Prac 6) Write a program for creating simple Servlet with JDBC

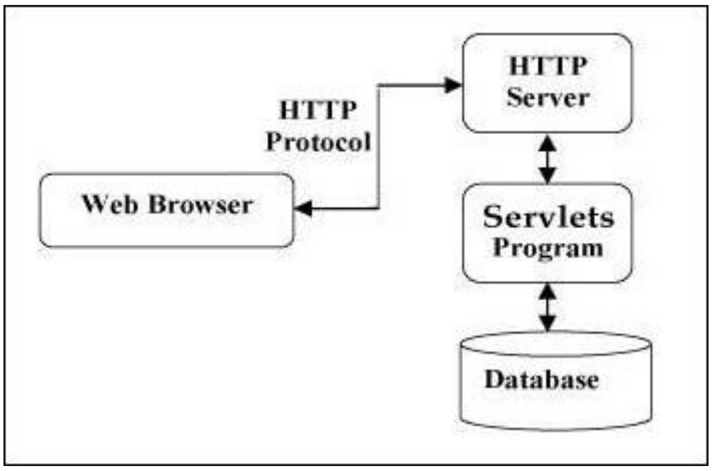
What are Servlets?

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

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

There are several varieties of interfaces and classes available in the Servlet API. Some of them are as follows:

- HTTP Servlet
- Generic Servlet
- Servlet Request
- Servlet Response



Java Servlet Life-Cycle

The Java Servlet Life cycle includes three stages right from its start to the end until the Garbage Collector clears it. These three stages are described below.

- init()
- service()
- destroy()

1. init()

The init method is called only once. It is called only when the servlet is created, and not called for any user requests afterwards. So, it is used for one-time initializations, just as with the init method of applets.

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... }
```

2. service()

The service() method 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(browsers) 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.

```
public void service(ServletRequest request, ServletResponse response) throws
ServletException, IOException { }
```

The doGet() and doPost() are most frequently used methods with in each service request.

2.1) 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.

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the ? (question mark) symbol as follows −

http://www.test.com/hello?key1 = value1&key2 = value2

The GET method is the default method to pass information from browser to web server and it produces a long string that appears in your browser's Location:box. Never use the GET method if you have password or other sensitive information to pass to the server. The GET method has size limitation: only 1024 characters can be used in a request string.

This information is passed using QUERY_STRING header and will be accessible through QUERY_STRING environment variable and Servlet handles this type of requests using doGet() method.

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException { // Servlet code }
```

2.2) The 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.

A generally more reliable method of passing information to a backend program is the POST method. This packages the information in exactly the same way as GET method, but instead of sending it as a text string after a ? (question mark) in the URL it sends it as a separate message. This message comes to the backend program in the form of the standard input which you can parse and use for your processing. Servlet handles this type of requests using doPost() method.

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException { // Servlet code }
```

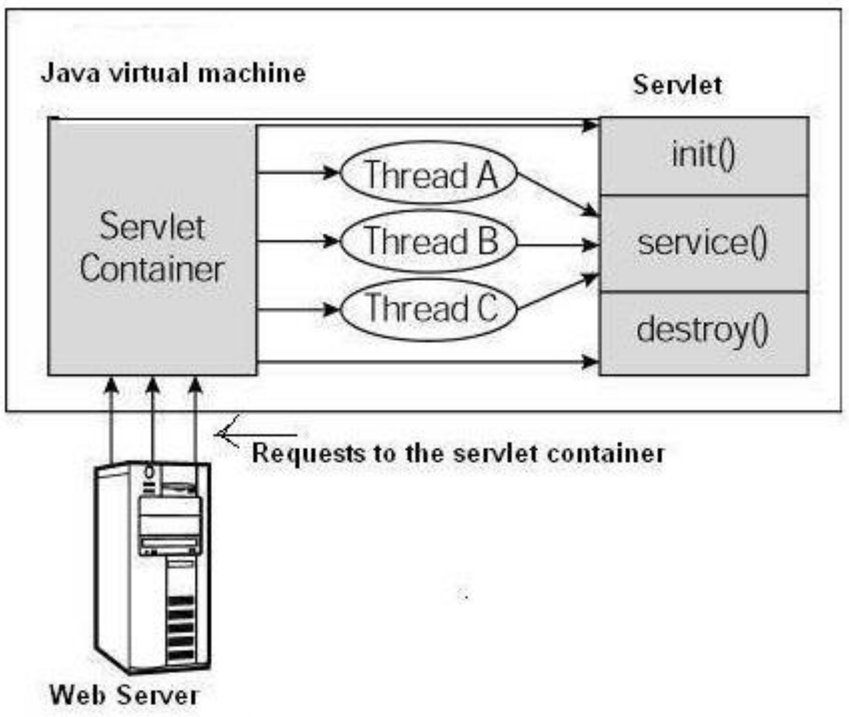
3. destroy()

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

```
public void destroy() { // Finalization code... }
```

Architecture Diagram



- First the HTTP requests coming to the server are delegated to the servlet container.
- The servlet container loads the servlet before invoking the service() method.
- Then the servlet container handles multiple requests by spawning multiple threads, each thread executing the service() method of a single instance of the servlet.

Reading Form Data using Servlet

Servlets handles form data parsing automatically using the following methods depending on the situation −

- getParameter() − You call request.getParameter() method to get the value of a form parameter.
- getParameterValues() − Call this method if the parameter appears more than once and returns multiple values, for example checkbox.
- getParameterNames() − Call this method if you want a complete list of all parameters in the current request.

Java Servlet Request

The Java Servlets operate in the form of Requests and Responses. The first phase is the request from the user's end. The HttpRequest object represents the HTTP request to a browser that the user sends to the web application. Thus, anything the browser sends is accessible through the HttpRequest.

1. HTTP Request Header
2. HTTP Request Parameters
3. HTTP Request InputStream
4. HTTP Request Context
5. HTTP Request Session

Java Servlet Response

The second important part is the Response phase. The HttpResponse object is used to represent the HTTP response to your request. A web application sends back a response page to the user to respond to the HTTP request from the browser sent to your web application.

parts of the Response Phase in the Java Servlets as follows.

1. HTTP Response Header
2. HTTP Response Content Type
3. HTTP Response Content Length
4. HTTP Response Write HTML
5. HTTP Response Redirection

Servlets - Http Status Codes

The status line consists of the HTTP version ,a status code, and a very short message corresponding to the status code.

Following is a list of HTTP status codes and associated messages that might be returned from the Web Server **–**

1. 404 - Not Found - The server cannot find the requested page.

(Search and Write any 10 HTTP Status code and their meaning here**)**

(Also write ur project Structure here)

RegisterServelet

```java
package com.example;

import java.io.*;


import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

import java.sql.*;

@WebServlet("/register")
public class RegisterServlet extends HttpServlet {

    static final String URL = "jdbc:mysql://localhost:3306/studentdb";
    static final String USER = "root";
    static final String PASS = "root";

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

        int id = Integer.parseInt(request.getParameter("id"));
        String name = request.getParameter("name");
        String email = request.getParameter("email");

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

        String insertSQL = "INSERT INTO students (id, name, email) VALUES (?, ?, ?)";

        try {

            Class.forName("com.mysql.cj.jdbc.Driver");

            try (Connection conn = DriverManager.getConnection(URL, USER, PASS);
                 PreparedStatement ps = conn.prepareStatement(insertSQL)) {

                ps.setInt(1, id);
                ps.setString(2, name);
                ps.setString(3, email);
                int rows = ps.executeUpdate();

                if (rows > 0) {
                    out.println("<h2>Student Registered Successfully!</h2>");
                } else {
                    out.println("<h2>Registration Failed.</h2>");
                }
            }
            try (Connection conn = DriverManager.getConnection(URL, USER, PASS);
                 Statement stmt = conn.createStatement();
                 ResultSet rs = stmt.executeQuery("SELECT * FROM students")) {

                out.println("<h2>Registered Students</h2>");
                out.println("<table border='1' cellpadding='8'>");
                out.println("<tr><th>ID</th><th>Name</th><th>Email</th></tr>");

                while (rs.next()) {
                    out.println("<tr>");
                    out.println("<td>" + rs.getInt("id") + "</td>");
                    out.println("<td>" + rs.getString("name") + "</td>");
                    out.println("<td>" + rs.getString("email") + "</td>");
                    out.println("</tr>");
                }

                out.println("</table>");
            }

        } catch (Exception e) {
            e.printStackTrace();
            out.println("<h3>Error: " + e.getMessage() + "</h3>");
        }
    }
}
```

Index.html

```html
<!DOCTYPE html>
<html>
<head>
    <title>Student Registration</title>
</head>
<body>
    <h2>Student Registration</h2>
    <form action="register" method="post">
        ID: <input type="text" name="id"><br><br>
        Name: <input type="text" name="name"><br><br>
        Email: <input type="text" name="email"><br><br>
        <input type="submit" value="Register">
    </form>
</body>
</html>
```