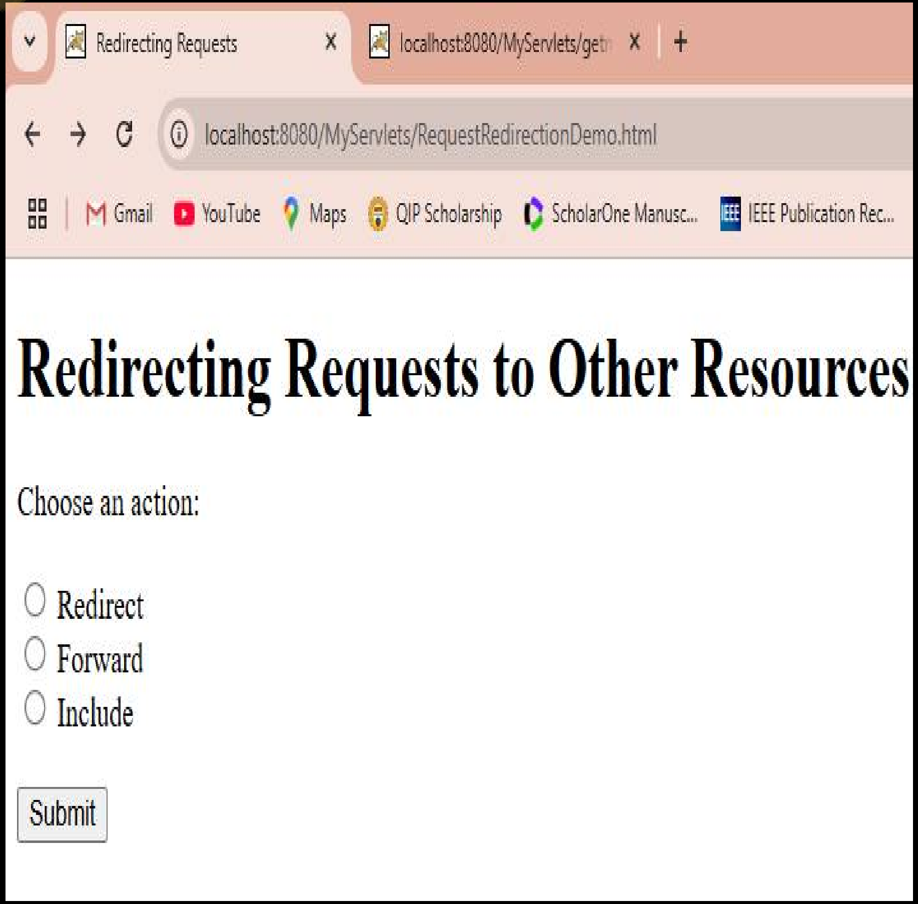
```
<!DOCTYPE html>
<html>
<head>
    <title>Redirecting Requests</title>
</head>
<body>
    <h1>Redirecting Requests to Other Resources</h1>
    <form action="RequestRedirectiontDemo" method="POST">
        <label for="action">Choose an action:</label><br><br>

        <input type="radio" id="redirect" name="action" value="redirect">
        <label for="redirect">Redirect</label><br>

        <input type="radio" id="forward" name="action" value="forward">
        <label for="forward">Forward</label><br>

        <input type="radio" id="include" name="action" value="include">
        <label for="include">Include</label><br><br>

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



Redirecting Requests to Other Resources

Choose an action:

Redirect
Forward
Include

Submit

Servlets

Redirecting Requests to Other Resources

Example:

RequestRedirectionDemo.java

```
import jakarta.servlet.*;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.*;
import java.io.*;

@WebServlet({ "/RequestRedirectiontDemo" })
public class RequestRedirectionDemo extends HttpServlet {

    public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        String action = request.getParameter("action");
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html><body>");
        if (action != null) {

            switch (action) {
                case "redirect":
                    // Perform client-side redirect using sendRedirect()
                    response.sendRedirect("http://localhost:8080/MyServlets/");
                    break;
```

Servlets

Redirecting Requests to Other Resources

Example:

RequestRedirectionDemo.html

```
                    case "forward":
                        // Perform server-side forwarding using RequestDispatcher.forward()
                        out.println("<strong>The response from the 'forward' case !!</strong>");
                        RequestDispatcher forwardDispatcher = request.getRequestDispatcher("/forwardedPage.html");
                        forwardDispatcher.forward(request, response);
                        break;
                    case "include":
                        out.println("<strong>The response from the 'include' case !!</strong>");
                        // Perform server-side include using RequestDispatcher.include()
                        RequestDispatcher includeDispatcher = request.getRequestDispatcher("/includedPage.html");
                        includeDispatcher.include(request, response);
                        break;
                    default:
                        out.println("<h3>Invalid action selected</h3>");
                }
            } else {
                out.println("<h3>No action selected. Please go back and select an action.</h3>");
            }

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

Servlets

Redirecting Requests to Other Resources

Example:

forwardedPage.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Forwarded Page</title>
</head>
<body>
    <h1>Welcome to the Forwarded Page!</h1>
    <p>This page was forwarded by the servlet
using the
<strong>RequestDispatcher.forward()</strong>
method.</p>
</body>
</html>
```

includedPage.html

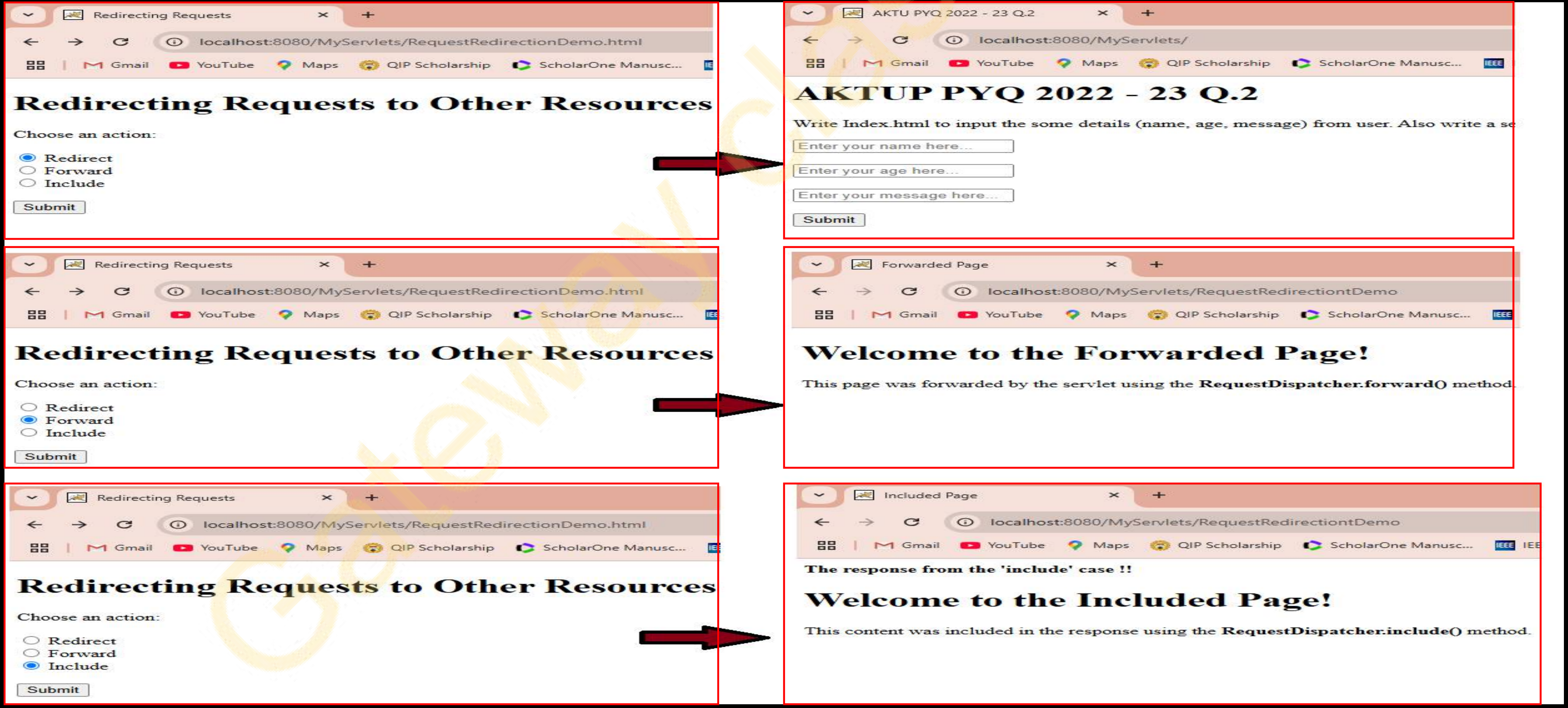
```
<!DOCTYPE html>
<html>
<head>
    <title>Included Page</title>
</head>
<body>
    <h1>Welcome to the Included Page!</h1>
    <p>This content was included in the response
using the
<strong>RequestDispatcher.include()</strong>
method.</p>
</body>
</html>
```

Servlets

Redirecting Requests to Other Resources

Example:

Output



Redirecting Requests

localhost:8080/MyServlets/RequestRedirectionDemo.html

Redirecting Requests to Other Resources

Choose an action:

[✓ Redirect]
Forward
Include

Submit

AKTU PYQ 2022 - 23 Q.2

localhost:8080/MyServlets/

AKTUP PYQ 2022 – 23 Q.2

Write Index.html to input the some details (name, age, message) from user. Also write a se

Enter your name here...

Enter your age here...

Enter your message here...

Submit

Redirecting Requests

localhost:8080/MyServlets/RequestRedirectionDemo.html

Redirecting Requests to Other Resources

Choose an action:

Redirect
[✓ Forward]
Include

Submit

Forwarded Page

localhost:8080/MyServlets/RequestRedirectiontDemo

Welcome to the Forwarded Page!

This page was forwarded by the servlet using the **RequestDispatcher.forward()** method.

Redirecting Requests

localhost:8080/MyServlets/RequestRedirectionDemo.html

Redirecting Requests to Other Resources

Choose an action:

Redirect
Forward
[✓ Include]

Submit

Included Page

localhost:8080/MyServlets/RequestRedirectiontDemo

The response from the 'include' case !!

Welcome to the Included Page!

This content was included in the response using the **RequestDispatcher.include()** method.

AKTU

B.Tech 5th Sem

GATEWAY CLASSES

CS IT & CS Allied

Web Technology

UNIT-5 Lecture-3

Today's Target

- Servlets
  - Session Tracking
    - Techniques
    - Cookies
    - HttpSession
- AKTU PYQs

By Amol Sharma sir

- Pursuing Ph.D. from IIT (BHU)
- Wipro Certified Faculty (WCF)

Servlets

Session Tracking

"A session is defined as a series of related browser requests that come from the same client during a certain time period."

"Session Tracking is a technique to maintain user-specific information across multiple requests on a web application."

- HTTP is a stateless protocol, meaning it does not retain information about a user between requests.
- Session tracking helps create a continuous interaction with the user.

Need / Use of Session Tracking

- To maintain user login state.
- To personalize user experiences, like showing recommendations.
- To store temporary user data during their interaction with the application.

Example: Tracking items in a shopping cart of a customer in an e-commerce application.

Servlets

Session Tracking Techniques

1. Cookies:

Cookies are small pieces of data stored on the client-side by the browser.

They are used for persistent data storage across sessions, allowing information to be retained even when the browser is closed and reopened.

Example: A cookie can be set using the Set-Cookie header, such as Set-Cookie: user=John; Max-Age=3600, which stores the user's name for 3600 seconds (1 hour).

Benefits (Pros):

1. Persistent across browser sessions (optional).
2. No server resources required for storage.

Limitations (Cons):

1. Dependent on client/browser cookie support.
2. Limited size (~4KB) and potential security risks.

Servlets

Session Tracking Techniques

2. URL Rewriting:

URL rewriting appends the session ID as a parameter to the URLs of web requests.

Example: a rewritten URL might look like this:

http://example.com/home?sessionId=12345,

where the session ID is passed as part of the query string.

This technique does not rely on browser cookie support and can be used as an alternative for session tracking when cookies are disabled.

Benefits (Pros):

1. Works even if cookies are disabled.
2. Simple to implement for static links.

Limitations (Cons):

1. Exposes session ID in URLs, posing security risks.
2. Requires appending IDs to all links.

Servlets

Session Tracking Techniques

3. Hidden Form Fields:

Hidden form fields are used to add a hidden <input> field within HTML forms to store session data.

Example: a hidden field can be defined as

```
<input type="hidden" name="sessionId" value="12345">,
```

where the session ID is stored as its value.

This technique requires form submission to propagate the session data from the client to the server.

Benefits (Pros):

1. Works without cookies.
2. Simple for form-driven workflows.

Limitations (Cons):

1. Requires form submission for propagation.
2. Limited to applications that rely on form inputs.

Servlets

Session Tracking Techniques

4. HttpSession:

A server-side session object is provided by servlets through the HttpSession interface.

The HttpSession object automatically manages session data and session IDs, allowing the server to track user sessions.

Example: We can create or retrieve the session object using

HttpSession session = request.getSession();

which returns the session associated with the current request.

Benefits (Pros):

1. Handles session management automatically.
2. Stores large, complex data securely on the server.

Limitations (Cons):

1. Consumes server memory and resources.
2. Session data is lost if the server restarts unless stored persistently.

Servlets

'Cookie' class

The Cookie class encapsulates a cookie, which is stored on the client and contains session information.

Example: In an online store, a cookie can store user information like name and address, so the user does not need to re-enter this data on subsequent visits.

A servlet can send a cookie to the user's machine using the addCookie() method of the HttpServletResponse interface:

void addCookie(Cookie cookie): Adds cookie to the HTTP response.

The cookie's data is included in the header of the HTTP response sent to the browser.

The key components of a cookie stored on the user's machine are:

- The name and value of the cookie.
- The expiration date of the cookie.
- The domain and path of the cookie.

- The expiration date determines when the cookie is deleted from the user's machine. If not set, the cookie is deleted when the browser session ends.
- The domain and path of the cookie define when it should be sent in the header of an HTTP request. The cookie is sent if the domain and path match the URL being accessed.

Servlets

‘Cookie’ class

Constructor:

The Cookie class has a constructor with the following signature:

Cookie(String name, String value)

The name and value of the cookie are passed as arguments to this constructor.

Methods:

getName():	Returns the name of the cookie.
setName(String name):	Sets the name of the cookie.
getValue():	Returns the value of the cookie.
setValue(String value):	Sets the value of the cookie.
getMaxAge():	Returns the maximum age of the cookie, in seconds.
setMaxAge(int expiry):	Sets the maximum age of the cookie, in seconds.

Servlets

'Cookie' class

Example:

(AKTU PYQ: Write a servlet program for servlet login and logout using cookies.*)*

```
<!DOCTYPE html>
<html>
<head>
    <title>Login</title>
</head>
<body>
    <h1>Login</h1>
    <form action="LoginServlet" method="post">
        <label for="username">Username:</label>
        <input type="text" name="username" id="username" required>
        <br><br>
        <label for="password">Password:</label>
        <input type="password" name="password" id="password" required>
        <br><br>
        <button type="submit">Login</button>
    </form>
</body>
</html>
```

login.html

Servlets

'Cookie' class

Example:

(AKTU PYQ: Write a servlet program for servlet login and logout using cookies.***)***

```java
import jakarta.servlet.*;
import jakarta.servlet.http.*;
import jakarta.servlet.annotation.WebServlet;
import java.io.*;
@WebServlet({ "/LoginServlet" })
public class LoginServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        // Get username from the login form
        String username = request.getParameter("username");
        String password = request.getParameter("password");
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        if ("admin".equals(username) && "password".equals(password)) {
            // Create a cookie for the logged-in user
            Cookie userCookie = new Cookie("user", username);
            userCookie.setMaxAge(60 * 60); // 1-hour (3600 seconds) expiration
            response.addCookie(userCookie);
            response.sendRedirect("WelcomeServlet");
        } else {
            out.println("<html><body>");
            out.println("<h1>Login Failed!</h1>");
            out.println("<a href='login.html'>Try Again</a>");
            out.println("</body></html>");
        }
    }
```

$\left.\right\}$ LoginServlet.java

Servlets

'Cookie' class

Example:

(AKTU PYQ: Write a servlet program for servlet login and logout using cookies.*)*

```
import jakarta.servlet.*;
import jakarta.servlet.http.*;
import jakarta.servlet.annotation.WebServlet;
import java.io.*;
@WebServlet({"/WelcomeServlet"})
public class WelcomeServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
                                                   throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        // Retrieve cookies
        Cookie[] cookies = request.getCookies();
        boolean loggedIn = false;
        String username = "";
        if (cookies != null) {
            for (Cookie cookie : cookies) {
                if (cookie.getName().equals("user")) {
                    loggedIn = true;
                    username = cookie.getValue();
                    break;
                }
            }
        }
```

WelcomeServlet.java

Servlets

'Cookie' class

Example:

(AKTU PYQ: Write a servlet program for servlet login and logout using cookies.*)*

```
        if (loggedIn) {
            // Display Welcome Page
            out.println("<html><body>");
            out.println("<h1>Welcome, " + username + "!</h1>");
            out.println("<form action='LogoutServlet' method='post'>");
            out.println("<input type='submit' value='Logout'>");
            out.println("</form>");
            out.println("</body></html>");
        } else {
            // Redirect to Login Page if no valid cookie is found
            out.println("<h1>You are logged out now, so can not access Welcome page directly,
                                                                please login first !!</h1>");
            request.getRequestDispatcher("login.html").include(request,response);
            //response.sendRedirect("login.html");
        }
    }
}
```

$\}$ WelcomeServlet.java

Servlets

'Cookie' class

Example:

(AKTU PYQ: Write a servlet program for servlet login and logout using cookies.*)*

```
import jakarta.servlet.*;
import jakarta.servlet.http.*;
import jakarta.servlet.annotation.WebServlet;
import java.io.*;

@WebServlet({"/LogoutServlet"})
public class LogoutServlet extends HttpServlet {

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

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

        // Invalidate the user cookie
        Cookie logoutCookie = new Cookie("user", "");
        logoutCookie.setMaxAge(0); // Expire the cookie immediately
        response.addCookie(logoutCookie);
        // Display Logout Message
        out.println("<html><body>");
        out.println("<h1>You have successfully logged out.</h1>");
        out.println("<a href='login.html'>Go to Login Page</a>");
        out.println("</body></html>");
    }
}
```

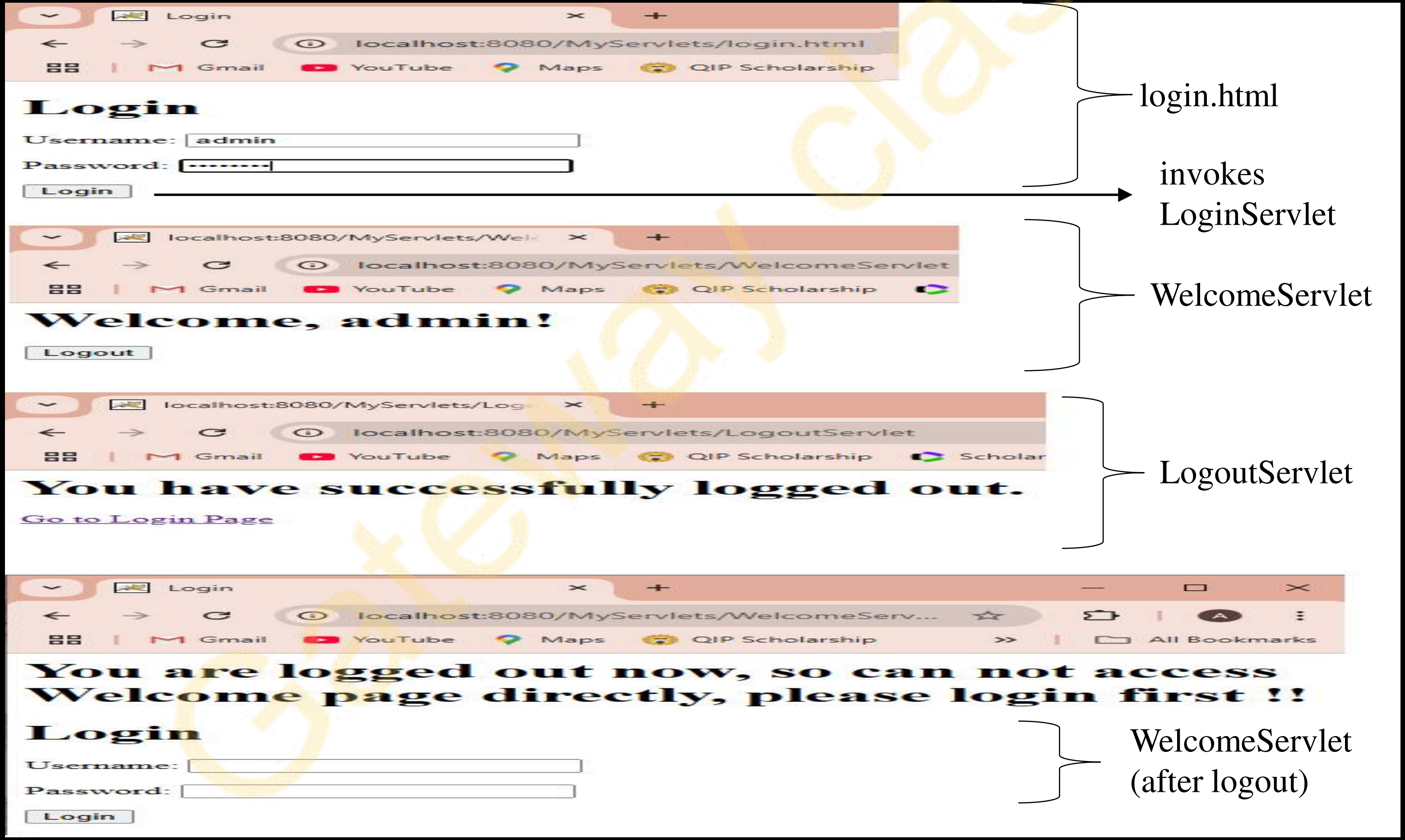
LogoutServlet.java

Servlets

'Cookie' class

Example:

(AKTU PYQ: Write a servlet program for servlet login and logout using cookies.*)*



localhost:8080/MySerlets/login.html

Login

Username: admin

Password: ·········

Login

localhost:8080/MyServlets/WelcomeServlet

Welcome, admin!

Logout

localhost:8080/MyServlets/LogoutServlet

You have successfully logged out.

Go to Login Page

localhost:8080/MyServlets/WelcomeServ...

You are logged out now, so can not access Welcome page directly, please login first !!

Login

Username:

Password:

Login

Servlets

Session Tracking with 'HttpSession'

The 'HttpSession' interface allows servlets to manage and maintain state information across multiple HTTP requests from the same user.

- Session state is shared across all servlets for a given client.
- Sessions are automatically managed and identified using a unique session ID.
- Sessions are stored on the server, reducing client-side dependency.

- A session can be created via the getSession( ) method of HttpServletRequest. An HttpSession object is returned.

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

If parameter is true and no session exists, creates and returns a session for this request. otherwise, returns the existing session for this request.

Servlets

Session Tracking with 'HttpSession'

Key Methods of HttpSession

void setAttribute(String name, Object value):
Stores data in the session with a name-value pair.

Object getAttribute(String name):
Retrieves the value associated with a name.

Enumeration<String> getAttributeNames():
Returns an enumeration of all attribute names.

void removeAttribute(String name):
Removes an attribute from the session.

void invalidate():
Ends the session and removes all attributes bound to it.

Servlets

Session Tracking with 'HttpSession'

Example:

```
<!DOCTYPE html>
<html>
<head>
    <title>Login</title>
</head>
<body>
    <h1>Login</h1>
    <form action="LoginServletHttpSession" method="post">
        <label for="username">Username:</label>
        <input type="text" name="username" id="username" required>
        <br><br>
        <label for="password">Password:</label>
        <input type="password" name="password" id="password" required>
        <br><br>
        <button type="submit">Login</button>
    </form>
</body>
</html>
```

login.html

Servlets

Session Tracking with 'HttpSession'

Example:

```java
import jakarta.servlet.*;
import jakarta.servlet.http.*;
import jakarta.servlet.annotation.WebServlet;
import java.io.*;
@WebServlet({ "/LoginServletHttpSession" })
public class LoginServletHttpSession extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        // Get username and password from the login form
        String username = request.getParameter("username");
        String password = request.getParameter("password");
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        if ("admin".equals(username) && "password".equals(password)) {
            // Create a session and store the username in it
            HttpSession session = request.getSession();
            session.setAttribute("username", username);
            session.setMaxInactiveInterval(60 * 60); // Set session timeout to 1 hour
            response.sendRedirect("WelcomeServletHttpSession");
        } else {
            out.println("<html><body>");
            out.println("<h1>Login Failed!</h1>");
            out.println("<a href='login.html'>Try Again</a>");
            out.println("</body></html>");
        }
    }
}
```

LoginServletHttpSession.java

Servlets

Session Tracking with 'HttpSession'

Example:

```java
import jakarta.servlet.*;
import jakarta.servlet.http.*;
import jakarta.servlet.annotation.WebServlet;
import java.io.*;
@WebServlet({ "/WelcomeServletHttpSession" })
public class WelcomeServletHttpSession extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        // Retrieve the current session, but do not create a new one
        HttpSession session = request.getSession(false);
        if (session != null && session.getAttribute("username") != null) {
            String username = (String) session.getAttribute("username");
            // Display Welcome Page
            out.println("<html><body>");
            out.println("<h1>Welcome, " + username + "!</h1>");
            out.println("<form action='LogoutServletHttpSession' method='post'>");
            out.println("<input type='submit' value='Logout'>");
            out.println("</form>");
            out.println("</body></html>");
        } else {
            // Redirect to Login Page if no valid session is found
            out.println("<html><body>");
            out.println("<h1>You are logged out and cannot access the Welcome page directly.
                                                              Please log in first.</h1>");
            request.getRequestDispatcher("login.html").include(request, response);
            out.println("</body></html>");
        }
    }
}
```

WelcomeServletHttpSession.java

Servlets

Session Tracking with 'HttpSession'

Example:

```java
import jakarta.servlet.*;
import jakarta.servlet.http.*;
import jakarta.servlet.annotation.WebServlet;
import java.io.*;
@WebServlet({ "/LogoutServletHttpSession" })
public class LogoutServletHttpSession extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        // Invalidate the session
        HttpSession session = request.getSession(false);
        if (session != null) {
            session.invalidate();
        }
        out.println("<html><body>");
        out.println("<h1>You have successfully logged out.</h1>");
        out.println("<a href='login.html'>Go to Login Page</a>");
        out.println("</body></html>");
    }
}
```

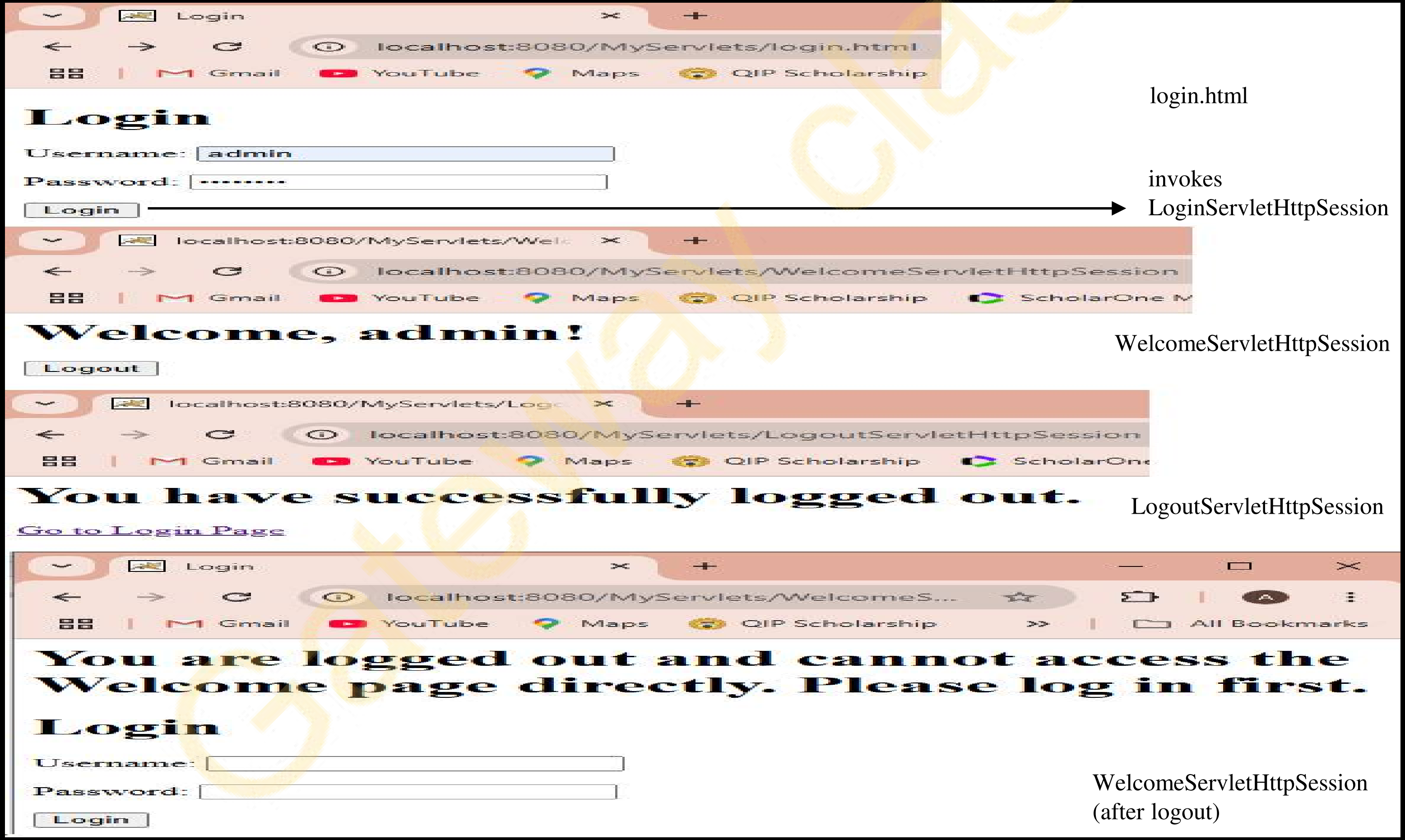
LogoutServletHttpSession.java

Servlets

'Cookie' class

Example:

(AKTU PYQ: Write a servlet program for servlet login and logout using cookies.*)*


localhost:8080/MySerlvets/login.html

Login

Username: admin

Password: ••••••••

Login

localhost:8080/MyServlets/WelcomeServletHttpSession

Welcome, admin!

Logout

localhost:8080/MyServlets/LogoutServletHttpSession

You have successfully logged out.

Go to Login Page

localhost:8080/MyServlets/WelcomeS...

You are logged out and cannot access the Welcome page directly. Please log in first.

Login

Username:

Password:

Login

login.html
invokes LoginServletHttpSession
WelcomeServletHttpSession
LogoutServletHttpSession
WelcomeServletHttpSession (after logout)

Output

Servlets

Servlets and CGI

AKTU PYQ: Define the advantages of servlet over CGI? Explain with an example.

In the early stages of web development, CGI (Common Gateway Interface) was used to handle client requests dynamically. However, CGI faced several limitations, such as performance issues and platform dependence. Servlets, a Java-based technology, were introduced to overcome these limitations.

The key advantages of servlets over CGI are as follows:

Comparison	CGI	Servlets
Performance	Creates a separate process for each client request, consuming more resources.	Executes within the web server's address space; no separate process needed, ensuring faster execution.
Platform Independence	Platform-dependent; written in various languages like C, C++, and Perl.	Platform-independent; written in Java and portable across operating systems.
Database Connectivity	Opens and closes database connections for every request, leading to inefficiency.	Maintains persistent database connections, ensuring efficient handling of operations.
Security	Limited security features, increasing the risk of unauthorized access.	Uses Java's security manager to enforce restrictions and protect server resources.
Access to Java Libraries	Not available.	Full access to Java class libraries, enabling advanced features like socket programming.
Session Tracking	Session tracking is challenging and requires custom implementation.	Built-in support for session tracking using mechanisms like HttpSession and cookies.

Servlets

Servlets and CGI

AKTU PYQ: Define the advantages of servlet over CGI? Explain with an example.

Example:

A web application retrieves user information from a database and displays it on a web page.

Using CGI:

1. A client sends a request to the web server.
2. The web server spawns a new process to handle the request.
3. The CGI program connects to the database, retrieves data, and sends it back to the web server.
4. The process is terminated after handling the request.

Disadvantage: High resource consumption due to process creation and termination for each request.

Using Servlets:

1. A client sends a request to the web server.
2. The servlet runs within the address space of the web server. The web server loads the servlet into its memory.
3. The servlet connects to the database (using persistent connections), retrieves the data, and sends it back as part of the response.
4. The servlet stays in memory and handles multiple requests.

Advantage: No new process creation, faster execution, and efficient database handling.

AKTU

B.Tech 5th Sem

GATEWAY CLASSES

CS IT & CS Allied

Web Technology

UNIT-5 Lecture-4

Today's Target

- Java Server Pages (JSP)
  - Introduction
  - Overview,
  - JSP Processing
  - JSP Life Cycle
  - A First JSP Example
  - Implicit Objects
- AKTU PYQs

By Amol Sharma sir
- Pursuing Ph.D. from IIT (BHU)
- Wipro Certified Faculty (WCF)

Java Server Pages (JSP)

Introduction

"Java Server Pages (JSP) is a server-side technology used for creating dynamic, platform-independent web applications. JSP allows developers to embed Java code directly into HTML pages using special JSP tags."

Overview

- JSP enables the creation of dynamic content by integrating Java with HTML.
- It simplifies the process of developing interactive and dynamic websites.
- Java code is inserted in the HTML using tags like <% ... %>, known as scriptlets, which are processed by the server to generate the final page.
- JSP is platform-independent and can run on any server that supports the Java Servlet API, making it versatile for different environments.

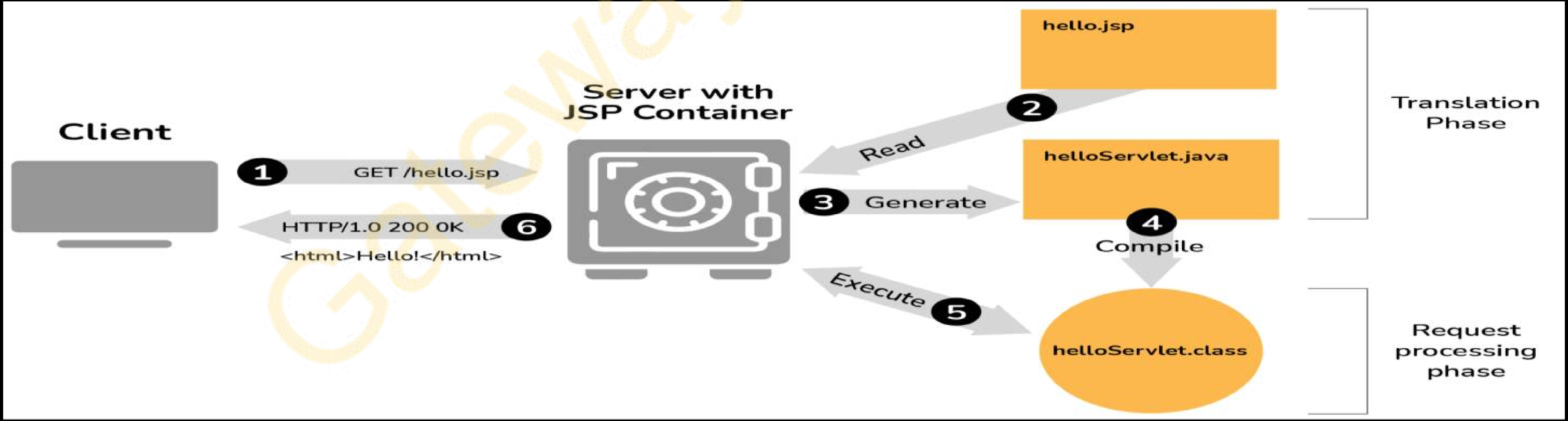
Java Server Pages (JSP)

JSP Processing

1. JSP Request: A client sends a request to the JSP page, which is processed by the server.

Translation Phase:

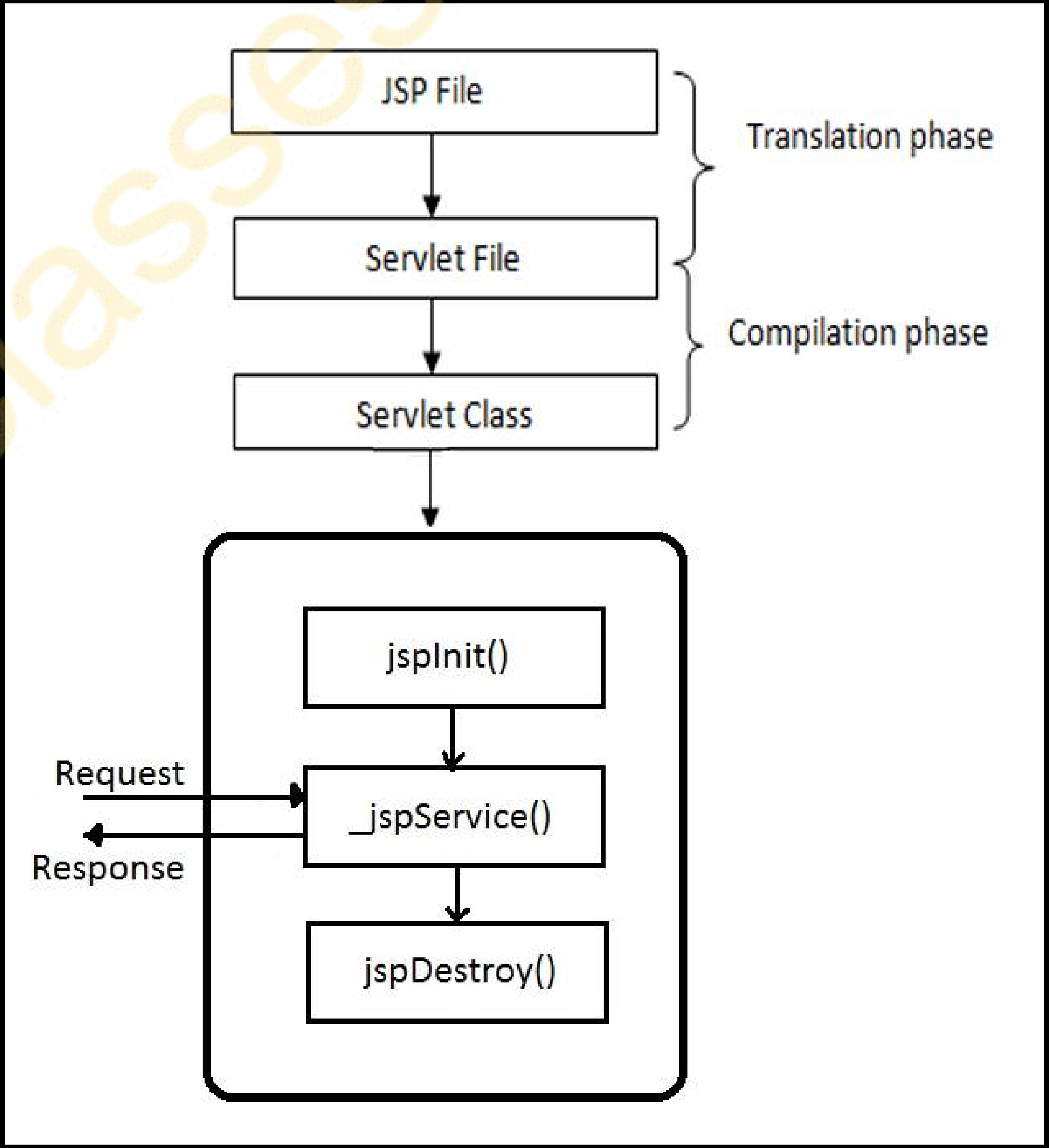
2. The JSP engine Reads the JSP Page and
3. translates it into a Java Servlet.
4. Compilation Phase: The generated Servlet is compiled into a bytecode class by the server's Java compiler and the compiled Servlet class is loaded into the JSP container.
5. Request Processing: The request is passed to the compiled Servlet, which processes the request and generates dynamic content.
6. Response Generation: The Servlet sends the dynamically generated HTML content back to the client as the HTTP response.



Java Server Pages (JSP)

JSP Life Cycle

1. Translation Phase:
   The JSP engine converts the JSP page into a Java Servlet (if not already compiled).
2. Compilation Phase:
   The generated Servlet is compiled into Java bytecode.
3. Initialization Phase:
   jspInit() method is called to perform any initialization tasks (e.g., setting up resources like database connections).
   This method is invoked once when the JSP is loaded.
4. Request Processing Phase:
   _jspService() method is called for every client request to process the request and generate dynamic content.This method handles the interaction between the JSP and the client.
5. Destroy Phase:
   jspDestroy() method is invoked when the JSP is unloaded from memory, allowing for cleanup of resources.This method is called once during the JSP page's lifecycle.



Java Server Pages (JSP)

A First JSP Example

```
<html>
<head>
    <title>My First JSP Page</title>
</head>
<body>
    <h2>Welcome to JSP!</h2>
    <p style="color: blue;"><strong><% out.print("Hello JSP World."); %></strong></p>
    <p>Current Date and Time: <%= new java.util.Date() %></p>
</body>
</html>
```

JSP Elements:

<% out.print("Hello JSP World."); %>

Scriptlet Tag: Embeds Java code within the JSP. out.print: Outputs the string "Hello JSP World." to the web page. The Java code inside the scriptlet runs on the server, and the output is sent to the client.

<%= new java.util.Date() %>

Expression Tag: Shorter syntax to embed and display the result of a Java expression. new java.util.Date() creates a new Date object representing the current date and time. The result of the expression is directly inserted into the web page's output.

Java Server Pages (JSP)

Implicit Objects

"In JSP, implicit objects are pre-defined objects that the container makes available to the developer without the need for explicit declaration."

- These objects provide easy access to essential components of the request, response, session, and application, among others.
- Implicit objects simplify common tasks such as handling HTTP requests, generating dynamic content, and managing sessions.

request

Represents the HttpServletRequest object. Used to retrieve request parameters, headers, and attributes.

response

Represents the HttpServletResponse object. Used to send data back to the client, such as setting headers or redirection.

out

Represents the JspWriter object. Used to send content as output to the client.

Java Server Pages (JSP)

Implicit Objects

session

Represents the HttpSession object. Used to manage user session data across multiple requests.

application

Represents the ServletContext object. Used to share data and access application-wide configuration details.

config

Represents the ServletConfig object. Used to retrieve initialization parameters for the JSP page.

pageContext

Provides access to all JSP-related objects and server-specific features, like attributes in different scopes (page, request, session, application).

page

A reference to the current JSP page object. Equivalent to this in Java.

exception

Available only in JSP error pages. Represents the exception object that caused the error.

Java Server Pages (JSP)

Implicit Objects - Example

implicitObjects.jsp

```
<%@ page language="java" contentType="text/html" errorPage="errorPage.jsp" %>
    <!DOCTYPE html>
    <html>
    <head>
        <title>Implicit Objects Demo</title>
    </head>
    <body>
        <h1>Demonstrating Implicit Objects in JSP</h1>

        <!-- request -->
        <p><strong>Request URI:</strong>
            <%= request.getRequestURI() %>
        </p>
        <p><strong>Client IP:</strong>
            <%= request.getRemoteAddr() %>
        </p>

        <!-- response -->
        <p><strong>Setting a Response Header:</strong>
            <% response.setHeader("Custom-Header", "JSP Demo" ); out.print("Custom Header added to the response!"); %>
        </p>

        <!-- out -->
        <p><strong>Output Content:</strong>
            <% out.print("This content is printed using the 'out' object."); %>
        </p>
```

Java Server Pages (JSP)

Implicit Objects

```
<!-- session -->
<p><strong>Session ID:</strong>
    <%= session.getId() %>
</p>
<p><strong>Setting Session Attribute:</strong>
    <% session.setAttribute("username", "Amol" ); out.print("Session attribute 'username' set to Amol.");
        %>
</p>

<!-- application -->
<p><strong>Application Context:</strong>
    <%= application.getServerInfo() %>
</p>

<p><strong>Setting Application Attribute:</strong>
    <% application.setAttribute("globalVar", "Global Value" );
    out.print("Application attribute 'globalVar' set."); %>
</p>

<!-- config -->
<p><strong>Page Name (from config):</strong>
    <%= config.getServletName() %>
</p>

<!-- pageContext -->
<p><strong>Accessing Attribute via pageContext:</strong>
    <% pageContext.setAttribute("tempVar", "Temporary Value" );
    out.print("Value of 'tempVar': " + pageContext.getAttribute("tempVar")); %>
</p>
```

Java Server Pages (JSP)

Implicit Objects

```
        <!-- page -->
        <p><strong>Current Page Object:</strong>
            <%= page.toString() %>
        </p>

        <!-- exception -->
        <p><strong>Error Simulation:</strong></p>
        <% if (request.getParameter("simulateError") !=null) { throw new RuntimeException("Simulated Exception"); } %>
            <p>Click <a href="implicitObjects.jsp?simulateError=true">here</a> to simulate an error.</p>

    </body>
    </html>
```

errorPage.jsp

```
<%@ page language="java" contentType="text/html" isErrorPage="true" %>
<!DOCTYPE html>
<html>
<head>
    <title>Error Page</title>
</head>
<body>
    <h1>An Error Occurred</h1>
    <p><strong>Exception Type:</strong> <%= exception.getClass().getName() %></p>
    <p><strong>Error Message:</strong> <%= exception.getMessage() %></p>
    <p>Return to the <a href="implicitObjects.jsp">main page</a>.</p>
</body>
</html>
```

AKTU

B.Tech 5th Sem

GATEWAY CLASSES

CS IT & CS Allied

Web Technology

UNIT-5 Lecture-5

Today's Target

- Java Server Pages (JSP)
  - Scripting,
  - Standard Actions,
  - Directives,
  - Custom Tag Libraries,
  - Comparison with Servlets.
- AKTU PYQs

By Amol Sharma sir

- Pursuing Ph.D. from IIT (BHU)
- Wipro Certified Faculty (WCF)

Java Server Pages (JSP)

Scripting

"Scripting elements in JSP allow developers to include Java code directly within the HTML structure."

- There are three main types of scripting elements: Declaration, Scriptlet, and Expression.
- Each serves a unique purpose in embedding and processing Java code.

Declaration Tag (<%! ... %>)

Used to declare variables, methods, or even classes that will be available to the entire JSP page.
This code is initialized only once when the Servlet is loaded.

Example:

```
<%! int count = 0; %>
<%! String getMessage() { return "Hello JSP!"; } %>
```

Scriptlet Tag (<% ... %>)

Allows embedding Java code that executes every time the JSP is requested.
The code within <% ... %> is included in the Servlet's service() method

Example:

```
<%    count++;
      if (count > 5) {        out.print("Count exceeded 5!");     }
%>
```

Java Server Pages (JSP)

Scripting

Expression Tag (<%= ... %>)

Simplifies outputting the value of a Java expression directly into the HTML response.
The result is automatically converted into a String and sent to the client.
Useful for displaying dynamic values like results or variables without using out.print().

Example:

```
<p>Today's date is: <%= new java.util.Date() %></p>
<p>Sum of 5 and 10 is: <%= 5 + 10 %></p>
%>
```

Summary of Scripting Elements

Declaration:	For global variables and methods.
Scriptlet:	For embedding logic to be executed at runtime.
Expression:	For directly printing values of expression into the response.

Java Server Pages (JSP)

Scripting

Example:

```
<!DOCTYPE html>
<html>
    <head>
      <title>User Visit Counter</title>
    </head>
    <body>
        <%!
          // Declaration: A variable to count visits
          int visitCount = 0;
        %>
        <%
          // Scriptlet: Increment the visitCount
          visitCount++;
        %>
        <h2>Welcome to the User Visit Counter</h2>
        // Expression: Outputs the value of visitCount
        <p>Your current visit number is: <strong><%= visitCount %></strong></p>
    </body>
</html>
```

Java Server Pages (JSP)

Standard Actions

"Actions are predefined tags in JSP that allow dynamic behavior and manipulation of the request and response within the JSP page."

- Using action tags, we can dynamically insert a file, reuse JavaBeans components, or forward the user to another page.
  Syntax: <jsp:action_name attribute="value" />

Action elements are basically predefined functions. Following are the standard action tags available in JSP:

<jsp:include>

Used to include one JSP or html inside another JSP.
It will include the response of the the included page at runtime.
Its functionality is similar to include() method of RequestDispatcher..

Example: <jsp:include page="includedPage.html"/>

Java Server Pages (JSP)

Standard Actions

<jsp:forward>

Forwards the requester to a new page/ resource.
It forwards the control from JSP to other resource,
Its functionality is similar to forward() method of RequestDispatcher..

Example: <jsp:forward page="forwardedPage.html"/>

<jsp:useBean>

Used to find or create the object of a Java Bean.

Example: <jsp:useBean id = "Student" class = "mybeans.Student"/>

Attributes:

id:	Name for the bean.
class:	Full class name of the bean including package(s).
scope:	(Optional) Scope of the bean (page, request, session, application).

Java Server Pages (JSP)

Standard Actions

<jsp:setProperty>

Sets the value of the property of a JavaBean.

Example: <jsp:setProperty name = "Student" property = "rollNo” value = "14" />

Attributes:

name:	Bean name.
property:	Property to set.
value:	Value to assign to the property.

<jsp:getProperty>

Retrieves the value of a bean’s property and outputs it to the page.

Example: <jsp:getProperty name = "Student" property = "rollNo" />

Attributes:

name:	Bean name.
property:	Property to get.

Java Server Pages (JSP)

Standard Actions

Example:

```
<html>
  <head>     <title>Using Standard Actions in JSP</title>   </head>
  <body>
       <h2>Using Standard Actions in JSP</h2>
      <jsp:useBean id = "Student" class = "com.mybeans.Student"/>
      <jsp:setProperty name = "Student"  property = "rollNo"  value = "14"/>
     <p>RollNo:
      <jsp:getProperty name = "Student" property = "rollNo" />
     </p>
     <jsp:setProperty name = "Student"  property = "name" value = "Amol Sharma"/>
     <p>Name:
      <jsp:getProperty name = "Student" property = "name" />
     </p>
     <jsp:include page="includedPage.html"/>
     <%-- <jsp:forward page="forwardedPage.html"/> --%>
  </body>
</html>
```

Java Server Pages (JSP)

Directives

"JSP directives are special instructions in JSP that provide information to the JSP container about how the JSP page should be processed."

- Directives define global settings and configuration for the JSP page.
- Directives are processed at compile-time, and affect the entire page.

Types of JSP Directives

Page Directive

Defines properties related to the entire JSP page.

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

Example: `<%@ page language="java" contentType="text/html" session="false" %>`

Attributes:

language:	Defines the programming language used in the JSP page (usually java).
contentType:	Specifies the content type (e.g., text/html).
session:	Whether the page uses session tracking (true or false).
isErrorPage:	Specifies if the JSP page handles errors (true or false).
errorPage:	Defines the page to redirect to when an error occurs.
import:	Imports Java classes or packages for use in the JSP page.
extends:	Specifies the parent class for the generated servlet from the JSP page.

Java Server Pages (JSP)

Directives

Types of JSP Directives

Include Directive

Includes an external file (e.g., HTML, JSP) at compile-time.

Syntax: `<%@ include file="filename" %>`

Example: `<%@ include file="header.jsp" %>`

This includes the contents of header.jsp at the time of compilation.

Taglib Directive

Defines a custom tag library and associates it with a prefix.

Syntax: `<%@ taglib uri="uri" prefix="prefix" %>`

Example: `<%@ taglib uri="/WEB-INF/tags.tld" prefix="custom" %>`

This imports the tag library with the prefix custom.

Java Server Pages (JSP)

Custom Tag Libraries

"Custom Tag Libraries allow developers to define custom tags for JSP pages. Custom tags are user-defined tags that can be used within JSP."

Advantages of Custom tags over Scriptlets

- Custom tags delegate processing to a separate Java class (tag handler) whereas Scriptlets embed Java code in JSP, mixing logic with presentation.
- JSP using Custom tags is easier to read and maintain.
- Custom tags can be reused across applications, unlike scriptlet code.
- Tags are simpler for frontend developers compared to writing Java code using Scriptlets.

Components of Custom Tag Libraries:

Tag Handler Class:	Implements the tag logic in Java.
Tag Library Descriptor (TLD):	XML file that defines tags and maps them to classes.
JSP Page:	Uses custom tags with the taglib directive.

Java Server Pages (JSP)

Custom Tag Libraries

Example:

HelloTag.java

```java
package com.mytags;

import jakarta.servlet.jsp.JspException;
import jakarta.servlet.jsp.JspWriter;
import jakarta.servlet.jsp.tagext.TagSupport;
import java.io.*;
public class HelloTag extends TagSupport {
    public int doStartTag() throws JspException {
        try {
            JspWriter out = pageContext.getOut();
            out.print("Hello from Custom Tag!");
        } catch (IOException e) {
            throw new JspException("Error in HelloTag", e);
        }
        return SKIP_BODY; // No body content to process
    }
}
```

tags.tld

```xml
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.1">
    <tlib-version>1.0</tlib-version>
    <tag>
        <name>hello</name>
        <tag-class>com.mytags.HelloTag</tag-class>
        <body-content>empty</body-content>
    </tag>
</taglib>
```

tagUse.jsp

```jsp
<%@ taglib uri="/WEB-INF/tags.tld" prefix="custom" %>
<html>
    <body>
        <custom:hello/>
    </body>
</html>
```

Java Server Pages (JSP)

Comparison with Servlets

Point of Comparison	JSP (Java Server Pages)	Servlets
Definition	JSP is a technology used to create dynamic web pages using HTML, CSS, and Java code.	Servlets are Java programs that handle requests and responses in a web application.
Primary Purpose	Focuses on the presentation layer (view).	Focuses on the business logic (controller).
Syntax	Allows mixing of HTML with Java code (via scriptlets or custom tags).	Pure Java code; HTML must be generated programmatically.
Ease of Development	Easier to develop and maintain as it is more HTML-centric.	More complex, as HTML must be written using Java code.
Separation of Concerns	Encourages separation of concerns with MVC (Model-View-Controller) patterns.	Often mixes logic and presentation in the same code, making maintenance harder.
Compilation	Compiled into a servlet by the JSP engine at runtime.	Manually written and compiled as a Java class.
Usage	Used for dynamic content and user interfaces.	Used for handling business logic and HTTP requests/responses.
Performance	Similar to servlets after compilation (as JSPs are converted to servlets).	High performance as they directly execute Java code.
Complexity	Simpler for developers with an HTML background.	More suited for Java developers with backend focus.
Reusability	Allows reuse through custom tags and JSP includes.	Reusability is limited and requires modular design.

AKTU PYQs

Web Technology (AKTU 2022 – 23)

1. Explain the implicit and explicit objects in JSP. Describe request, response and session objects in detail.
2. Describe Java Servlets. Write Index.html to input the some details (name, age, message) from user. Also write a servlet to fetch and display the same data upon the submission of request.
3. Compare Servlet and JSP. Justify why Servlets perform faster in comparison to JSP. Explain the life cycle of JSP with suitable diagram.

Web Technology (AKTU 2021 – 22)

1. Outline Implicit Objects in JSP.
2. Explain the Directory structure of web application.
3. Define the advantages of servlet over CGI? Explain with an example.
4. Illustrate why the session is required in web applications. Mention all the session tracking techniques in servlet with their pros and cons. Also give example of each.
5. Describe Servlet. Explain types of Servlet. Give an example of HttpServlet. Write steps to execute the servlet.

Web Technology (AKTU 2020 – 21)

1. Explain various JSP implicit objects in detail.
2. Describe Session Tracking. How Session Tracking using Http Session is performed?

AKTU PYQs

Web Technology (AKTU 2019 – 20)

1. Discuss about tomcat server. How to set the classpath for servlet in tomcat server?
2. Explain Servlets with its life cycle. How its lifecycle is different from the life cycle of JSP? Explain with an example.
3. Discuss JSP in details. What are JSP directives? Explain various types of directives with suitable example.

Web Technology (AKTU 2018 – 19)

1. Compare JSP and Servlet. Explain the lifecycle of a JSP page with a suitable diagram. Also list any five action tags used in JSP.
2. What is difference between Session and Cookies? Write a servlet program for servlet login and logout using cookies.

Thank you