**PRACTICAL # 10**

**OBJECT:**

**Java Servlets**

**THEORY:**

Java Servlets are programs that run on a Web or Application server. They act as a middle layer between a request coming from a Web browser or other HTTP client and the applications or databases running on the HTTP server.

Servlets provide a platform-independent method for building Web-based applications. Servlets can access all the Java APIs, including the JDBC API to access enterprise databases.

Using Servlets, you can get input from users through web page forms, present records from a database or another source, and create web pages dynamically.

Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But Servlets offer several advantages in comparison with the CGI.

* Performance is significantly better.
* Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request.
* Java Servlets are platform-independent.
* Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. So servlets are trusted.
* The full functionality of the Java class libraries is available to a servlet.

## **Servlets Architecture:**

A typical **servlet** architecture is designed as follows

**HTTP client (browser) ↔ HTTP Server ↔ Servlet Program ↔ Database**

## **Servlets Packages**

Java Servlets are Java classes run by web server.

Servlets can be created using the **javax.servlet** and **javax.servlet.http** packages, which are part of the Java enterprise edition, the Java class library that supports large-scale projects.

Java servlets can be created and compiled just like any other Java class. After installing servlet packages, add them to the Classpath or environment path. Compile servlet applications with the Java compiler.

## **Setting up Web Server: Tomcat**

After installing Java JDK, you need to setup a web server for the servlet applications. A number of Web Servers that support servlets are available. In this lab, we will use Apache Tomcat.

Apache Tomcat supports the Java Servlet and Java Server Pages (JSP) technologies and can act as a standalone server for testing servlets. Here are the steps to setup Tomcat on your machine:

Download the installation package, and extract the binary distribution to a convenient location. For example in C:\apache-tomcat-5.5.29 on windows and create CATALINA\_HOME environment variable pointing to the location.

Tomcat can be started by executing the following commands on windows machine:

%CATALINA\_HOME%\bin\startup.bat

or

C:\apache-tomcat-5.5.29\bin\startup.bat

After startup, the default web applications included with Tomcat will be available by visiting [**http://localhost:8080/**](http://localhost:8080/).

Tomcat can be stopped by executing the following commands on windows machine:

C:\apache-tomcat-5.5.29\bin\shutdown

**Setting up Environment Variables**

On Windows, right-click on My Computer, select Properties, then Advanced, then Environment Variables. Then, you would update the CLASSPATH value and press the OK button.

CATALINA=C:\apache-tomcat-5.5.29

CLASSPATH=%CATALINA%\common\lib\servlet-api.jar;%CLASSPATH%

**Servlet life cycle**

* The servlet is initialized by calling the **init()** method.
* The servlet calls **service()** method to process a client's request.
* The servlet is terminated by calling the **destroy()** method.
* Finally, servlet is garbage collected by the garbage collector of the JVM.

## **The init() method**

The init method is called only once, when the servlet is first created, and not called again for each user request.

The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.

When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to **doGet** or **doPost** as appropriate. The init() method simply creates or loads some data that will be used throughout the life of the servlet.

public void init() throws ServletException {

// Initialization code...

}

## **The service() method**

This is the main method to perform the actual task. The servlet container (i.e. web server) calls the service() method to handle requests coming from the client and to write the formatted response back to the client.

Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

Override either doGet() or doPost() depending on what type of request you receive from the client.

The doGet() and doPost() are most frequently used methods with in each service request. Here is the signature of these two methods.

## The doGet() Method

A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

public void doGet(HttpServletRequest request,  
HttpServletResponse response)

throws ServletException, IOException {

// Servlet code

}

## doPost() Method

A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

public void doPost(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException {

// Servlet code

}

## The destroy() method :

The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.

After the destroy() method is called, the servlet object is marked for garbage collection. The destroy method definition looks like this:

public void destroy{

// Finalization code...

}

**Program:**

***import java.io.\*;***

***import javax.servlet.\*;***

***import javax.servlet.http.\*;***

***public class HelloServlet extends HttpServlet {***

***private String message;***

***public void init() throws ServletException***

***{***

***// Do required initialization***

***message = "Hello Servlet";***

***}***

***public void doGet(HttpServletRequest request, HttpServletResponse response)***

***throws ServletException, IOException***

***{***

***// Set response content type***

***response.setContentType("text/html");***

***// Actual logic goes here.***

***PrintWriter out = response.getWriter();***

***out.println("<h1>" + message + "</h1>");***

***}***

***public void destroy()***

***{***

***// do nothing.***

***}***

***}***

## **Compiling a Servlet:**

Put above code in Index.java file and put this file in C:\Projects\ServletEx. You would also need to add these directories in CLASSPATH.

Now change directory to the project directory and Compile and execute Index.java file.

Example 2: Getting request parameters.

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

// Extend HttpServlet class

public class HelloForm extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

// Set response content type

response.setContentType("text/html");

PrintWriter out = response.getWriter();

String title = "Using GET Method to Read Form Data";

String docType =

"<!doctype html public \"-//w3c//dtd html 4.0 " + "transitional//en\">\n";

out.println(docType +

"<html>\n" +

"<head><title>" + title + "</title></head>\n" +

"<body bgcolor = \"#f0f0f0\">\n" +

"<h1 align = \"center\">" + title + "</h1>\n" +

"<ul>\n" +

" <li><b>First Name</b>: "

+ request.getParameter("first\_name") + "\n" +

" <li><b>Last Name</b>: "

+ request.getParameter("last\_name") + "\n" +

"</ul>\n" +

"</body>

</html>"

);

}

}

Calling servlet with example:

http://localhost:8080/Sevlet?first\_name = ZARA&last\_name = ALI

**ACTIVITIES**

**Activity 1**

Write

**Activity 2**

Find

**REVIEW QUESTIONS**

1. How
2. What
3. What