

ADVANCED JAVA PROGRAMMING LAB

PRACTICAL FILE

Faculty Name: Ms. Sakshi Jha

Student's Name: Harsh Dogra

Roll No.: 00514812721

Semester: 6

Group: 6 CST1 AIML



Maharaja Agrasen Institute of Technology

Sector – 22, Rohini, New Delhi – 110085

PROGRAM-9

AIM: Implement Producer-Consumer Problem using multithreading.

OVERVIEW:

The producer-consumer problem is a fundamental challenge in computer science that arises when two processes share a limited buffer. One process, the producer, creates data and adds it to the buffer. The other process, the consumer, removes data from the buffer and processes it. The key difficulty is ensuring synchronization between the producer and consumer to avoid data loss or corruption. This problem is a classic example of inter-process communication and is essential for understanding concurrency control in operating systems.

SOURCE CODE:

```
import java.util.concurrent.locks.Condition;

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;

public class ProducerConsumer { static class Buffer {

private final int size;

private final String[] items;

private int head = 0;

private int tail = 0;

private final Lock lock = new ReentrantLock();

private final Condition notFull = lock.newCondition();

private final Condition notEmpty = lock.newCondition();

public Buffer(int size) { this.size = size; this.items = new

String[size];

}

public void produce(String item) throws InterruptedException {

lock.lock();

try {
```

```

while ((tail + 1) % size == head) { //

Buffer is full, wait for consumption

notEmpty.await();

}

items[tail] = item; tail = (tail + 1) % size;

notEmpty.signal(); // Notify waiting consumer

System.out.println("Produced: " + item);

} finally { lock.unlock();

}

}

public String consume() throws InterruptedException {

lock.lock(); try {

while (head == tail) { // Buffer is empty,

wait for production notEmpty.await();

}

String item = items[head]; head =

(head + 1) % size;

notEmpty.signal(); // Notify waiting producer

System.out.println("Consumed: " + item); return

item;

} finally {

lock.unlock();

```

```

}

public static void main(String[] args) throws InterruptedException {

    int bufferSize = 5;

    Buffer buffer = new Buffer(bufferSize);

    Thread producerThread = new Thread(() -> {

        String[] items = {"Item 1", "Item 2", "Item 3", "Item 4", "Item

        5"}; for (String item : items) { try { buffer.produce(item);

            Thread.sleep(1000); // Simulate production time

        } catch (InterruptedException e) {

            e.printStackTrace();

        }

    }

});

    Thread consumerThread = new Thread(() -> {

        while (true) { try {

            buffer.consume();

            Thread.sleep(500); // Simulate consumption time

        } catch (InterruptedException e) {

            e.printStackTrace();

        }

    } });

```

OUTPUT:

```
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent  
Produced: Item 1  
Consumed: Item 1  
Produced: Item 2  
Consumed: Item 2  
Produced: Item 3  
Consumed: Item 3
```

PROGRAM-10

AIM: Illustrate Priorities in Multithreading via help of `getPriority()` and `setPriority()` method.

OVERVIEW:

Java threads have priorities (1-10) influencing CPU access. `getPriority()` retrieves the current priority, while `setPriority()` attempts to change it (actual priority might be lower due to system restrictions). Use these methods to create scenarios where higher priority threads are more likely to be scheduled first by the CPU, showcasing the impact of priority on thread execution order.

SOURCE CODE:

```
public class PriorityDemo {

    public static void main(String[] args) {
        Thread highPriorityThread = new Thread() -> {
            System.out.println("High Priority Thread: Priority - " +
                Thread.currentThread().getPriority()); // Simulate some work for
            (int i = 0; i < 10000; i++) {
                Math.sqrt(i);
            }
        });

        Thread lowPriorityThread = new Thread() -> {
            System.out.println("Low Priority Thread: Priority - " +
                Thread.currentThread().getPriority()); // Simulate some work for
            (int i = 0; i < 100000; i++) {
                Math.sin(i);
            }
        });

        // Set priorities (might be adjusted by the system)
        highPriorityThread.setPriority(Thread.MAX_PRIORITY);
        lowPriorityThread.setPriority(Thread.MIN_PRIORITY);

        highPriorityThread.start(); lowPriorityThread.start();
    }
}
```

OUTPUT:

```
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent
Low Priority Thread: Priority - 1
High Priority Thread: Priority - 10

Process finished with exit code 0
```

PROGRAM-11

AIM: Illustrate Deadlock in multithreading.

OVERVIEW:

Deadlock in multithreading occurs when two or more threads are permanently blocked, waiting for resources held by each other. Imagine Thread A holding Resource 1 and waiting for Resource 2 held by Thread B. Simultaneously, Thread B holds Resource 2 and waits for Resource 1 held by Thread A. Neither thread can proceed, creating a deadlock. To avoid this, acquire resources in a consistent order or use techniques like timeouts to prevent indefinite waiting.

SOURCE CODE:

```
public class ANUDeadlockDemo {

    static Object resource1 = new Object(); static
    Object resource2 = new Object();

    public static void main(String[] args) {
        Thread thread1 = new Thread(() -> { synchronized
            (resource1) {
                System.out.println("Thread 1: Acquired resource 1"); try {
                    Thread.sleep(1000); // Simulate work
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                System.out.println("Thread 1: Waiting for resource 2"); synchronized
                (resource2) {
                    System.out.println("Thread 1: Acquired resource 2 (This should not happen due to
deadlock)");
                }
            }
        });

        Thread thread2 = new Thread(() -> { synchronized
            (resource2) {
                System.out.println("Thread 2: Acquired resource 2"); try {
                    Thread.sleep(1000); // Simulate work
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                System.out.println("Thread 2: Waiting for resource 1"); synchronized
                (resource1) {
                    System.out.println("Thread 2: Acquired resource 1 (This should not happen due to
deadlock)");
                }
            }
        });

        thread1.start(); thread2.start();
    }
}
```

OUTPUT:

```
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent  
Thread 1: Acquired resource 1  
Thread 2: Acquired resource 2  
Thread 1: Waiting for resource 2  
Thread 2: Waiting for resource 1
```


PROGRAM-12

AIM: Illustrate use of java beans.

OVERVIEW: JavaBeans facilitate reusable software components in Java. They encapsulate functionality, expose properties accessed via getter/setter methods, and handle events. Supporting customization and design-time features, they follow a lifecycle model and support serialization. Adhering to naming conventions, JavaBeans ensure interoperability and platform independence, enabling portable and versatile component-based development.

SOURCE CODE:

```
public class PersonBean {
    private String name;
    private int age;

    public PersonBean() {

    }

    // Getter method for name
    public String getName() {
        return name;
    }

    // Setter method for name
    public void setName(String name) {
        this.name = name;
    }

    // Getter method for age
    public int getAge() {
        return age;
    }

    // Setter method for age
    public void setAge(int age) {
        this.age = age;
    }
}

// Main.java
public class Main {
    public static void main(String[] args) {
        // Create an instance of PersonBean
        PersonBean person = new PersonBean();


        // Set properties using setter methods
        person.setName("John");
        person.setAge(30);

        // Get properties using getter methods
        String name = person.getName();
```

```
int age = person.getAge();

// Print the values
System.out.println("Name: " + name);
System.out.println("Age: " + age);
    }
}
```

OUTPUT:

A dark-themed terminal window showing the output of the Java program. The first line is "Name: John" and the second line is "Age: 30".

```
Name: John
Age: 30
```

PROGRAM-13

AIM: Illustrate use of encapsulation in java beans.

OVERVIEW: Encapsulation in JavaBeans involves concealing the internal state of objects by declaring fields as private and providing public methods to access or modify them. This safeguards data integrity and controls access to properties, promoting modular design and facilitating reusable software components with well-defined interfaces and controlled behavior.

SOURCE CODE:

```
// PersonBean.java
public class PersonBean {
    // Private fields
    private String name;
    private int age;

    // Getter method for name
    public String getName() {
        return name;
    }

    // Setter method for name
    public void setName(String name) {
        // Encapsulation: Validation logic can be added here
        this.name = name;
    }

    // Getter method for age
    public int getAge() {
        return age;
    }

    // Setter method for age
    public void setAge(int age) {
        // Encapsulation: Validation logic can be added here
        this.age = age;
    }
}

// Main.java
public class Main {
    public static void main(String[] args) {
        // Create an instance of PersonBean
        PersonBean person = new PersonBean();


        // Set properties using setter methods
        person.setName("John");
        person.setAge(30);

        // Get properties using getter methods
        String name = person.getName();
        int age = person.getAge();
    }
}
```

```
// Print the values
System.out.println("Name: " + name);
System.out.println("Age: " + age);

}
}
```

OUTPUT:



```
Name: John
Age: 30
```

PROGRAM-14

AIM: Write a Java program to insert data into a table using JSP.

OVERVIEW: In JSP, data insertion into a database table typically involves accepting user input through HTML forms, processing it using JSP, and executing SQL queries to insert the data into the database. This process integrates presentation (HTML forms) with logic (JSP) and data manipulation (SQL), enabling dynamic and interactive web applications with data persistence capabilities.

SOURCE CODE:

```
<% @ page import="java.sql.*" %>
<% @ page import="javax.sql.DataSource" %>
<% @ page import="javax.naming.Context" %>
<% @ page import="javax.naming.InitialContext" %>

<%
    // Initialize variables
    String name = request.getParameter("name");
    int age = Integer.parseInt(request.getParameter("age"));

    Connection conn = null;
    PreparedStatement pstmt = null;

    try {
        // Obtain DataSource from JNDI (Java Naming and Directory Interface)
        Context initContext = new InitialContext();
        Context envContext = (Context)initContext.lookup("java:/comp/env");
        DataSource dataSource = (DataSource)envContext.lookup("jdbc/myDataSource");

        // Establish database connection
        conn = dataSource.getConnection();

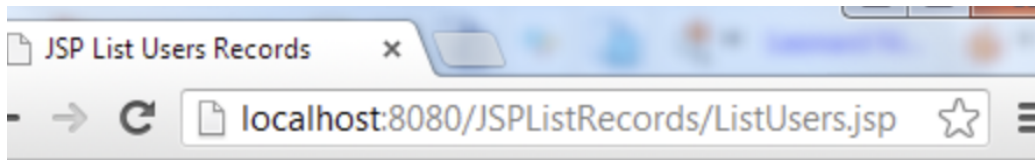
        // Prepare SQL statement
        String sql = "INSERT INTO my_table (name, age) VALUES (?, ?)";
        pstmt = conn.prepareStatement(sql);
        pstmt.setString(1, name);
        pstmt.setInt(2, age);

        // Execute the insert statement
        int rowsAffected = pstmt.executeUpdate();

        // Check if insertion was successful
        if (rowsAffected > 0) {
            out.println("Data inserted successfully!");
        } else {
            out.println("Failed to insert data.");
        }
    } catch (Exception e) {
        out.println("Error: " + e.getMessage());
    } finally {
        // Close resources
        if (pstmt != null) pstmt.close();
```

```
        if (conn != null) conn.close();
    }
%>
```

OUTPUT:



List of users			
ID	Name	Email	Profession
1	Peter	peter@gmail.com	Programmer
2	Tom	tom@hotmail.com	Developer
3	Sam	sam@yahoo.com	Consultant
4	David	david@gmail.com	Designer

PROGRAM-15

AIM: Implement Regular Expressions validation before submitting data in JSP.

OVERVIEW: In JSP, implementing Regular Expressions (regex) validation before submitting data involves validating user input against predefined patterns to ensure it matches specific criteria (e.g., email format, password strength). This enhances data integrity and security by preventing invalid or malicious inputs. Regex validation in JSP typically occurs on the client-side using JavaScript or on the server-side using Java.

SOURCE CODE:

```
<!DOCTYPE html>
<html>
<head>
  <title>Regex Validation in JSP</title>
</head>
<body>
  <h2>Regex Validation in JSP</h2>
  <form action="process" method="post">
    Name: <input type="text" name="name" required><br>
    Age: <input type="text" name="age" required><br>
    <input type="submit" value="Submit">
  </form>
</body>
</html>
```

```
import java.io.IOException;
import java.util.regex.Pattern;
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("/process")
public class ProcessServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // Retrieve form data
        String name = request.getParameter("name");
        String ageStr = request.getParameter("age");

        // Regular expression patterns
        Pattern namePattern = Pattern.compile("[a-zA-Z\\s]+$");
        Pattern agePattern = Pattern.compile("\\d+$");

        // Perform validation
        if (!namePattern.matcher(name).matches()) {
            response.getWriter().println("Invalid name. Please enter only letters and spaces.");
            return;
        }
    }
}
```

```

if (!agePattern.matcher(ageStr).matches()) {
    response.getWriter().println("Invalid age. Please enter only positive integers.");
    return;
}

// Convert age to integer
int age = Integer.parseInt(ageStr);

response.getWriter().println("Data submitted successfully!");
}
}

```

OUTPUT:

The screenshot shows a web page titled "Email Registration" with a light blue background. In the center is a "Registration Form" box. The form contains the following fields and elements:

- First Name: "Sagar"
- Last Name: "Sumit"
- Email: "s@123@4@gmail.com" (This field is highlighted with a red border, and a validation error message points to it.)
- Password: "****" (masked)
- Confirm Password: "7565678798"
- Gender: "male" (selected from a dropdown menu)
- Submit Button: "create"

A yellow warning icon with an exclamation mark is shown next to the error message: "A part following '@' should not contain the symbol '@'." This message is displayed in a white box with a light blue border, pointing to the email field.

EXPERIMENT 16

Aim: Write a Java program to show user validation using Servlet.

Overview: Servlets are Java classes that are used to extend the capabilities of servers that host applications accessed by means of a request-response programming model. Servlets handle requests coming from clients, process them, and then generate responses that are sent back to the client. They are particularly used for creating dynamic web pages and web applications. User validation in Servlets typically involves verifying the credentials provided by the user, such as username and password, against some stored data (e.g., in a database or text file). If the credentials are valid, the user is granted access; otherwise, access is denied.

Source Code:

```
import java.io.IOException;

import java.io.PrintWriter;

import java.util.HashMap;

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 LoginServlet extends HttpServlet {

    private static final long serialVersionUID = 1L;

    private HashMap<String, String> users = new HashMap<>();

    public LoginServlet() {

        super();

        users.put("user1", "password1");

        users.put("user2", "password2");

    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)

        throws ServletException, IOException {

        String username = request.getParameter("username");
```

```

String password = request.getParameter("password");

if (users.containsKey(username) && users.get(username).equals(password)) {

    response.setContentType("text/html");

    PrintWriter out = response.getWriter();

    out.println("<html><body>");

    out.println("<h2>Welcome, " + username + "!</h2>");

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

} else {

    response.setContentType("text/html");

    PrintWriter out = response.getWriter();

    out.println("<html><body>");

    out.println("<h2>Invalid username or password.</h2>");

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

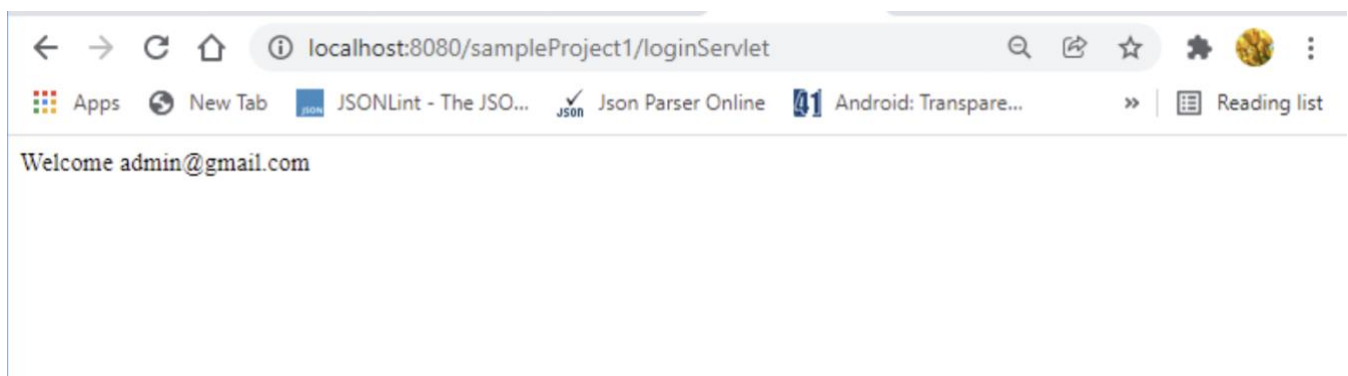
}

}

}

```

OUTPUT:



EXPERIMENT 17

Aim: Write a program to set cookie information using Servlet.

Overview: Cookies: Cookies are small pieces of data that a web server sends to the user's web browser. The browser stores this data and sends it back with subsequent requests to the same server. Cookies are commonly used for session management, user tracking, and personalization.

Types of Cookies:

Session Cookies: These cookies are stored temporarily in the browser's memory and are deleted when the browser is closed.

Persistent Cookies: These cookies are stored on the user's device and remain there until they expire or are deleted by the user.

Working with Cookies in Servlets:

Servlets can create and manipulate cookies using the `javax.servlet.http.Cookie` class. Servlets can set cookies by adding them to the response object, and retrieve cookies from the request object.

Source Code:

```
import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.Cookie;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

@WebServlet("/SetCookieServlet")

public class SetCookieServlet extends HttpServlet {

    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse response)

        throws ServletException, IOException {

        Cookie cookie = new Cookie("username", "Ravi Malik ");

        cookie.setMaxAge(3600); // Cookie will expire in 1 hour (3600 seconds)

        cookie.setPath("/"); // Cookie is valid for the entire application
```

```
response.addCookie(cookie);

response.setContentType("text/html");

response.getWriter().println("<html><body>");

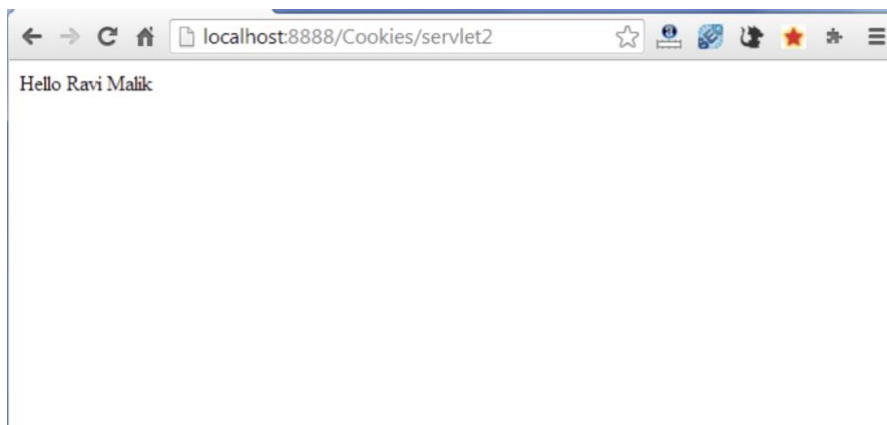
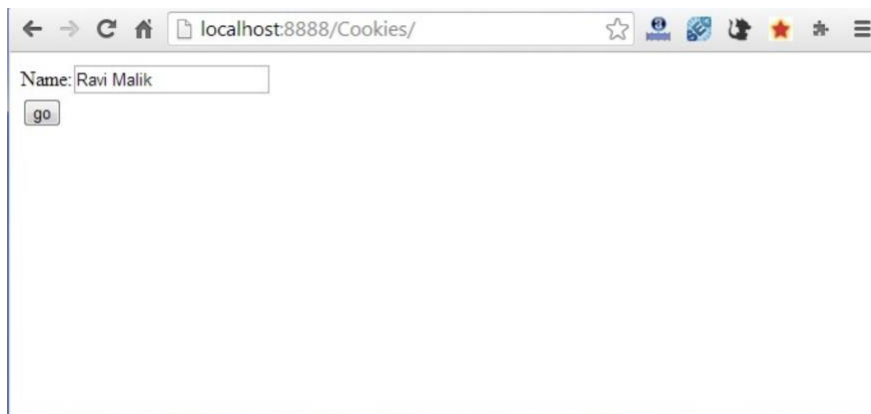
response.getWriter().println("<h2>Hello</h2>");

response.getWriter().println("</body></html>");

}

}
```

OUTPUT:



EXPERIMENT 18

Aim: Design Servlet Login and Logout using Cookies.

Overview: Cookies: Cookies are small pieces of data that a web server sends to the user's web browser. The browser stores this data and sends it back with subsequent requests to the same server. Cookies are commonly used for session management, user tracking, and personalization.

Types of Cookies:

Session Cookies: These cookies are stored temporarily in the browser's memory and are deleted when the browser is closed.

Persistent Cookies: These cookies are stored on the user's device and remain there until they expire or are deleted by the user.

Working with Cookies in Servlets:

Servlets can create and manipulate cookies using the `javax.servlet.http.Cookie` class. Servlets can set cookies by adding them to the response object, and retrieve cookies from the request object.

Source Code:

```
<!DOCTYPE html>

<html>

<head>

<meta charset="ISO-8859-1">

<title>Servlet Login Example</title>

</head>

<body>

<h1>Welcome to Login App by Cookie</h1>

<a href="login.html">Login</a>|

<a href="LogoutServlet">Logout</a>|

<a href="ProfileServlet">Profile</a>

</body>

</html>
```

File: link.html

```
<a href="login.html">Login</a> |
```

`Logout |`

`Profile`

`<hr>`

File: login.html

`<form action="LoginServlet" method="post">`

Name:`<input type="text" name="name">
`

Password:`<input type="password" name="password">
`

`<input type="submit" value="login">`

`</form>`

File: LoginServlet.java

`import java.io.IOException;`

`import java.io.PrintWriter;`

`import javax.servlet.ServletException;`

`import javax.servlet.http.Cookie;`

`import javax.servlet.http.HttpServlet;`

`import javax.servlet.http.HttpServletRequest;`

`import javax.servlet.http.HttpServletResponse;`

`public class LoginServlet extends HttpServlet {`

`protected void doPost(HttpServletRequest request, HttpServletResponse response)`

`throws ServletException, IOException {`

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

`PrintWriter out=response.getWriter();`

`request.getRequestDispatcher("link.html").include(request, response);`

`String name=request.getParameter("name");`

`String password=request.getParameter("password");`

```

if(password.equals("admin123")){

    out.print("You are successfully logged in!");

    out.print("<br>Welcome, "+name);

    Cookie ck=new Cookie("name",name);

    response.addCookie(ck);

}else{

    out.print("sorry, username or password error!");

    request.getRequestDispatcher("login.html").include(request, response);

}

out.close();

}

}

```

File: LogoutServlet.java

```

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.http.Cookie;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class LogoutServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response)

        throws ServletException, IOException {

        response.setContentType("text/html");

        PrintWriter out=response.getWriter();

```

```

        request.getRequestDispatcher("link.html").include(request, response);

        Cookie ck=new Cookie("name","");

        ck.setMaxAge(0);

        response.addCookie(ck);

        out.print("you are successfully logged out!");

    }

}

```

File: ProfileServlet.java

```

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.http.Cookie;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class ProfileServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response)

        throws ServletException, IOException {

        response.setContentType("text/html");

        PrintWriter out=response.getWriter();

        request.getRequestDispatcher("link.html").include(request, response);

        Cookie ck[]=request.getCookies();

        if(ck!=null){

            String name=ck[0].getValue();

```



```

if(!name.equals("")||name!=null){

    out.print("<b>Welcome to Profile</b>");

    out.print("<br>Welcome, "+name);

}

}else{

    out.print("Please login first");

    request.getRequestDispatcher("login.html").include(request, response);

}

out.close();

}

}

```

File: web.xml

```

<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee

http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">

    <servlet>

        <description></description>

        <display-name>LoginServlet</display-name>

        <servlet-name>LoginServlet</servlet-name>

        <servlet-class>com.javatpoint.LoginServlet</servlet-class>

    </servlet>

    <servlet-mapping>

        <servlet-name>LoginServlet</servlet-name>

        <url-pattern>/LoginServlet</url-pattern>

    </servlet-mapping>

```

```
<servlet>

  <description></description>

  <display-name>ProfileServlet</display-name>

  <servlet-name>ProfileServlet</servlet-name>

  <servlet-class>com.javatpoint.ProfileServlet</servlet-class>

</servlet>

<servlet-mapping>

  <servlet-name>ProfileServlet</servlet-name>

  <url-pattern>/ProfileServlet</url-pattern>

</servlet-mapping>

<servlet>

  <description></description>

  <display-name>LogoutServlet</display-name>

  <servlet-name>LogoutServlet</servlet-name>

  <servlet-class>com.javatpoint.LogoutServlet</servlet-class>

</servlet>

<servlet-mapping>

  <servlet-name>LogoutServlet</servlet-name>

  <url-pattern>/LogoutServlet</url-pattern>

</servlet-mapping>

</web-app>
```

OUTPUT:



[Login](#) | [Logout](#) | [Profile](#)

Please login first

Name:

Password:

[Login](#) | [Logout](#) | [Profile](#)

You are successfully logged in!

Welcome, sonoo

[Login](#) | [Logout](#) | [Profile](#)

you are successfully logged out!

EXPERIMENT 19

Aim: Create a servlet that recognizes a visitor for the first time to a web application and responds by saying “Welcome, you are visiting for the first time”. When the page is visited for the second time, it should say “Welcome Back”.

Overview: Servlet Implementation:

1. First Visit Detection:

The servlet checks if the client has a cookie named "visited".

If the "visited" cookie doesn't exist, it indicates that it's the visitor's first visit.

2. Cookie Setting:

If it's the first visit, the servlet sets a cookie named "visited" to indicate that the client has visited the site.

3. Response Generation:

Based on whether the "visited" cookie exists, the servlet generates different responses:

If the "visited" cookie doesn't exist, it displays "Welcome, you are visiting for the first time!".

If the "visited" cookie exists, it displays "Welcome Back!".

Source Code:

```
import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.Cookie;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

@WebServlet("/WelcomeServlet")

public class WelcomeServlet extends HttpServlet {

    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse response)

        throws ServletException, IOException {
```

```

Cookie[] cookies = request.getCookies();

boolean visitedBefore = false;

if (cookies != null) {

    for (Cookie cookie : cookies) {

        if (cookie.getName().equals("visited")) {

            visitedBefore = true;

            break;

        }

    }

}

if (!visitedBefore) {

    Cookie cookie = new Cookie("visited", "true");

    response.getWriter().println("<h2>Welcome, you are visiting for the first time!</h2>");

} else {

    response.getWriter().println("<h2>Welcome Back!</h2>");

}

```

OUTPUT:



EXPERIMENT 20

Aim: Develop a small web program using Servlets, JSPs with Database connectivity.

Overview: Set up the database:

1. Create a MySQL database named user_registration.
2. Create a table named users with columns id (auto-increment), username, and password.

Create the Java Servlet:

1. Create a servlet named RegistrationServlet to handle user registration.
2. Inside the servlet, retrieve the username and password from the request parameters, validate the input, and insert the user data into the database.

Create the JSP file:

1. Create a JSP file named registration.jsp to display a registration form.
2. The form should allow users to enter a username and password.
3. Upon submission, the form should send the data to the RegistrationServlet.

Create another Servlet:

1. Create a servlet named UserListServlet to retrieve the list of registered users from the database.
2. Inside the servlet, query the database for user data and forward the results to a JSP for display.

Create a JSP file to display the list of users:

1. Create a JSP file named userlist.jsp to display the list of registered users.
2. Iterate over the user data received from the UserListServlet and display it in a tabular format.

Configure deployment descriptor:

1. Update the web.xml deployment descriptor to map servlets to their respective URLs.

Source Code:

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
  </head>
  <body>
    <form action="StudentSer" method="post">
      <input type="text" name="txtrno" placeholder="Enter rno" /> <br><br>
      <input type="text" name="txtname" placeholder="Enter name" /> <br><br>
      <input type="text" name="txtbranch" placeholder="Enter branch" /> <br><br>
      <input type="text" name="txtfees" placeholder="Enter fees" /> <br><br>
      <input type="submit" name="btnsubmit" value="Insert" /> <br><br>
    </form>
    <a href="viewstudent.jsp">VIEW STUDENT RECORD HERE</a>
  </body>
</html>
```

Servlet Code:-

```
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.DriverManager;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.sql.*;
```

```

public class StudentSer extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        try {
            PrintWriter out = response.getWriter();
            Class.forName("com.mysql.jdbc.Driver");
            Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/stuinfo","root","");
            Statement st = con.createStatement();
            int x = st.executeUpdate("insert into student(rno,name,branch,fees)
values('"+request.getParameter("txtrno")+"','"+request.getParameter("txtname")+"','"+request.getParam
eter("txtbranch")+"','"+request.getParameter("txtfees")+"')");
            if(x!=0)
            {
                out.print("Data Inserted Successfully");
            }

        } catch (Exception ex) {

        }
    }
}

```

OUTPUT:

Fill in your details

ID:

Name:

Email Id:

Phone Number:

Query Editor

Query History

```

1 SELECT * FROM public.studentdetails
2 ORDER BY stuid ASC

```

Data Output

Explain

Messages

Notifications

	stuid [PK] numeric	stuname character varying (30)	email character varying (40)	phonenumber character varying (10)
1	107952	Geeks	Geeksforgeeks@gmail.com	9876543221
2	107963	Java_Servlets	Java_programming@gmail.c...	7899766555

