

Bharat College Of Engineering,
Badlapur (W), Thane
Affiliated To Mumbai University



LAB MANUAL

[SUBJECT NAME:FULL STACK JAVA]

[SUBJECT CODE: 2343611]

[CLASS: SECOND YEAR][SEMESTER: III]

FACULTY NAME: Prof.RUPALI JADHAV

Lab Objectives: This subject seeks to give students an understanding of full stack development in Java.

The main aim of this course is to:

1. Familiarize with Basic OOP concepts in Java,
2. Understand the concepts of inheritance and exceptions in java,
3. Design and implement programs involving Client and Server Side Programming,
4. Describe and utilize the functioning of DOM and Java script,
5. Study different design patterns in web programming and understand the working of react framework.
6. To describe the Spring Framework and implement the related case studies.

.

Lab Outcomes:

At the end of the course, the students should be able to:

1. Understand and apply the fundamentals of Java Programming and Object-Oriented Programming,
2. Analyze and Illustrate Inheritance and Exception Handling Mechanisms,
3. Elaborate and design applications using Client and Server Side Programming,
4. Understand the concepts in JavaScript for interactive Web Development,
5. Implement the real-world application development in web programming using React,
6. Design and Develop Enterprise-Level Applications Using the Spring Framework

LIST OF EXPERIMENT

Sr.No	Title of Experiments	Hrs
1	Programs on classes and objects	2
2	Programs on method and constructor overloading.	2
3	Programs on various types of inheritance and Exception handling	2
4	Program on Implementing Generic and HTTP servlet.	2
5	Design a login webpage in JSP that makes validation through Database using JDBC and call the servlet for various operations	2
6	Program on Implicit and Explicit objects in JSP	2
7	Program to create a website using HTML CSS and JavaScript	2
8	Program using Java Script to validate the email address entered by the user (check the presence of “@” & “.” character. If this character is missing, the script should display an alert box reporting the error and ask the user to re-enter it again).	2
9	Program based on Document Object Model to change background color of the web page automatically after every 5 seconds.	2
10	Program for making use of React Hooks that displays four buttons namely, “Red”, “Blue”, “Green”, “Yellow”. On clicking any of these buttons, the code displays the message that you have selected that particular color.	2
11	Creating a Single Page website using the concepts in React like Hooks, Router, Props and States.	2
12	Program to create a Monolithic Application using SpringBoot.	2
13	Program for Building RESTful APIs with spring boot.	2

Experiment 1

AIM: Programs on classes and objects.

THEORY:

Class : A class is a template to create objects having similar properties and behavior, or in other words, we can say that a class is a blueprint for objects.

Class Declaration in Java

```
access_modifier class<class_name>
{
    data member;
    method;
    constructor;
    nested class;
    interface ;
}
```

Components of Java Classes:

In general, class declarations can include these components, in order:

- **Modifiers:** A class can be public or has default access.
- **Class keyword:** Class keyword is used to create a class.
- **Class name:** The name should begin with an initial letter (capitalized).
- **Superclass (if any):** The name of the class's parent (superclass), if any, preceded by the keyword `extends`. A class can only extend (subclass) one parent.
- **Interfaces(if any):** A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword `implements`. A class can implement more than one interface.
- **Body:** The class body is surrounded by braces, { }.

- Fields: Fields are variables defined within a class that hold the data or state of an object.
- Constructors: It is a special method in a class that is automatically called when an object is created. And its name is same as class name.

Java Objects:

Objects are the instances of a class that are created to use the attributes and methods of a class. A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of:

- State: It is represented by attributes of an object. It also reflects the properties of an object.
- Behavior: It is represented by the methods of an object. It also reflects the response of an object with other objects.
- Identity: It gives a unique name to an object and enables one object to interact with other objects.

PROGRAM:

```
class Student
{
    int id;
    String n;
    // Added constructor to initialize both fields
    public Student(int id, String n)
    {
        this.id = id;
        this.n = n;
    }
}

public class Main // File: Main.java
{
    public static void main(String[] args)
```

```
{  
    // Creating Student object using the new constructor  
    Student s1 = new Student(10, "Alice");  
    System.out.println(s1.id);  
    System.out.println(s1.n);  
}  
}
```

OUTPUT:

10
Alice

Experiment 2

AIM: Programs on method and constructor overloading

THEORY:

Method Overloading: Method Overloading allows us to define multiple methods with the same name but different parameters within a class. This difference may include:

- The number of parameters
- The types of parameters
- The order of parameters

Method overloading in Java is also known as Compile-Time Polymorphism, Static Polymorphism, or Early Binding, because the decision about which method to call is made at compile time.

Key features of Method Overloading

- Multiple methods can share the same name in a class when their parameter lists are different.
- Overloading is a way to increase flexibility and improve the readability of code.
- Overloading does not depend on the return type of the method, two methods cannot be overloaded by just changing the return type.

```
int sum(int x, int y)
{
    return (x + y);
}
int sum(int x, int y, int z)
{
    return (x + y + z);
}
double sum(double x, double y)
{
    return (x + y);
}
```

Constructor Overloading:

A constructor is a special block of code that is called when an object is created. Its main job is to initialize the object, to set up its internal state, or to assign default values to its attributes.

Characteristics of Constructors:

- **Same Name as the Class:** A constructor has the same name as the class in which it is defined.
- **No Return Type:** Constructors do not have any return type, not even void. The main purpose of a constructor is to initialize the object, not to return a value.
- **Automatically Called on Object Creation:** When an object of a class is created, the constructor is called automatically to initialize the object's attributes.
- **Used to Set Initial Values for Object Attributes:** Constructors are primarily used to set the initial state or values of an object's attributes when it is created.

Sometimes there is a need of initializing an object in different ways. This can be done using constructor overloading.

For example, the Thread class has 8 types of constructors. If we do not want to specify anything about a thread then we can simply use the default constructor of the Thread class, however, if we need to specify the thread name, then we may call the parameterized constructor of the Thread class with a String args like this:

```
Thread t= new Thread (" MyThread ");
```

PROGRAM:

Method Overloading:

```
class Calculator {  
    // Method with 2 int parameters  
    int add(int a, int b) {
```



```

        return a + b;
    }
    // Method with 3 int parameters (Overloaded)
    int add(int a, int b, int c) {
        return a + b + c;
    }
    // Method with 2 double parameters (Overloaded)
    double add(double a, double b) {
        return a + b;
    }
    // Method with different order of parameters (Overloaded)
    String add(String a, int b) {
        return a + b;
    }
}
public class MethodOverloadingDemo
{
    public static void main(String[] args)
    {
        Calculator calc = new Calculator();
        System.out.println("Sum of 2 integers: " + calc.add(10, 20));
        System.out.println("Sum of 3 integers: " + calc.add(5, 15, 25));
        System.out.println("Sum of 2 doubles: " + calc.add(12.5, 7.5));
        System.out.println("String and int:"+calc.add("Number:",50));
    }
}

```

OUTPUT:

```

Sum of2 integers: 30
Sum of 3 integers: 45
Sum of 2 doubles: 20.0
String and int: Number: 50
=== Code Execution Successful ===

```

Constructor Overloading:

```
class Box {
    double width, height, depth;

    // constructor used when all dimensions
    // specified
    Box(double w, double h, double d)
    {
        width = w;
        height = h;
        depth = d;
    }

    // constructor used when no dimensions
    // specified
    Box() { width = height = depth = 0; }

    // constructor used when cube is created
    Box(double len) { width = height = depth = len; }

    // compute and return volume
    double volume() { return width * height * depth; }
}

// Driver code
public class Test {
    public static void main(String args[])
    {
        // create boxes using the various
        // constructors
        Box mybox1 = new Box(10, 20, 15);
        Box mybox2 = new Box();
        Box mycube = new Box(7);

        double vol;
```

```
// get volume of first box
vol = mybox1.volume();
System.out.println("Volume of mybox1 is " + vol);

// get volume of second box
vol = mybox2.volume();
System.out.println("Volume of mybox2 is " + vol);

// get volume of cube
vol = mycube.volume();
System.out.println("Volume of mycube is " + vol);
    }
}
```

OUTPUT

```
Volume of mybox1 is 3000.0
Volume of mybox2 is 0.0
Volume of mycube is 343.0
```

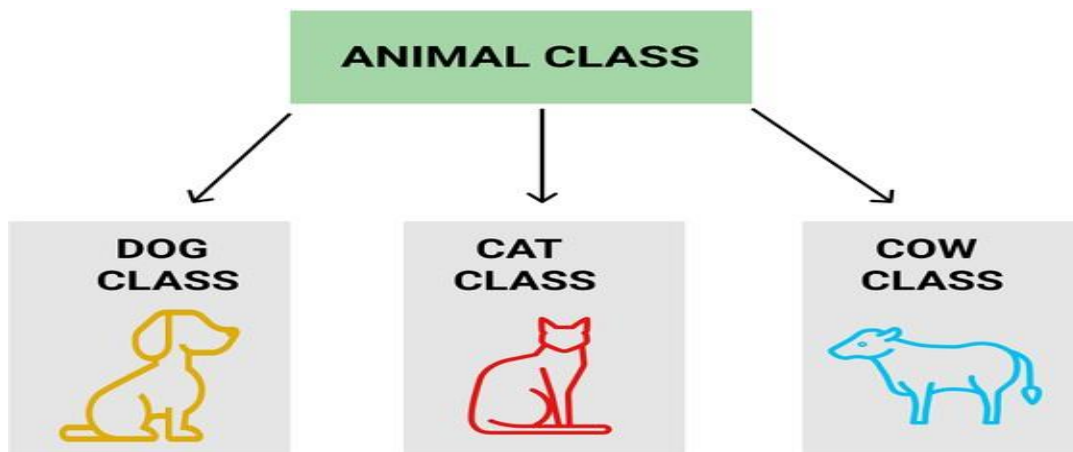
Experiment No-3

AIM: Programs on various types of inheritance and Exception handling.

THEORY:

Inheritance in Java:

Java Inheritance is a fundamental concept in OOP(Object Oriented Programming). It is the mechanism in Java by which one class is allowed to inherit the features(fields and methods) of another class. In Java, Inheritance means creating new classes based on existing ones. A class that inherits from another class can reuse the methods and fields of that class.



Syntax:

```
class ChildClass extends ParentClass {  
  
    // Additional fields and methods  
  
}
```

Why Use Inheritance in Java?

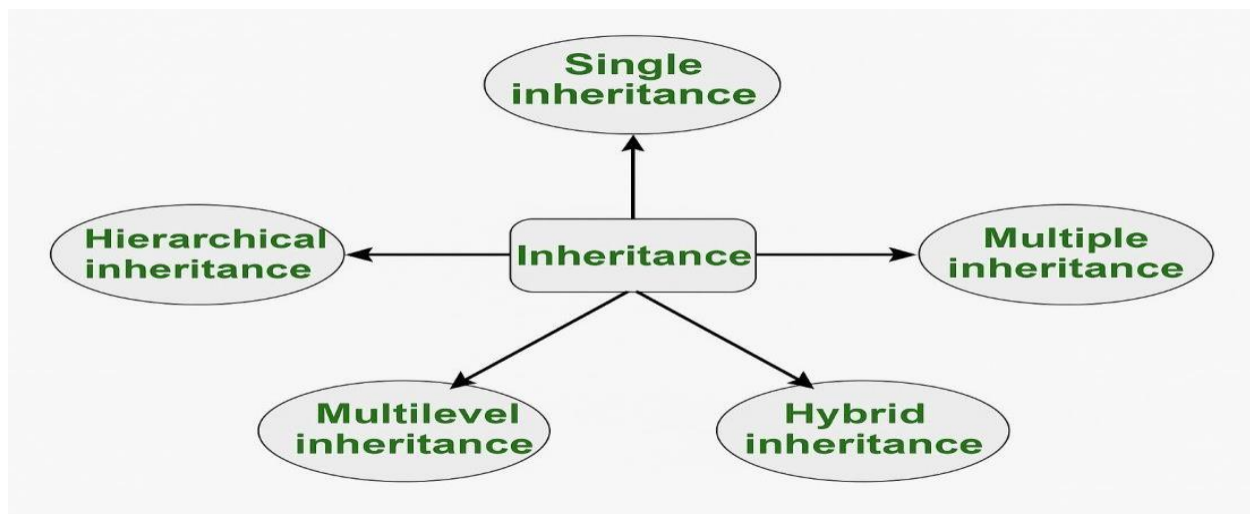
- **Code Reusability:** The code written in the Superclass is common to all subclasses. Child classes can directly use the parent class code.
- **Method Overriding:** Method Overriding is achievable only through Inheritance. It is one of the ways by which Java achieves Run Time Polymorphism.

- **Abstraction:** The concept of abstraction where we do not have to provide all details, is achieved through inheritance. Abstraction only shows the functionality to the user.

Key Terminologies Used in Java Inheritance:

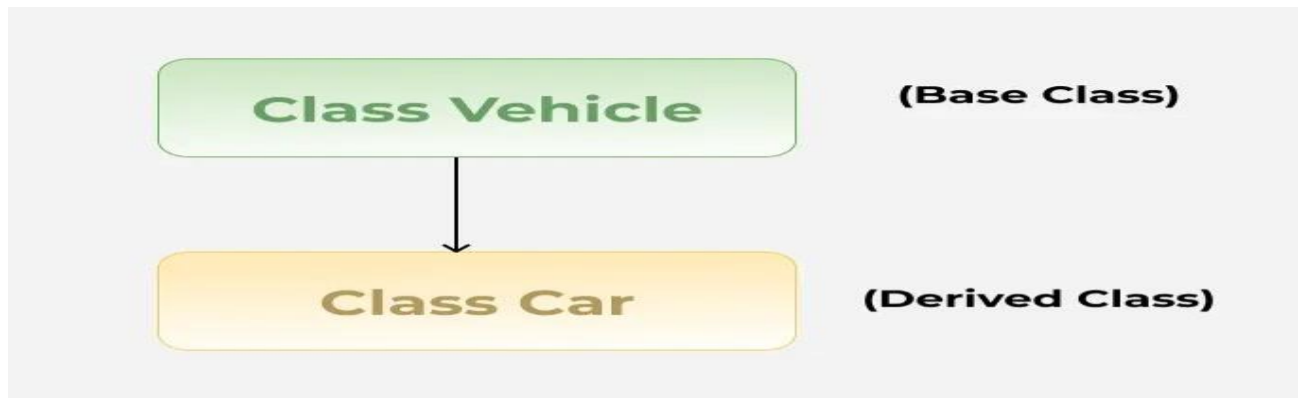
- **Class:** Class is a set of objects that share common characteristics/ behavior and common properties/ attributes. Class is not a real-world entity. It is just a template or blueprint or prototype from which objects are created.
- **Super Class/Parent Class:** The class whose features are inherited is known as a superclass(or a base class or a parent class).
- **Sub Class/Child Class:** The class that inherits the other class is known as a subclass(or a derived class, extended class or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.
- **Extends Keyword:** This keyword is used to inherit properties from a superclass.

Types of Inheritance in Java:



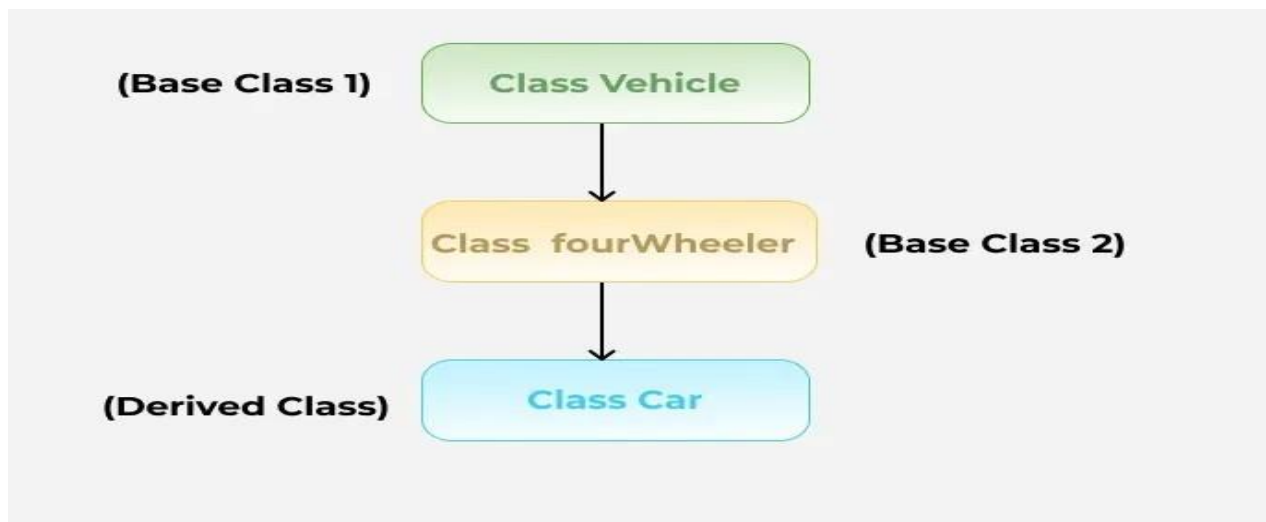
fi. Single Inheritance:

In single inheritance, a sub-class is derived from only one super class. It inherits the properties and behavior of a single-parent class. Sometimes, it is also known as simple inheritance.



2. Multilevel Inheritance:

In Multilevel Inheritance, a derived class will be inheriting a base class and as well as the derived class also acts as the base class for other classes.



Java Exception Handling;

Exception handling in Java is an effective mechanism for managing runtime errors to ensure the application's regular flow is maintained. Some Common examples of

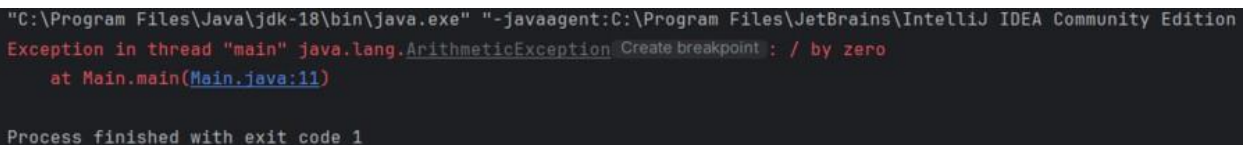
exceptions include `ClassNotFoundException`, `IOException`, `SQLException`, `RemoteException`, etc. By handling these exceptions, Java enables developers to create robust and fault-tolerant applications.

Example: Showing an arithmetic exception or we can say a divide by zero exception.

```
import java.io.*;
class Geeks {
    public static void main(String[] args)
    {
        int n = 10;
        int m = 0;

        int ans = n / m;

        System.out.println("Answer: " + ans);
    }
}
```

A screenshot of a Java IDE console window. The text in the console is as follows: "C:\Program Files\Java\jdk-18\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition\... Exception in thread "main" java.lang.ArithmeticException: / by zero at Main.main(Main.java:11) Process finished with exit code 1. The exception message is highlighted in red.

```
"C:\Program Files\Java\jdk-18\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition
Exception in thread "main" java.lang.ArithmeticException: / by zero
at Main.main(Main.java:11)
Process finished with exit code 1
```

PROGRAM:

Single level inheritance

```
class Person {
    String name;
    int age;
    void acceptDetails(String n, int a)
    {
        name = n;
        age = a;
    }
}
```

```

        void displayDetails() {
            System.out.println("Name: " + name);
            System.out.println("Age: " + age);
        }
    }
    class Student extends Person    // Child class inheriting from Person
    {
        int rollNo;
        String course;
        void acceptStudentDetails(String n, int a, int r, String c) {
            acceptDetails(n, a);
            rollNo = r;
            course = c;
        }
        void displayStudentDetails() {
            // Calling parent method to display name and age
            displayDetails();
            System.out.println("Roll No: " + rollNo);
            System.out.println("Course: " + course);
        }
    }
    public class SingleInheritanceExample {
        public static void main(String[] args) {
            Student s1 = new Student();
            // Set values
            s1.acceptStudentDetails("Deepak", 20, 101, "Computer Science");
            // Display values
            s1.displayStudentDetails();
        }
    }
}

```

OUTPUT:


```
C:\Users\rahul\OneDrive\Documents>java SingleInheritanceExample
Name: Deepak
Age: 20
Roll No: 101
Course: Computer Science
```

Multilevel inheritance:

```
class Person {
    String name;
    int age;
    void acceptDetails(String n, int a) {
        name = n;
        age = a;
    }
    void displayDetails() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }
}
// Child class (Derived from Person)
class Student extends Person {
    String course;
    void setCourse(String c) {
        course = c;
    }
    void showCourse() {
        System.out.println("Course: " + course);
    }
}
// Grandchild class (Derived from Student)
class Exam extends Student {
    int marks;
    void setMarks(int m) {
        marks = m;
    }
    void showResult() {
        System.out.println("Marks: " + marks);
    }
}
```

```

}
// Main class
public class MultilevelInheritanceDemo {
    public static void main(String[] args) {
        Exam e1 = new Exam();

        // Using methods from all levels of inheritance
        e1.acceptDetails("Rohit", 21);    // From Person
        e1.setCourse("Computer Science"); // From Student
        e1.setMarks(85);                  // From Exam
        // Displaying details
        e1.displayDetails();
        e1.showCourse();
        e1.showResult();
    }
}

```

OUTPUT:

```

C:\Users\rahul\OneDrive\Documents>javac MultilevelInheritanceDemo.java

C:\Users\rahul\OneDrive\Documents>java MultilevelInheritanceDemo
Name: Rohit
Age: 21
Course: Computer Science
Marks: 85

```

Experiment No:4

AIM: Program on Implementing Generic and HTTP servlet

THEORY: Java Servlet is a Java program that runs on a Java-enabled web server or application server. It handles client requests, processes them and generates responses dynamically. Servlets are the backbone of many server-side Java applications due to their efficiency and scalability.

Key Features:

- Servlets work on the server side.
- Servlets are capable of handling complex requests obtained from the web server.
- Generate dynamic responses efficiently.

Creating a Basic Servlet

Prerequisites

- [Install JDK \(Java Development Kit\)](#)
- [Install Apache Tomcat server \(e.g., version 9 or 10\)](#)
- [Install an IDE like Eclipse or IntelliJ IDEA](#)
- *Download the Servlet API JAR (already included in Tomcat)*

Step 1: Create a Dynamic Web Project (in Eclipse).

- Open Eclipse -> File -> New -> Dynamic Web Project
- Name the project (e.g., HelloWorldServlet)
- Target runtime -> Select Apache Tomcat
- Click Finish

Step 2: Create Servlet class.

- Right-click on src -> New -> Servlet
- Name it HelloWorldServlet and click Finish

HelloWorldServlet.java

```

import java.io.*;
import jakarta.servlet.*;
import jakarta.servlet.http.*;
public class HelloWorldServlet extends HttpServlet {

protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException
{
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    out.println("<html><body><h1>Hello,World!</h1></body></html>");
    }

}

```

Explanation:

- Above mentions HelloWorldServlet, which is a servlet class that extends HTTP Servlet, allowing it to handle HTTP requests.
- doGet() is used to process GET requests.
- response.setContentType("text/html") tells the browser it is HTML content.
- PrintWriter out = response.getWriter() is used to write the response.
- out. println() sends a simple HTML message: "Hello, World!" to the client.

Configuring a Servlet

To deploy a servlet, you need to configure it in the web.xml file. This file maps URLs to servlets. For example,

```

<web-app
xmlns="http://www.oracle.com/webfolder/technetwork/jsc/xml/ns/javaee/index
.html"
xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.oracle.com/webfolder/technetwork/jsc/xml/ns
/javaee/index.html

http://www.oracle.com/webfolder/technetwork/jsc/xml/ns/javaee/index.html/w
eb-app_3_0.xsd"
version="3.0">

```

```

    <servlet>
        <servlet-name>HelloWorldServlet</servlet-name>
        <servlet-class>HelloWorldServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>HelloWorldServlet</servlet-name>
        <url-pattern>/hello</url-pattern>
    </servlet-mapping>

</web-app>

```

Explanation:

This is a web.xml file used for mapping URLs to servlets. So, when you visit `http://localhost:8080/yourApp/hello`, the servlet runs and shows "Hello, World!" in the browser.

PROGRAM:

```

package cscorner;

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.io.IOException;
import java.io.PrintWriter;

@WebServlet("/ServletDemo")
public class ServletDemo extends HttpServlet {
    private static final long serialVersionUID = 1L;

    public ServletDemo() {
        super();
    }

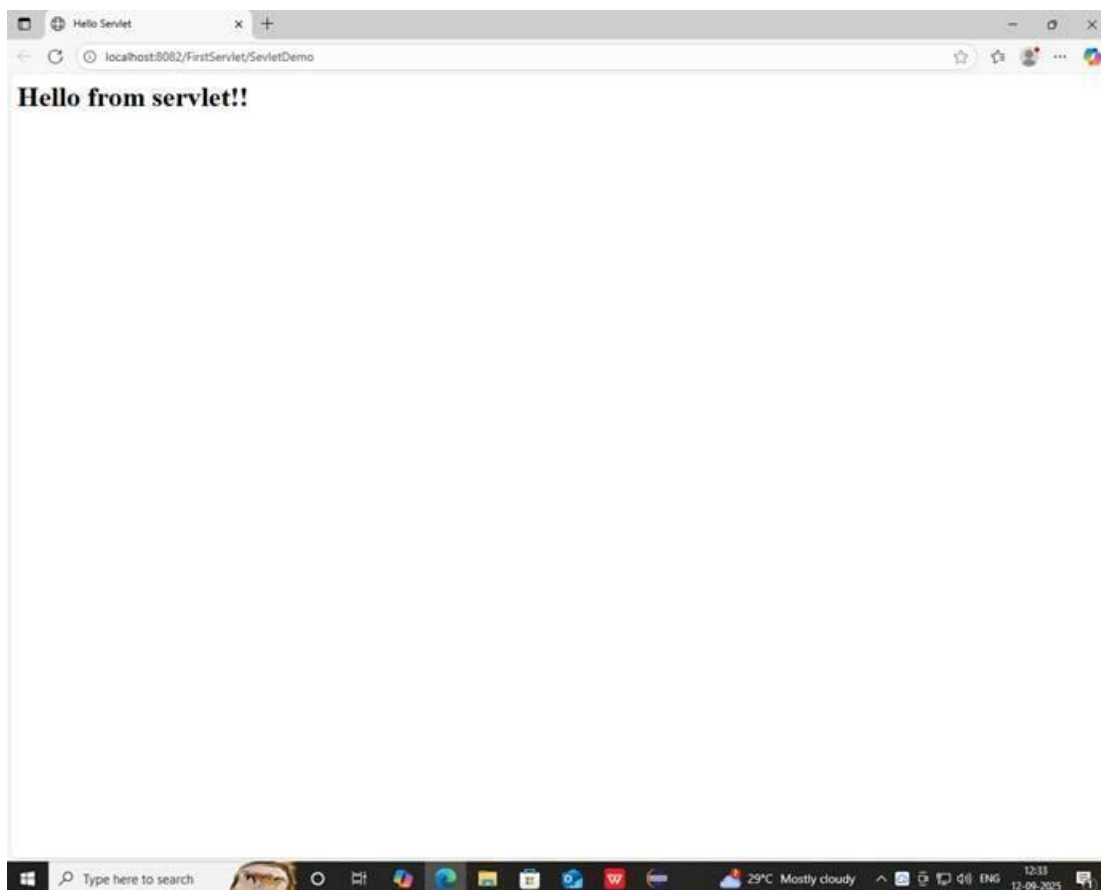
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/HTML");
        PrintWriter pw=response.getWriter();
        pw.println("<html>");
        pw.println("<head><title>Hello Servlet</title></head>");
        pw.println("<body>");
        pw.println("<h1>Hello from Servlet!!</h1>");
    }
}

```

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

```
protected void doPost(HttpServletRequest request, HttpServletResponse  
response) throws ServletException, IOException {  
doGet(request, response);  
}  
  
}
```

Output:



Experiment No:5

AIM: Design a login webpage in JSP that makes validation through Database using JDBC and call the servlet for various operations

THEORY: To create a login webpage using JSP, JDBC, and a servlet for validation, you'll need several components:

1. JSP page to collect the login credentials.
2. JDBC (Java Database Connectivity) to validate the user against the database.
3. Servlet to handle the request, perform database operations, and manage session states.

Step-by-Step Design:

1. Database Setup

Make sure you have a database (e.g., MySQL) with a user table. Here's an example of the user table:

```
CREATE TABLE users (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    username VARCHAR(50) NOT NULL,  
    password VARCHAR(50) NOT NULL  
);
```

Insert sample data into the `users` table:

```
INSERT INTO users (username, password) VALUES ('john_doe',  
'password123');
```

2. JSP Login Page (login.jsp)

This will be the front-end where users input their credentials.

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
```

```

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html lang="en">
<head>    <meta charset="UTF-8">

        <meta name="viewport" content="width=device-width,
initial-scale=1.0">

        <title>Login Page</title>
</head>
<body>
    <h2>Login</h2>
    <form action="LoginServlet" method="post">
        <label for="username">Username:</label>
        <input type="text" id="username" name="username"
required><br><br>
        <label for="password">Password:</label>
        <input type="password" id="password" name="password"
required><br><br>
        <input type="submit" value="Login">
    </form>
    <!-- Display error message -->
    <c:if test="${not empty errorMessage}">
        <p style="color:red">${errorMessage}</p>
    </c:if>
</body>
</html>

```

4. Servlet for Handling Login (LoginServlet.java)

This servlet will process the login form submission, validate credentials through JDBC, and redirect accordingly.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.sql.*;

public class LoginServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        String username = request.getParameter("username");
        String password = request.getParameter("password");

        // Check database for user credentials
        try {
            if (isValidUser(username, password)) {
                // If valid, redirect to the dashboard or main page
                response.sendRedirect("dashboard.jsp");
            } else {
                // If invalid, send error message to login page
                request.setAttribute("errorMessage", "Invalid username
or password");
                RequestDispatcher dispatcher =
request.getRequestDispatcher("login.jsp");
                dispatcher.forward(request, response);
            }
        } catch (SQLException e) {
            e.printStackTrace();
            response.getWriter().write("Error occurred during
validation");
        }
    }

    private boolean isValidUser(String username, String password)
throws SQLException {
        boolean isValid = false;
        Connection conn = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;

        try {
```

```

        conn = DBUtil.getConnection();
        String query = "SELECT * FROM users WHERE username = ? AND
password = ?";
        stmt = conn.prepareStatement(query);
        stmt.setString(1, username);
        stmt.setString(2, password);
        rs = stmt.executeQuery();

        if (rs.next()) {
            isValid = true;
        }
    } finally {
        if (rs != null) {
            rs.close();
        }
        if (stmt != null) {
            stmt.close();
        }
        if (conn != null) {
            conn.close();
        }
    }
    return isValid;
}
}

```

5. web.xml Configuration (web.xml)

You need to define the servlet and map it to the URL pattern. Here's how to configure it in [web.xml](#):

```

<web-app xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_1.xsd" version="3.1">

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

    <servlet-mapping>

```

```

        <servlet-name>LoginServlet</servlet-name>
        <url-pattern>/LoginServlet</url-pattern>
    </servlet-mapping>

    <welcome-file-list>
        <welcome-file>login.jsp</welcome-file>
    </welcome-file-list>

</web-app>

```

6. Dashboard Page (dashboard.jsp)

This page will be shown if the user successfully logs in.

```

<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"
%>

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Welcome to Dashboard</title>
</head>
<body>
    <h2>Welcome to the Dashboard</h2>
    <a href="logout.jsp">Logout</a>
</body>
</html>

```

7. Logout Page (logout.jsp)

To handle user logout, invalidate the session and redirect back to the login page.

```

<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@ page import="javax.servlet.http.*, javax.servlet.*" %>

<%
    HttpSession session = request.getSession(false);

```

```
        if (session != null) {  
            session.invalidate();  
        }  
        response.sendRedirect("login.jsp");  
%>
```

8. Deploy and Test

1. Compile the Java classes and deploy them to a web server like Apache Tomcat.
2. Test the application by navigating to login.jsp.
3. After entering valid credentials, you should be redirected to dashboard.jsp. Invalid credentials will show an error message on the login page.

OUTPUT:

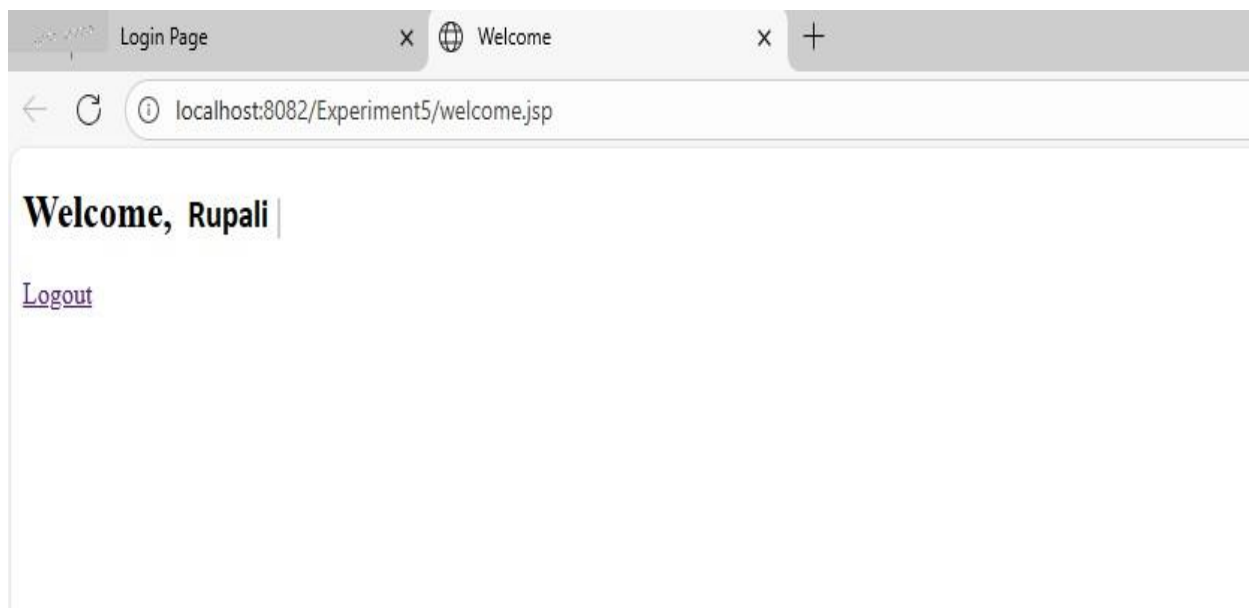


A screenshot of a web browser window. The title bar shows 'Login Page' with a close button and a plus sign for new tabs. The address bar shows 'localhost:8082/Experiment5/login.jsp'. The page content includes the heading 'Login Form', followed by 'Username:' and a text input field, 'Password:' and a text input field, and a 'Login' button at the bottom.

Login Form

Username:

Password:



A screenshot of a web browser window with two tabs: 'Login Page' and 'Welcome'. The 'Welcome' tab is active, showing 'localhost:8082/Experiment5/welcome.jsp' in the address bar. The page content includes the heading 'Welcome, Rupali' and a 'Logout' link.

Welcome, Rupali

[Logout](#)

Experiment No-6

AIM: Program on Implicit and Explicit objects in JSP

THEORY: JSP stands for Java Server Pages, and it's a server side technology. It's used for creating web applications and dynamic web content. The main property of JSP is that we can insert our java code inside our HTML page using the JSP tag. JSP provides you platform-independent pages.

Implicit objects are a set of Java objects that the JSP Container makes available to developers on each page. These objects may be accessed as built-in variables via scripting elements and can also be accessed programmatically by JavaBeans and Servlets. JSP provides you Total 9 implicit objects which are as follows

1. **request:** This is the object of `HttpServletRequest` class associated with the request.
2. **response:** This is the object of `HttpServletResponse` class associated with the response to the client.
3. **config:** This is the object of `ServletConfig` class associated with the page.
4. **application:** This is the object of `ServletContext` class associated with the application context.
5. **session:** This is the object of `HttpSession` class associated with the request.
6. **page context:** This is the object of `PageContext` class that encapsulates the use of server-specific features. This object can be used to find, get or remove an attribute.
7. **page object:** The manner we use the keyword `this` for current object, page object is used to refer to the current translated servlet class.
8. **exception:** The exception object represents all errors and exceptions which is accessed by the respective jsp. The exception implicit object is of type `java.lang.Throwable`.
9. **out:** This is the `PrintWriter` object where methods like `print` and `println` help for displaying the content to the client.

PROGRAM:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Implicit and Explicit Objects in JSP</title>
</head>
<body>
<%@ page import="java.util.Date" %>

<h2>Implicit and Explicit Objects Demo</h2>

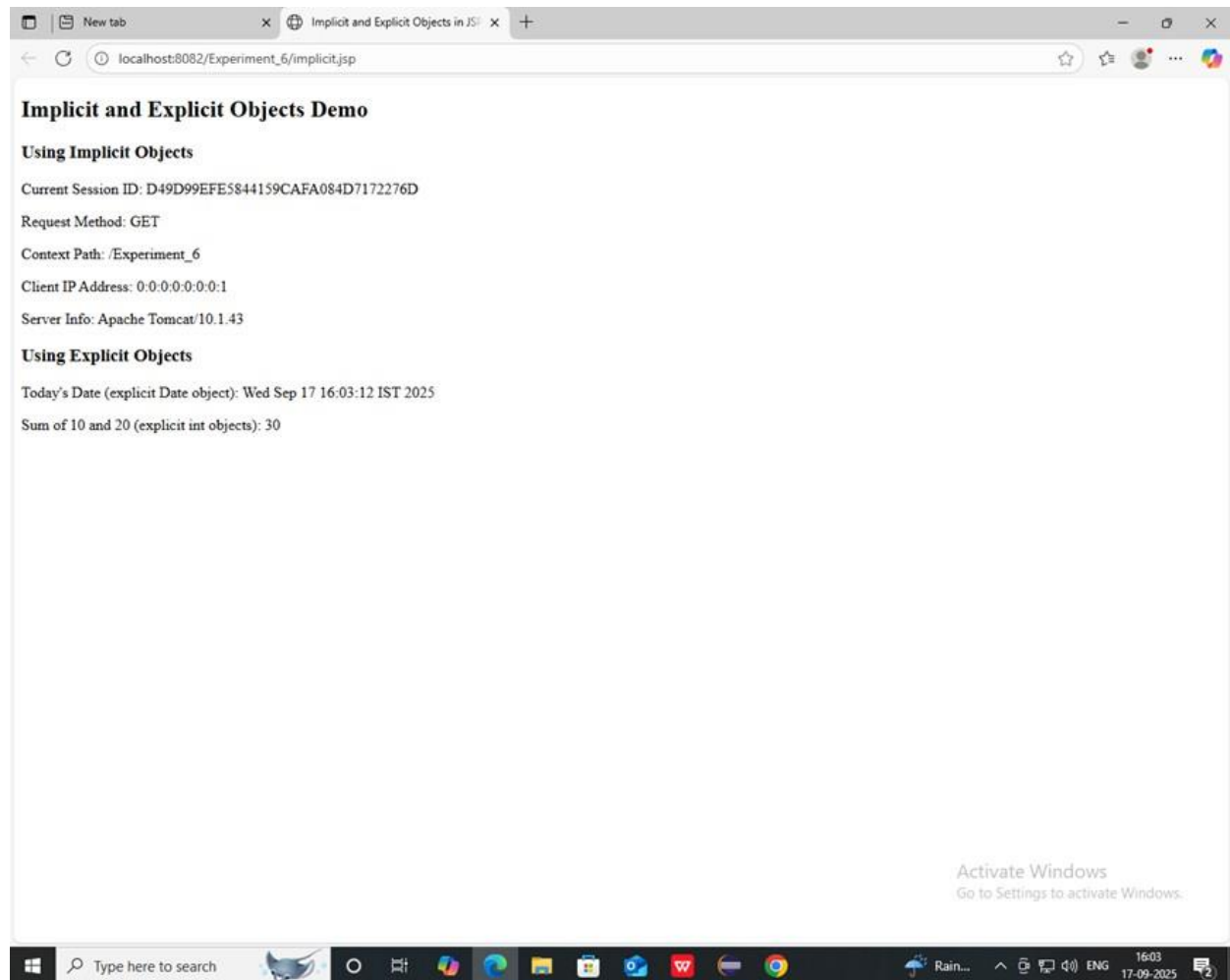
<!-- Using Implicit Objects --%>
<h3>Using Implicit Objects</h3>
<p>Current Session ID: <%= session.getId() %></p>
<p>Request Method: <%= request.getMethod() %></p>
<p>Context Path: <%= application.getContextPath() %></p>
<p>Client IP Address: <%= request.getRemoteAddr() %></p>
<p>Server Info: <%= application.getServerInfo() %></p>

<!-- Using Explicit Object --%>
<h3>Using Explicit Objects</h3>
<%
// Explicit object creation
Date today = new Date();
int a = 10, b = 20;
int sum = a + b;
%>
<p>Today's Date (explicit Date object): <%= today %></p>
<p>Sum of 10 and 20 (explicit int objects): <%= sum %></p>

</body>
```

</html>

OUTPUT:



Experiment No-7

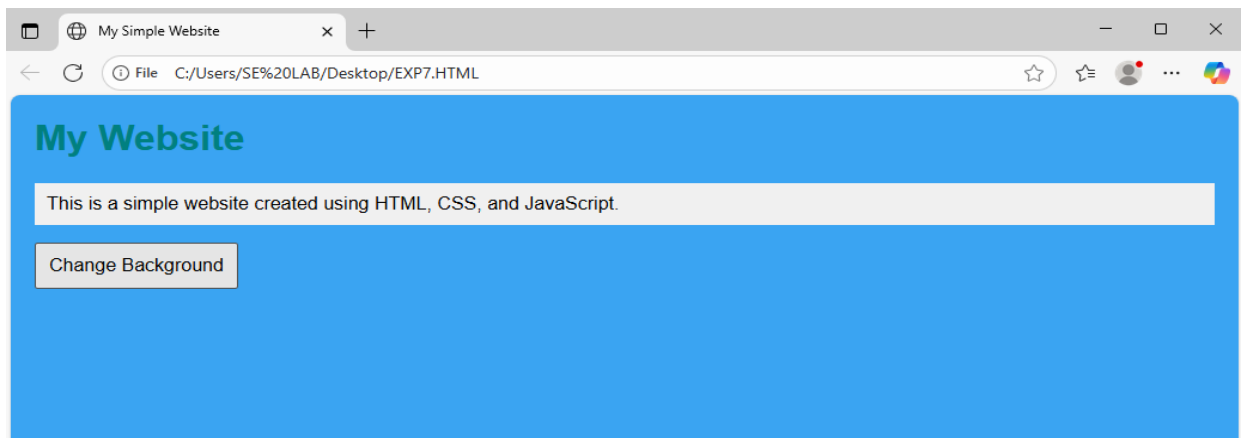
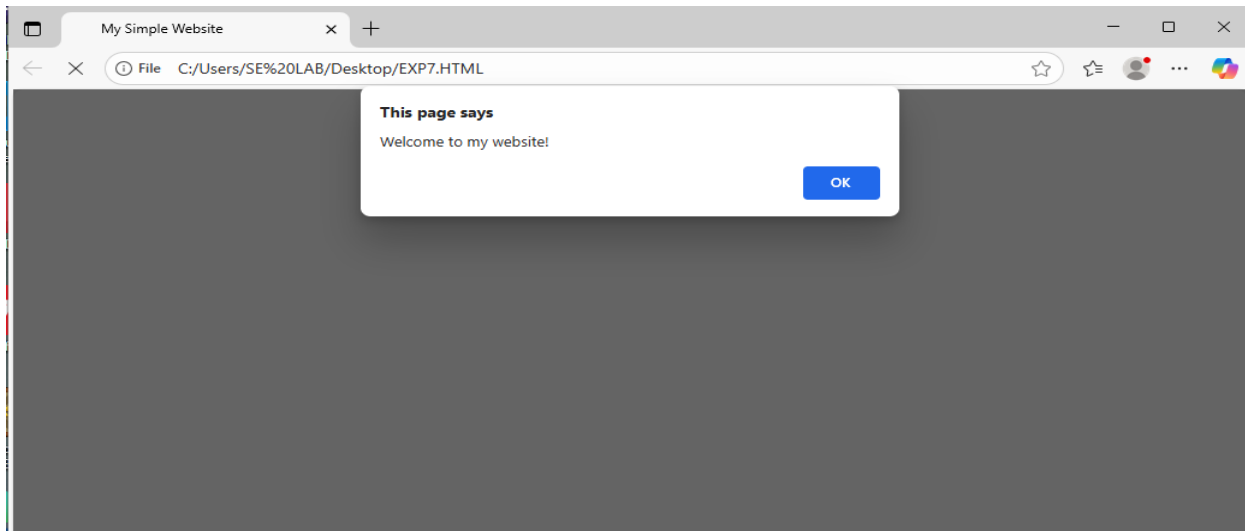
AIM:Program to create a website using HTML CSS and JavaScript

THEORY: A typical web page uses HTML for structure, CSS for styling, and JavaScript for behavior. HTML defines the page structure. The example below creates a simple page with styled heading and a button. The CSS (in a <style> block) sets fonts and colors. JavaScript (<script>) is used to show an alert on load and to change the page's background color when the button is clicked.

PROGRAM:

```
<!DOCTYPE html>
<html>
<head> <title>My Simple Website</title>
  <style>
    body { font-family: Arial, sans-serif; margin: 20px; }
    h1 { color: teal; }
    p { background-color: #f0f0f0; padding: 10px; }
    button { padding: 10px; font-size: 16px; }
  </style>
  <script>
    // Show welcome alert on page load
    window.onload = function() {
      alert("Welcome to my website!");
    }; // Function to change background color randomly
    function changeBackground() {
      var r = Math.floor(Math.random()*256);
      var g = Math.floor(Math.random()*256);
      var b = Math.floor(Math.random()*256);
      document.body.style.backgroundColor = "rgb(" + r + "," + g + "," +
+ b + ")";
    } </script>
</head>
<body>
  <h1>My Website</h1>
  <p>This is a simple website created using HTML, CSS, and
JavaScript.</p>
  <button onclick="changeBackground()">Change Background</button>
</body>
</html>
```

OUTPUT:



Experiment No-8

AIM: Program using Java Script to validate the email address entered by the user (check the presence of “@” & “.” character. If this character is missing, the script should display an alert box reporting the error and ask the user to re-enter it again).

THEORY: In this program, the main goal is to validate the email address entered by the user to ensure it meets basic requirements before submission. Specifically, we are checking for the presence of two critical characters: the "@" symbol and the "." (dot) symbol. These two characters are essential components in most email addresses, and their absence often indicates an invalid email address format.

1. Why Email Validation?

Email validation is crucial to ensure that the data entered by users is in a valid format before it is processed further, such as stored in a database or used in an application. A valid email typically contains:

- "@": Separates the local part (username) from the domain.
- ".": Appears after the "@" symbol to separate the domain name from the top-level domain (like [.com](#), [.org](#), etc.).

Example of a valid email: [john.doe@example.com](#)

Example of an invalid email: [johnndoe@example](#) (missing dot after the "@" symbol)

2. JavaScript's Role in Validation

JavaScript is a client-side scripting language used to manipulate the content of web pages dynamically. It allows us to validate user inputs in real-time before submitting the data to the server, reducing the load on the server and enhancing user experience.

In this case, JavaScript is used to:

- **Capture user input:** The email entered by the user is captured from an input field.
- **Check if the email contains necessary characters:** The presence of @ and . is validated using string manipulation functions.
- **Alert the user if the email is invalid:** If the email fails validation, JavaScript will show an error message and prompt the user to correct the input.

3. Components of the Program

HTML Form:

The program uses a simple HTML form with an `<input>` element where the user enters their email address. The form also has a submit button, which the user clicks after entering their data. On form submission, JavaScript is triggered to perform the validation.

JavaScript Function:

The main logic of the program resides in a JavaScript function that is called when the user submits the form. Here's the breakdown of how it works:

1. **Capture User Input:**

The email entered by the user is captured from the input field using the `document.getElementById("email").value` method. This retrieves the text that the user has typed into the email field.

2. **Validation:**

We use JavaScript's built-in `indexOf()` method to check if the `@` and `.` characters are present in the email string:

- `email.indexOf("@")`: If the `@` character is not found, it returns `-1`. This is used to check if the `@` symbol is missing.
- `email.indexOf(".")`: Similarly, it checks if the `.` character is missing.

3. **Error Handling:**

If either of the characters is missing, the program:

- Shows an alert box with a message prompting the user to enter a valid email.
- Clears the input field so the user can re-enter the email address.
- Focuses the cursor back on the email input field, improving the user experience.

4. **Form Submission Control:**

The function returns `false` if the email is invalid, which prevents the form from submitting. If the email is valid, the form can be submitted as usual.

4. Key JavaScript Methods Used

- **indexOf()**: This method searches for a specified substring (in this case, `"@"` and `"."`) within a string. It returns the index of the first occurrence of the substring, or `-1` if the substring is not found.

```
var email = "johndoe@example";
```

```
email.indexOf("@"); // Returns the position of "@" in the string
```

```
email.indexOf("."); // Returns the position of "." in the string
```

- **alert():** This method displays an alert box to the user with a specified message. It is used here to notify the user about the invalid email.

```
alert("Error: Please enter a valid email address with '@' and '.");
```

- **document.getElementById():** This method is used to get the value of an HTML element (like an input field) by its ID. We use it to access the email input field.

```
var email = document.getElementById("email").value;
```

- **focus():** This method is used to give focus to an HTML element (like the email input field) after clearing the input. It helps guide the user back to the problematic field.

```
document.getElementById("email").focus();
```

PROGRAM:

```
<!DOCTYPE html>
<head>
<title>Program for email validation</title>
</head>
<body>
<h2>JavaScript Email Validation</h2>

<input id="textEmail">

<button type="button" onclick="myFunction()">Submit</button>

<p id="demo" style="color: red;"></p>

<script>
function myFunction() {
    var email;

    email = document.getElementById("textEmail").value;

    var reg =
/^([A-Za-z0-9_\-\.])+\@([A-Za-z0-9_\-\.])+\.[A-Za-z]{2,4}$/;

    if (reg.test(textEmail.value) == false)
    {
```

```

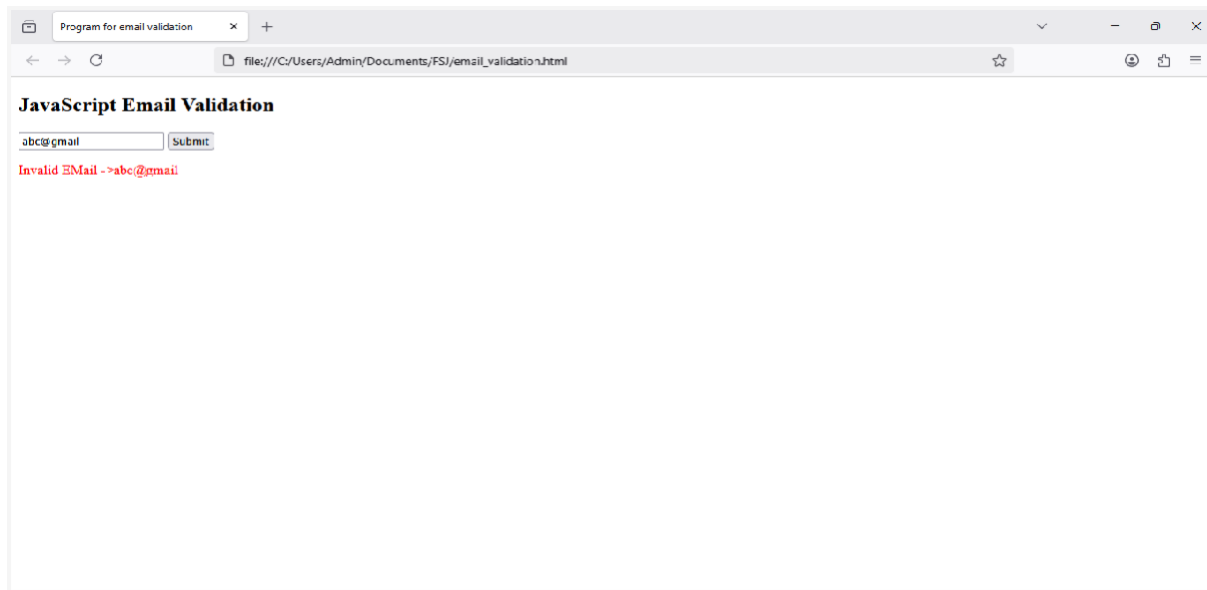
        document.getElementById("demo").style.color = "red";
        document.getElementById("demo").innerHTML="Invalid EMail ->"+
email;

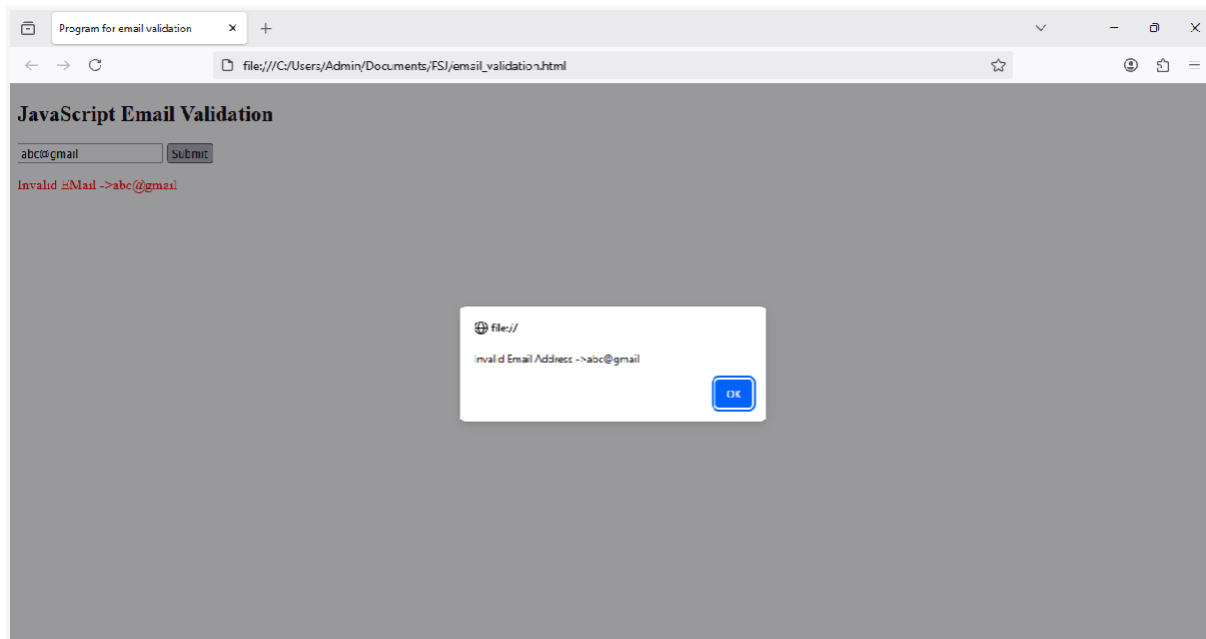
        alert('Invalid Email Address ->'+email);
        return false;
    } else{
        document.getElementById("demo").style.color = "DarkGreen";
        document.getElementById("demo").innerHTML="Valid Email ->"+email;
    }

    return true;
}
</script>
</body>
</html>

```

Output:





Experiment No-9

AIM:Program based on Document Object Model to change background color of the web page automatically after every 5 seconds.

THEORY:

- **HTML Structure:**
Basic structure with a `<body>` tag that displays a message.
A `<style>` block defines smooth transitions for a nicer visual effect.
- **JavaScript Logic:**
An array `colors[]` holds a list of different color codes.
A function `changeBackgroundColor()` changes the page's background using the DOM:
`document.body.style.backgroundColor = colors[index];`

The `setInterval()` method is used to call this function every 5 seconds.
- The index is updated in a cyclic manner using:

`index = (index + 1) % colors.length;`
- **DOM Use:**
JavaScript accesses and modifies the `body` element's `style.backgroundColor` property using the DOM (`document.body`).

PROGRAM:

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <title>Auto Background Color Changer</title>
```



```

<style>
  body {
    transition: background-color 1s;
    text-align: center;
    padding-top: 20%;
    font-family: Arial, sans-serif;
    font-size: 24px;
    color: white;
  }
</style>
<script>
  // Array of background colors
  var colors = ["#FF5733", "#33C1FF", "#75FF33", "#F333FF", "#FFC300", "#DAF7A6"];

  // Counter to keep track of color index
  var index = 0;

  // Function to change background color
  function changeBackgroundColor() {
    // Change the background color using DOM
    document.body.style.backgroundColor = colors[index];

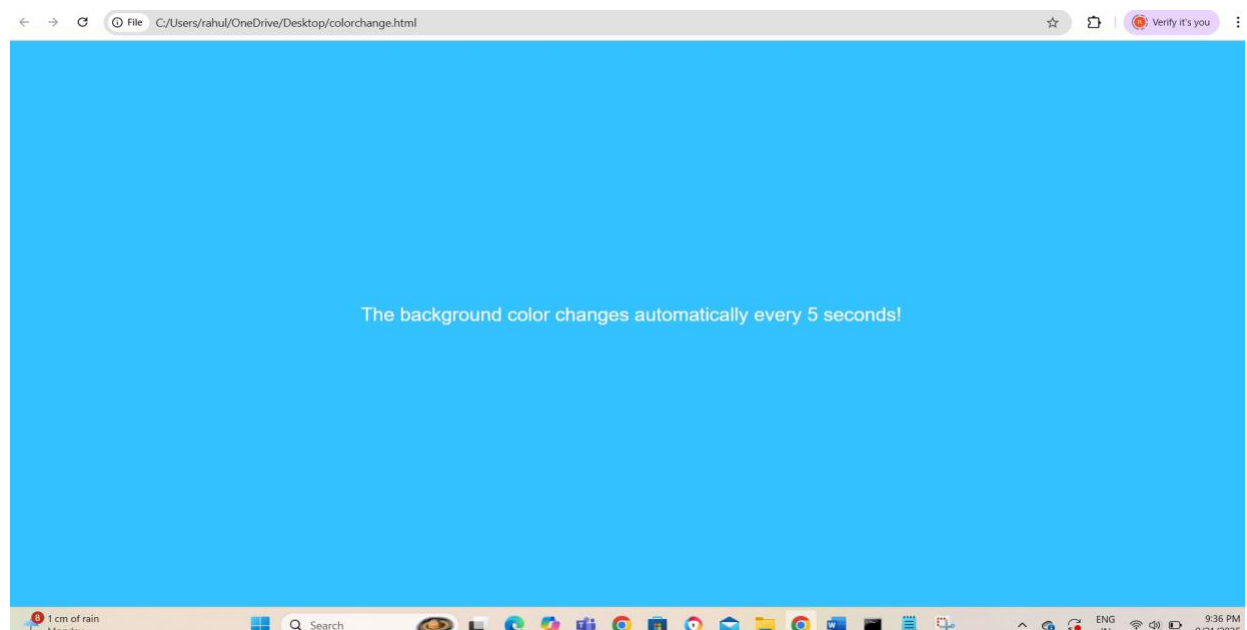
    // Update the index
    index = (index + 1) % colors.length; // Loop back to 0 after the last color
  }

  // Start the interval when the page loads

```

```
window.onload = function () {  
    // Change background color immediately on load  
    changeBackgroundColor();  
    // Set interval to change color every 5 seconds (5000 milliseconds)  
    setInterval(changeBackgroundColor, 5000);  
};  
</script>  
</head>  
<body>  
    <p>The background color changes automatically every 5 seconds!</p>  
</body>  
</html>
```

OUTPUT:



Experiment No-10

AIM: Program for making use of React Hooks that displays four buttons namely, “Red”, “Blue”, “Green”, “Yellow”. On clicking any of these buttons, the code displays the message that you have selected that particular color.

THEORY:How it works:

1. `useState("")` keeps track of the currently selected color.
2. Each button calls `handleClick(color)` to update the state.
3. When a color is selected, the message `You have selected: <color>` is displayed.

```
import React, { useState } from 'react';

function ColorSwitcher() {
  // Declare a state variable 'color' with initial value 'white'
  const [color, setColor] = useState('white');

  return (
    <div style={{ backgroundColor: color, minHeight: '100vh', padding: '20px' }}>
      <h1>Background: {color}</h1>
      <button onClick={() => setColor('red')}>Red</button>
      <button onClick={() => setColor('green')}>Green</button>
      <button onClick={() => setColor('blue')}>Blue</button>
    </div>
  );
}
```

OUTPUT:

