CIE-306T

Advanced Java Programming

Running Servlets via XAMPP

B.Tech (CSE)
January, 2025

Nihar Ranjan Roy
Associate Professor,
VIPS School of Engineering & Technology
nihar.roy@vips.edu

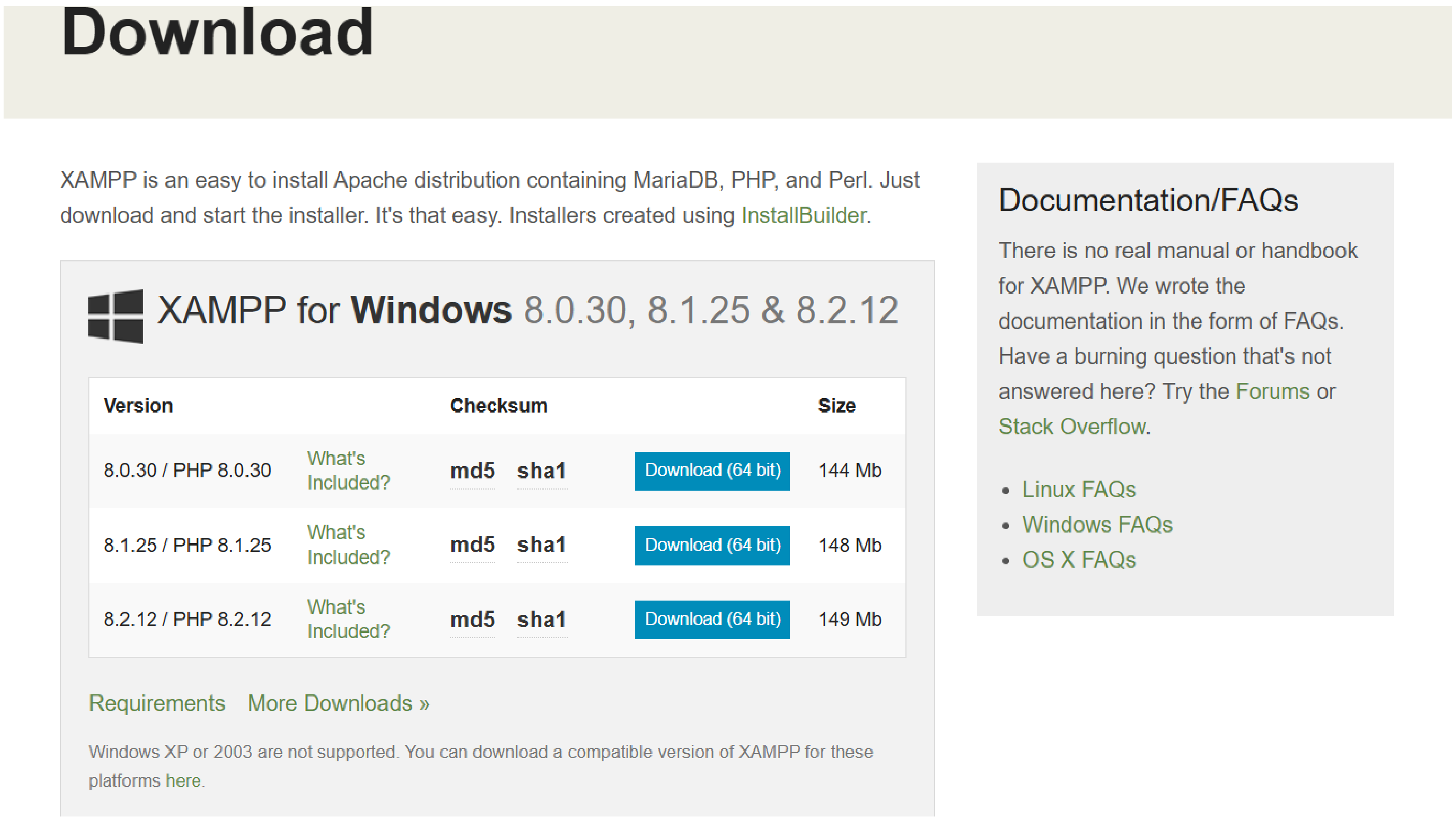
योगः कर्मसु कौशलम्
IN PURSUIT OF PERFECTION

Prerequisite

- Install XAMPP: Download and install XAMPP.
- Install Java JDK: Install and set up Java JDK (Ensure JAVA_HOME is set).
- Install Apache Tomcat: Download and extract Tomcat (Tomcat Download).
- Set Environment Variables:
  - Add JAVA_HOME and
  - CATALINA_HOME to system environment variables.

Download XAMPP

- https://www.apachefriends.org/download.html



Download

XAMPP is an easy to install Apache distribution containing MariaDB, PHP, and Perl. Just download and start the installer. It's that easy. Installers created using InstallBuilder.

XAMPP for **Windows** 8.0.30, 8.1.25 & 8.2.12

Version		Checksum			Size
8.0.30 / PHP 8.0.30	What's Included?	md5	sha1	Download (64 bit)	144 Mb
8.1.25 / PHP 8.1.25	What's Included?	md5	sha1	Download (64 bit)	148 Mb
8.2.12 / PHP 8.2.12	What's Included?	md5	sha1	Download (64 bit)	149 Mb

Requirements More Downloads »

Windows XP or 2003 are not supported. You can download a compatible version of XAMPP for these platforms here.

Documentation/FAQs

There is no real manual or handbook for XAMPP. We wrote the documentation in the form of FAQs. Have a burning question that's not answered here? Try the Forums or Stack Overflow.

- Linux FAQs
- Windows FAQs
- OS X FAQs

Configure Tomcat in XAMPP

Step 1: Locate Tomcat in XAMPP:

- If Tomcat is not installed, download and install it separately.
- If included in XAMPP, find it in C:\xampp\tomcat.

2.Start Tomcat in XAMPP:

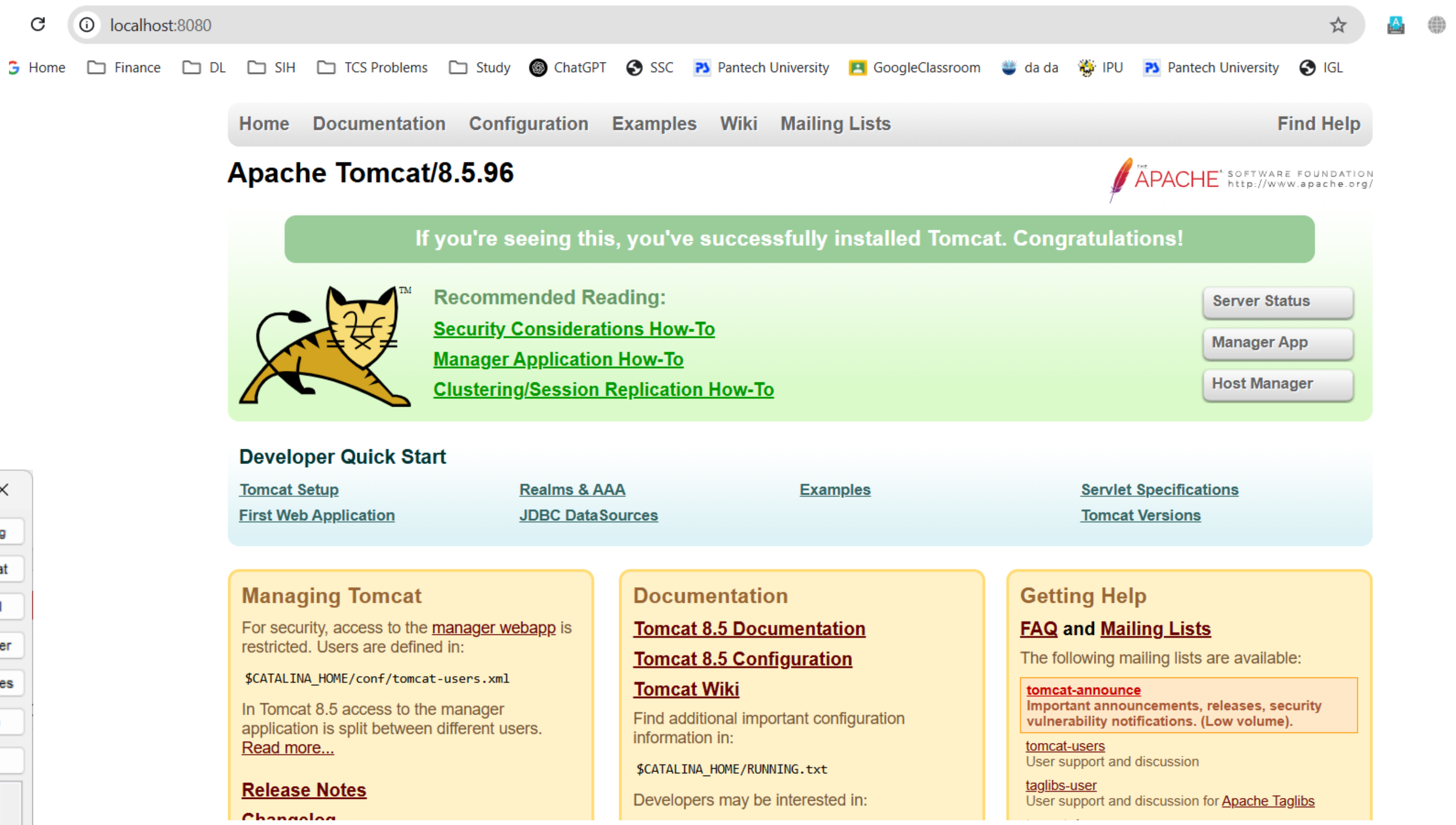
- Open XAMPP Control Panel.
- Start Apache and Tomcat.

3.Verify Tomcat:

- Open a browser and go to:

http://localhost:8080

- If Tomcat is running, you'll see the Tomcat welcome page.



localhost:8080
Home Finance DL SIH TCS Problems Study ChatGPT SSC Pantech University GoogleClassroom da da IPU Pantech University IGL
Home Documentation Configuration Examples Wiki Mailing Lists Find Help

Apache Tomcat/8.5.96

If you're seeing this, you've successfully installed Tomcat. Congratulations!

Recommended Reading:
Security Considerations How-To
Manager Application How-To
Clustering/Session Replication How-To

Server Status
Manager App
Host Manager

Developer Quick Start

Tomcat Setup	Realms & AAA	Examples	Servlet Specifications
First Web Application	JDBC DataSources		Tomcat Versions

Managing Tomcat

For security, access to the manager webapp is restricted. Users are defined in:

```
$CATALINA_HOME/conf/tomcat-users.xml
```

In Tomcat 8.5 access to the manager application is split between different users. Read more...

Release Notes

Documentation

Tomcat 8.5 Documentation
Tomcat 8.5 Configuration
Tomcat Wiki

Find additional important configuration information in:

```
$CATALINA_HOME/RUNNING.txt
```

Developers may be interested in:

Getting Help

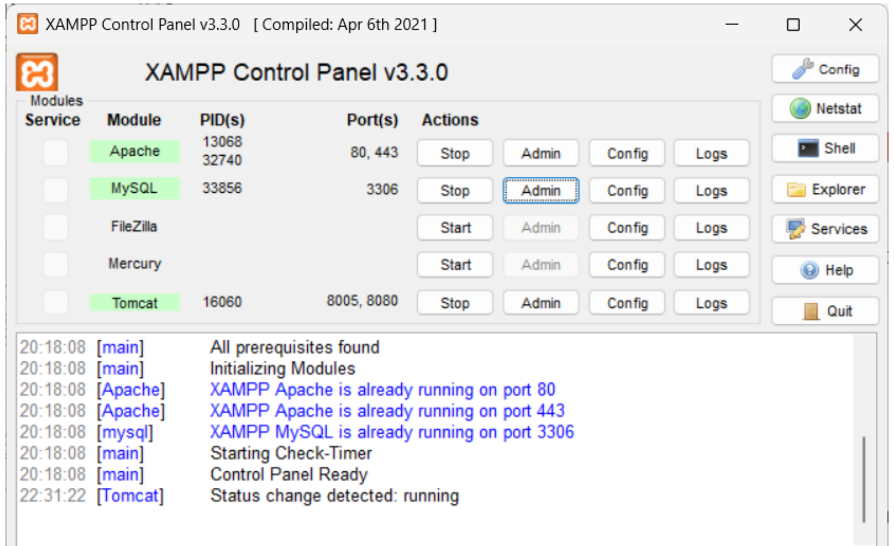
FAQ and Mailing Lists

The following mailing lists are available:

tomcat-announce
Important announcements, releases, security vulnerability notifications. (Low volume).

tomcat-users
User support and discussion

taglibs-user
User support and discussion for Apache Taglibs



XAMPP Control Panel v3.3.0 [Compiled: Apr 6th 2021]
XAMPP Control Panel v3.3.0
Config
Netstat
Shell
Explorer
Services
Help
Quit

Service	Module	PID(s)	Port(s)	Actions			
	Apache	13068 32740	80, 443	Stop	Admin	Config	Logs
	MySQL	33856	3306	Stop	Admin	Config	Logs
	FileZilla			Start	Admin	Config	Logs
	Mercury			Start	Admin	Config	Logs
	Tomcat	16060	8005, 8080	Stop	Admin	Config	Logs

```
20:18:08  [main]     All prerequisites found
20:18:08  [main]     Initializing Modules
20:18:08  [Apache]   XAMPP Apache is already running on port 80
20:18:08  [Apache]   XAMPP Apache is already running on port 443
20:18:08  [mysql]    XAMPP MySQL is already running on port 3306
20:18:08  [main]     Starting Check-Timer
20:18:08  [main]     Control Panel Ready
22:31:22  [Tomcat]   Status change detected: running
```

Set Up a Servlet Application

- Create a Project
    - `C:\xampp\tomcat\webapps\MyServletApp`
- Create Folder Structure:

```
myapps
├── WEB-INF
│    ├── web.xml
│    ├── classes  ──────────►  will have servlet files
├── index.html    ──────────►   html file
```

More detailed Directory Hierarchy

To facilitate creation of a Web Application Archive file in the required format, it is convenient to arrange the "executable" files of your web in your application's "document root" directory:

- *.html, *.jsp, etc. - The HTML and JSP pages, along with other files that must be visible to the client browser (such as JavaScript, stylesheet files, and images) for your application. In larger applications you may choose to divide these files into a subdirectory hierarchy, but for smaller apps, it is generally much simpler to maintain only a single directory for these files.
- /WEB-INF/web.xml - The Web Application Deployment Descriptor for your application. This is an XML file describing the servlets and other components that make up your application, along with any initialization parameters and container-managed security constraints that you want the server to enforce for you. This file is discussed in more detail in the following subsection.
- /WEB-INF/classes/ - This directory contains any Java class files (and associated resources) required for your application, including both servlet and non-servlet classes, that are not combined into JAR files. If your classes are organized into Java packages, you must reflect this in the directory hierarchy under /WEB-INF/classes/. For example, a Java class named `com.mycompany.mypackage.MyServlet` would need to be stored in a file named `/WEB-INF/classes/com/mycompany/mypackage/MyServlet.class`.
- /WEB-INF/lib/ - This directory contains JAR files that contain Java class files (and associated resources) required for your application, such as third party class libraries or JDBC drivers.

When you install an application into Tomcat (or any other 2.2 or later Servlet container), the classes in the WEB-INF/classes/ directory, as well as all classes in JAR files found in the WEB-INF/lib/ directory, are made visible to other classes within your particular web application. no adjustment to the system class path (or installation of global library files in your server) will be necessary.

Create Servlet Java File

•Inside C:\xampp\tomcat\webapps\myapps\WEB-INF\classes, create HelloServlet.java:

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class HelloServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<h1>Hello, Servlet!. Congratulations Nihar</h1>");
    }
}
```

Compile the Servlet:

- Open Command Prompt and navigate to classes folder:

```
cd C:\xampp\tomcat\webapps\MyServletApp\WEB-INF\classes
```

- Compile:

```
javac -cp C:\xampp\tomcat\lib\servlet-api.jar HelloServlet.java
```

If find it as lengthy command, you may create a batch file compile.bat with following content and use it frequently

```
javac -cp C:\xampp\tomcat\lib\servlet-api.jar %1
```

Now compile as below

```
Compile HelloServlet.java
```

Configure web.xml

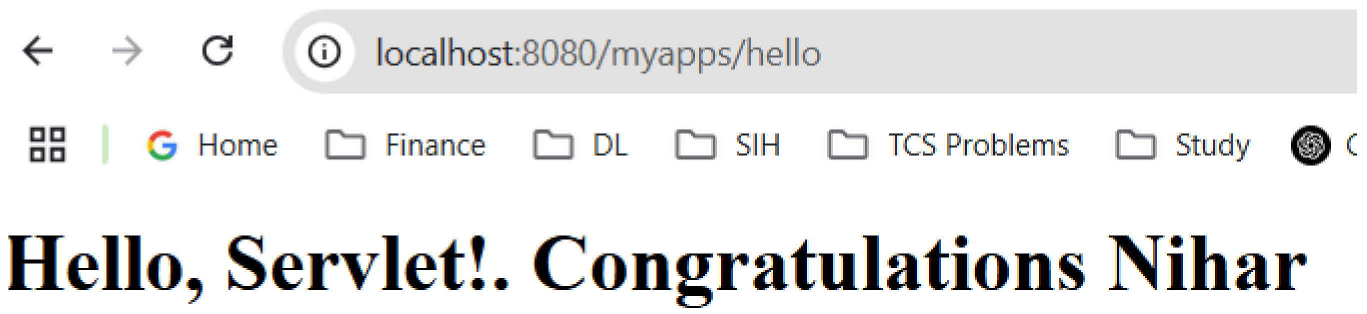
- Create web.xml in `C:\xampp\tomcat\webapps\MyServletApp\WEB-INF\`
- And write following content

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

Restart Tomcat and Test

- Restart Tomcat from the XAMPP control panel.
- Open a browser and enter

```
http://localhost:8080/myapps/hello
```



localhost:8080/myapps/hello

Hello, Servlet!. Congratulations Nihar

@WebServlet("/hello") annotation

- The annotation @WebServlet("/hello") is used in Java web applications to define a servlet and map it to a specific URL pattern.
- This annotation is part of the Java Servlet API (since Servlet 3.0) and eliminates the need to configure servlets in the web.xml file.
- Explanation:
    - @WebServlet is an annotation that marks a class as a servlet.
    - ("/hello") specifies the URL pattern at which this servlet will be accessible.
        - In this case, requests to http://localhost:8080/your-app/hello will be handled by this servlet.

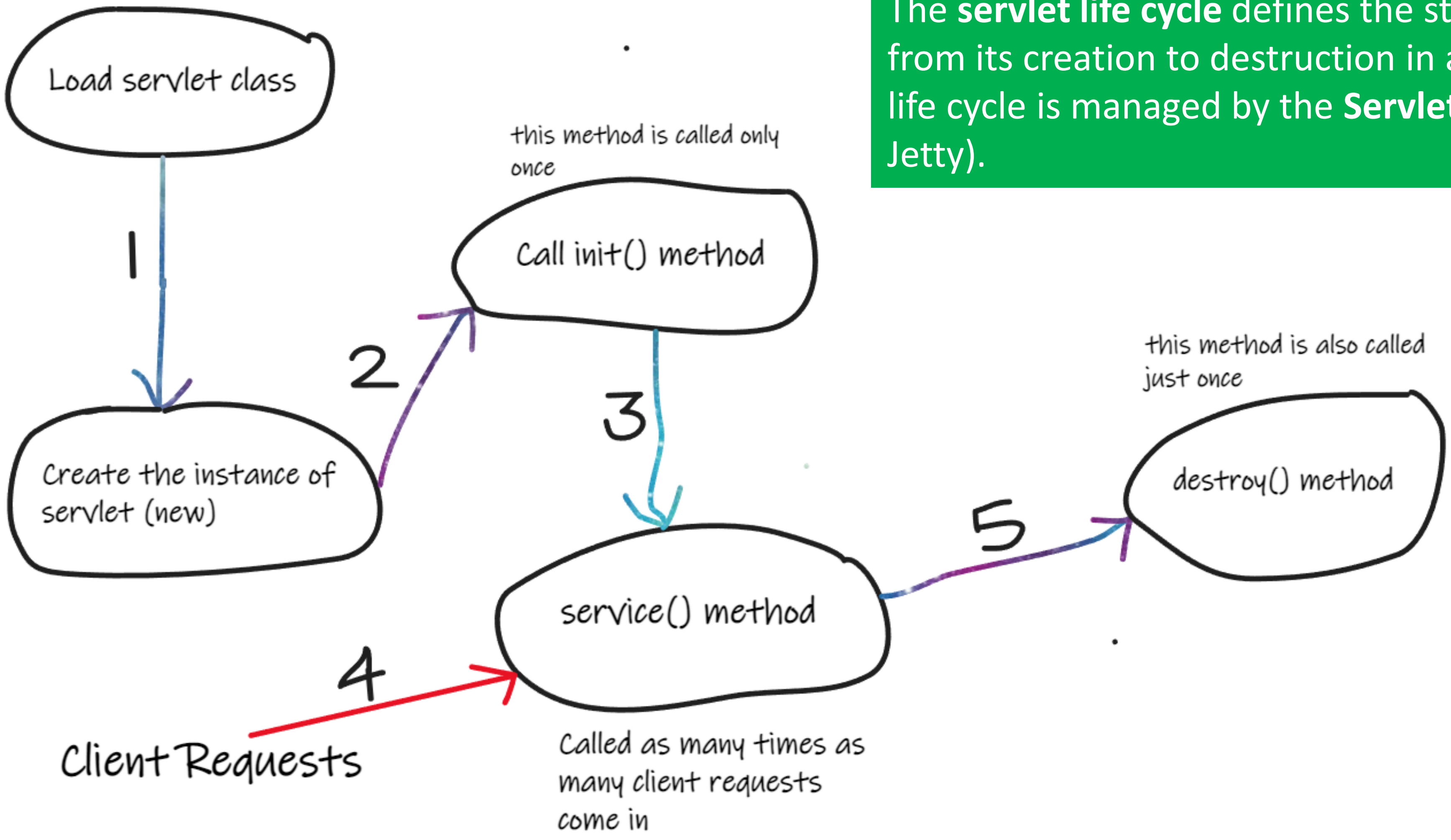
```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/hello")
public class HelloServlet extends HttpServlet {
    protected void doGet(HttpServletRequest
request, HttpServletResponse response)
            throws ServletException, IOException
{
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<h1>Hello, Servlet!.
Congratulations Nihar</h1>");
    }
}
```

Some more examples can be found at

- http://localhost:8080/examples/servlets/

Servlet Life cycle

The **servlet life cycle** defines the stages a servlet goes through from its creation to destruction in a Java web application. The life cycle is managed by the **Servlet Container** (e.g., Tomcat, Jetty).



Load servlet class
1
Create the instance of servlet (new)
2
this method is called only once
Call init() method
3
service() method
4
Client Requests
Called as many times as many client requests come in
5
this method is also called just once
destroy() method

Life Cycle of a Servlet

Servlet life cycle

1. Loading the Servlet class

As the first stage of servlet life cycle, the servlet container loads the servlet class. The loading process can happen in two ways: **1. Early loading, 2. Lazy Loading**

- **Early Loading :** The Servlet is loaded by the servlet container while initializing the context. (if configured with load-on-startup).
- **Lazy Loading :** In this case, the servlet is not loaded while initializing. But when the servlet container determines that there are requests for the servlet to respond, then it loads the servlet. It is known as lazy loading.

2. Instantiating the Servlet

As the next step, the servlet is instantiated, meaning object is created using the default constructor.

3. Call init() method

In this step, init() method of javax.servlet.Servlet is invoked and any initialization code written inside it is executed. This method is used to set any initialization parameters in the servlet.

Note -> init() is called only once in the life cycle of a servlet, init(ServletConfig config) .

```
public void init(ServletConfig config) throws ServletException
```

Servlet life cycle…

4. Request Handling (service())

- Whenever a request is sent to the servlet, the container calls the `service(HttpServletRequest, HttpServletResponse)` method.
- The service() method determines the HTTP request type (GET, POST, etc.) and calls the corresponding method (`doGet()`, `doPost()`, etc.).
- This method executes multiple times, once for each request.

5. Destruction (destroy())

- When the servlet is no longer needed, the container calls the `destroy()` method.
- This method is executed only once before the servlet is removed from memory.
- It is used to release resources like closing database connections, stopping background threads, etc.

Early Loading vs. Lazy Loading in Servlets

Lazy Loading (Default Behavior)

- By default, servlets in Java follow lazy loading. This means the servlet is not initialized when the server starts. Instead, it is loaded and initialized only when the first request is made.
- How Lazy Loading Works:
  - The servlet remains unloaded when the server starts.
  - When the first request comes, the servlet is loaded, instantiated, and initialized (init()).
  - Subsequent requests use the already loaded instance.
- Early Loading (Eager Loading)

  If we want the servlet to be loaded as soon as the server starts, we use the <load-on-startup> tag in web.xml.

```
<servlet>
    <servlet-name>EarlyServlet</servlet-name>
    <servlet-class>com.example.EarlyServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>EarlyServlet</servlet-name>
    <url-pattern>/early</url-pattern>
</servlet-mapping>
```

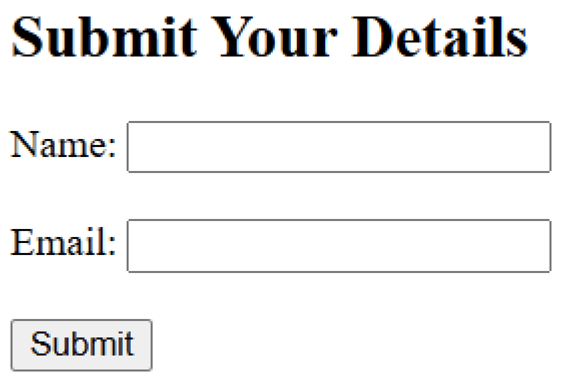
Value	Behavior
Omitted or -1	Lazy loading (default)
0 or positive number	Early loading (loaded at startup)
Lower number (1,2,3)	Higher priority (loaded first)

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/lifecycle")
public class LifeCycleServlet extends HttpServlet {
    @Override
    public void init() throws ServletException {
        System.out.println("Servlet initialized!");
    }
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.getWriter().println("Servlet is processing request...");
        System.out.println("Request handled.");
    }
    @Override
    public void destroy() {
        System.out.println("Servlet is being destroyed!");
    }
}
```

Handling Post requests

- Step-1: Create an HTML page to sent the POST request

```
<!DOCTYPE html>
<html>
<head>
    <title>POST Request Form</title>
</head>
<body>
    <h2>Submit Your Details</h2>
    <form action="handlePost" method="POST">
        Name: <input type="text" name="name"><br><br>
        Email: <input type="email" name="email"><br><br>
        <input type="submit" value="Submit">
    </form>
</body>
</html>
```



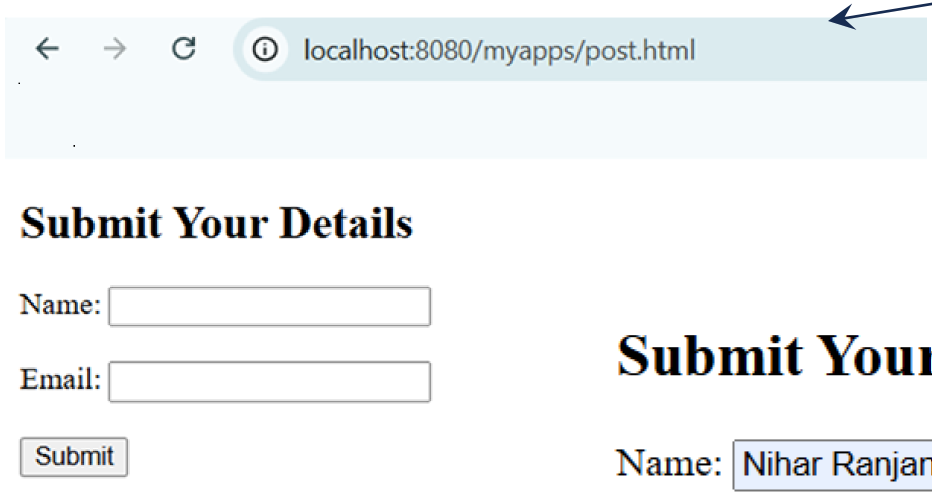
Handling Post requests

- **Step-2: Create servlet file to hand the POST request**

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.PrintWriter;
@WebServlet("/handlePost")  // URL mapping for this servlet
public class PostHandlerServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        // Set response content type
        response.setContentType("text/html");
        // Get form data from request
        String name = request.getParameter("name");
        String email = request.getParameter("email");
        // Prepare response
        PrintWriter out = response.getWriter();
        out.println("<html><body>");
        out.println("<h2>Received Data:</h2>");
        out.println("<p>Name: " + name + "</p>");
        out.println("<p>Email: " + email + "</p>");
        out.println("</body></html>");
    }
}
```

Execution

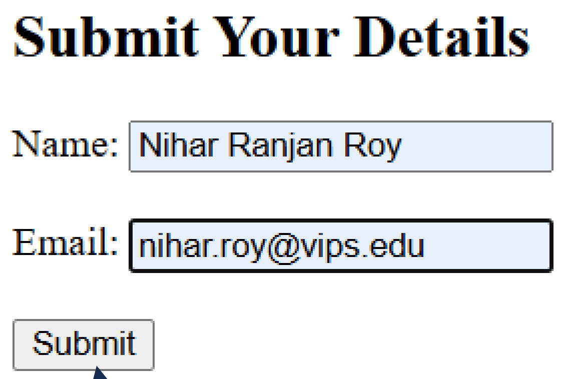
Access the html page→ post.html


localhost:8080/myapps/post.html

Submit Your Details

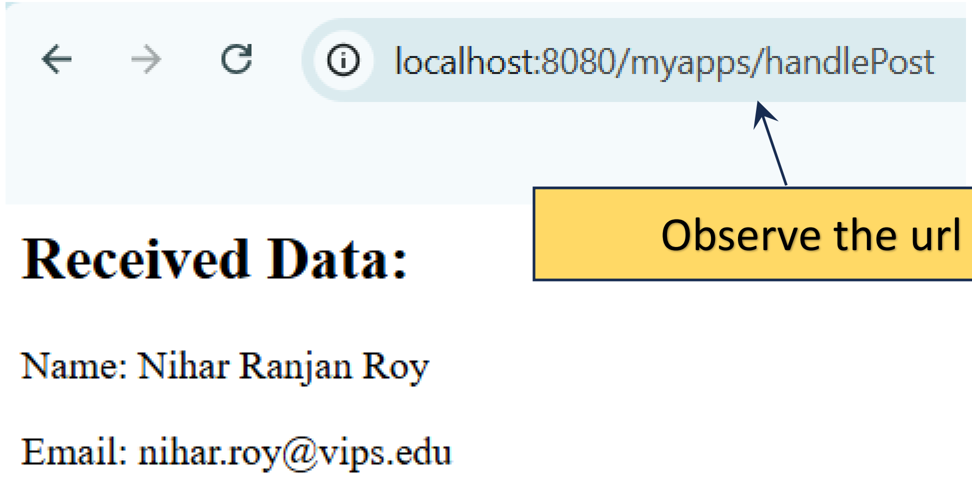
Name:

Email:


Submit Your Details

Name: Nihar Ranjan Roy

Email: nihar.roy@vips.edu


localhost:8080/myapps/handlePost

Received Data:

Name: Nihar Ranjan Roy

Email: nihar.roy@vips.edu

Observe the url

On click of submit button, along with data post request searches for servlet `handlepost` on the server

Problem

```
<!DOCTYPE html>
<html>
<head>
    <title>POST Request Form</title>
</head>
<body>
    <h2>Submit Your Details</h2>
    <form action="handlePost" method="GET">
        Name: <input type="text"
name="name"><br><br>
        Email: <input type="email"
name="email"><br><br>
        <input type="submit"
value="Submit">
    </form>
</body>
</html>
```

- What if the post method in html file is changed to GET?
- Is it possible?
- What changes in servlet code will be required?

Solution

Solution-1

1. Override `doGet` method also in the handler servlet.
2. Everything will be same that of `doPost` method only name will change to `doGet`

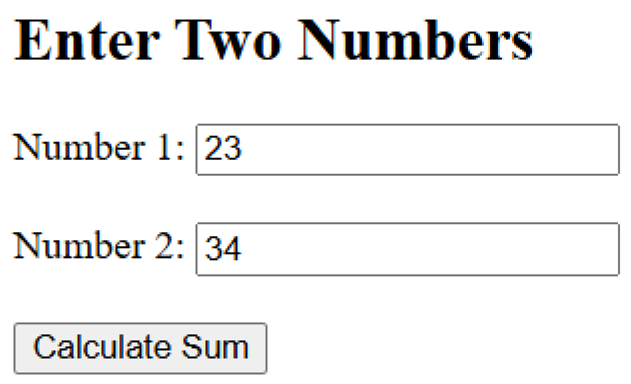
Solution-2

```
@Override

protected void doPost(HttpServletRequest
request, HttpServletResponse response)

        throws ServletException,
IOException {

    doGet(request, response); // Redirect
GET requests to POST handler

}
```

Which solution is better?

Problem

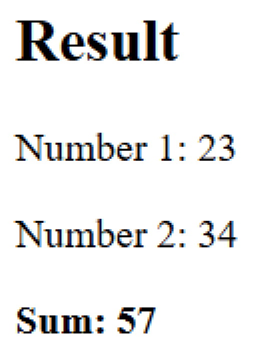
Design a servlet and an html file for addition of two numbers.



Enter Two Numbers

Number 1: 23

Number 2: 34



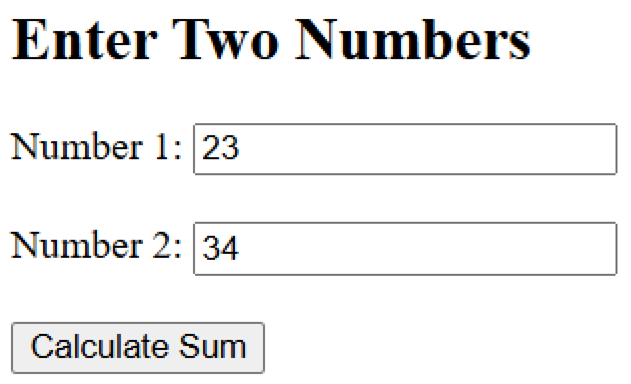
Result

Number 1: 23

Number 2: 34

Sum: 57

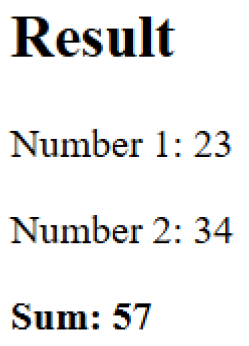
```
<!DOCTYPE html>
<html>
<head>
    <title>Addition Form</title>
</head>
<body>
    <h2>Enter Two Numbers</h2>
    <form action="AdditionServlet" method="POST">
        Number 1: <input type="number" name="num1" required><br><br>
        Number 2: <input type="number" name="num2" required><br><br>
        <input type="submit" value="Calculate Sum">
    </form>
</body>
</html>
```



Enter Two Numbers

Number 1: 23

Number 2: 34



Result

Number 1: 23

Number 2: 34

Sum: 57

```java
// assume necessary import packages
@WebServlet("/AdditionServlet")  // URL mapping
public class AdditionServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        // Set response content type
        response.setContentType("text/html");
              // Get numbers from the request
        int num1 = Integer.parseInt(request.getParameter("num1"));
        int num2 = Integer.parseInt(request.getParameter("num2"));

        // Calculate sum
        int sum = num1 + num2;
        // Display result
        PrintWriter out = response.getWriter();
        out.println("<html><body>");
        out.println("<h2>Result</h2>");
        out.println("<p>Number 1: " + num1 + "</p>");
        out.println("<p>Number 2: " + num2 + "</p>");
        out.println("<p><b>Sum: " + sum + "</b></p>");
        out.println("</body></html>");
    }
}
```

Problem

Write a servlet-based program where the servlets has record of number of times it has received client requests.

Solution

Only Get requests, what about Post requests?

What if the server is restarted? Counter should resume not reset.

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/RequestCounterServlet")   // URL Mapping
public class RequestCounterServlet extends HttpServlet {
    private static int requestCount = 0;  // Counter to track requests
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        requestCount++; // Increment request count
        response.setContentType("text/html"); // Set response content type
         // Display response
         PrintWriter out = response.getWriter();
         out.println("<html><body>");
         out.println("<h2>Servlet Request Counter</h2>");
         out.println("<p>This servlet has been accessed <b>" + requestCount + "</b>
times.</p>");
         out.println("</body></html>");
    }
}
```

Code for persistent counter

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

@WebServlet("/PersistentCounterServlet")  // URL Mapping
public class PersistentCounterServlet extends HttpServlet {
    private static final String FILE_PATH = "C:\\counter.txt"; // File to store count
    private static int requestCount = 0;

    @Override
    public void init() throws ServletException {
        // Load the request count from the file when the servlet starts
        requestCount = loadCount();
    }
```

Why here?

```
@Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
      // Increment request count
      requestCount++;

      // Save updated count to file
      saveCount(requestCount);

      // Set response content type
      response.setContentType("text/html");

      // Display response
      PrintWriter out = response.getWriter();
      out.println("<html><body>");
      out.println("<h2>Persistent Request Counter</h2>");
      out.println("<p>This servlet has been accessed <b>" + requestCount + "</b>
times.</p>");
      out.println("</body></html>");
    }
```

```
private int loadCount() {
        try (BufferedReader reader = new BufferedReader(new FileReader(FILE_PATH)))
{
            return Integer.parseInt(reader.readLine().trim());
        } catch (Exception e) {
            return 0; // If file doesn't exist or an error occurs, start from 0
        }
    }

    private void saveCount(int count) {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(FILE_PATH)))
{
            writer.write(String.valueOf(count));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Session Tracking in Servlets

- Session tracking is a technique used in web applications to maintain user-specific data across multiple requests from the same client.
- Since HTTP is stateless, each request is independent, meaning web servers do not retain information about users between requests.
- Session tracking allows identification of users and storage of their data while they interact with a web application.
- There are four common ways to track a session:
  - Cookies
  - Hidden Form Fields
  - URL Rewriting
  - Session API (HttpSession)

Using Cookies

- Cookies are small pieces of data stored in the user's browser. They help track users by storing a unique session ID.

```
import java.io.*;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;
@WebServlet("/CookieServlet")
public class CookieServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        // Creating a new cookie
        Cookie userCookie = new Cookie("username", “Nihar");
        userCookie.setMaxAge(60 * 60); // 1 hour
        response.addCookie(userCookie);
        out.println("<h3>Cookie has been set!</h3>");
    }
}
```

How It Works

1. A cookie named "username" is created and stored in the browser.
2. The browser sends this cookie with every subsequent request.

Retrieving cookies

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;
@WebServlet("/GetCookieServlet")
public class GetCookieServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        Cookie[] cookies = request.getCookies(); // Retrieve all cookies
        String userName = "Guest"; // Default value
        if (cookies != null) {
            for (Cookie cookie : cookies) {
                if (cookie.getName().equals("username")) {
                    userName = cookie.getValue();
                    break;
                }  }
        }
        out.println("<h3>Welcome, " + userName + "!</h3>");
    }}
```

Using Hidden Form Fields

- A hidden form field stores user information inside an HTML form and submits it with each request.

```
<form action="HiddenFieldServlet" method="POST">
    <input type="hidden" name="userId" value="12345">
    <input type="submit" value="Submit">
</form>
```

```
@WebServlet("/HiddenFieldServlet")
public class HiddenFieldServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        String userId = request.getParameter("userId");
        response.getWriter().println("User ID: " + userId);
    }
}
```

Using URL Rewriting

- When cookies are disabled, session data can be passed through the **URL itself.**

```
@WebServlet("/URLRewriteServlet")
public class URLRewriteServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String user = "nihar";
        out.println("<a href='NextServlet?user=" + user + "'>Click here to
continue</a>");
    }
}
```

How It Works

•The user information is **appended to the URL** as a query parameter.

•The next servlet can **extract the user parameter** from the URL.

Using HttpSession (Recommended)

- The HttpSession interface provides server-side session tracking.
- It allows you to store and retrieve user-specific data across multiple requests.

```
@WebServlet("/SessionServlet")
public class SessionServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        HttpSession session = request.getSession(); // Get or create session
        session.setAttribute("username", "nihar"); // Store session data
        response.getWriter().println("Session Created. Welcome, " +
session.getAttribute("username"));
    }
}
```

How It Works

- `request.getSession()` creates a new session (or retrieves an existing one).
- Data (e.g., "username") is stored in the session.
- The session persists across multiple requests.

Choosing the Right Session Tracking Method

Method	Advantages	Disadvantages
Cookies	Automatic, widely used	Users can disable cookies
Hidden Fields	Simple, no browser dependency	Works only for form submissions
URL Rewriting	Works without cookies	Exposes data in URL
HttpSession	Best for sensitive data, automatic management	Requires server memory

Response Handling Methods (From `HttpServletResponse`)

- These methods modify the response sent to the client.

Method	Description
`void sendRedirect(String url)`	Redirects client to a different page.
`void setContentType(String type)`	Sets the response type (e.g., text/html, application/json).
`void addCookie(Cookie cookie)`	Adds a cookie to the response.

- Example : redirecting user to another page

```
response.sendRedirect("home.jsp");
```

ServletContext and ServletConfig Methods

- These methods provide access to servlet configuration and shared application resources.

Method	Description
`ServletContext getServletContext()`	Retrieves the application-wide context.
`String getInitParameter(String name)`	Gets an initialization parameter from web.xml.

- Example (Reading web.xml Parameters):

```
String dbUser = getServletConfig().getInitParameter("dbUser");
```

Summary

Category	Important Methods
Lifecycle	`init()`, `service()`, `destroy()`
Request Handling	`doGet()`, `doPost()`, `doPut()`, `doDelete()`
Session Management	`getSession()`, `getParameter()`
Response Handling	`sendRedirect()`, `setContentType()`, `addCookie()`
ServletConfig & Context	`getInitParameter()`, `getServletContext()`

Thank You