|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **II-Year II-Semester** | **NameoftheCourse** | **L** | **T** | **P** | **C** |
| **PC2201L** | **Advanced Java Programming Lab** | **0** | **0** | **3** | **1.5** |

**Course Objectives**

At the end ofthe coursethestudents will understand

1. ImplementingdatastructuresusingcollectionFramework
2. Basictechnologiesto develop webdocuments.
3. DevelopingClient-Server applications.
4. XMLandWebServers.
5. JavaServlet technologies.

LISTOFEXPERIMENTS

1. To write a Java Program to design an interface for Stack ADTand implement Stack ADT using both Array and Linked List.

Procedure:

The step-by-step procedures for designing an interface for the Stack ADT and implementing it using both an array and a linked list in Java:

* Create a new Java interface called StackADT (Stack Abstract Data Type).
* Declare the following methods within the interface:
* void push(T item): Adds an item to the top of the stack.
* T pop(): Removes and returns the top item from the stack.
* T top(): Returns the top item without removing it.
* boolean isEmpty(): Checks if the stack is empty.
* Create a class called ArrayStack that implements the StackADT interface.
* Use an array to store the stack elements.
* Initialize the array, maintain a top pointer, and set the capacity.
* Implement the methods (push, pop, top, and isEmpty) based on the array operations.
* Handle stack overflow and underflow conditions.
* Create a class called LinkedListStack that also implements the StackADT interface.
* Use a linked list (e.g., java.util.LinkedList) to store the stack elements.
* Implement the methods (push, pop, top, and isEmpty) based on linked list operations.

Sample Code:

import java.io.\*;

interface Mystack

{

    public void pop();

    public void push();

    public void display();

}

class Stack\_array implements Mystack

{

    final static int n=5;

    int stack[]=new int[n];

    int top=-1;

    public void push()

    {

        try

        {

            BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

            if(top==(n-1))

            {

                System.out.println(" Stack Overflow");

                return;

            }

            else

            {

                System.out.println("Enter the element");

                int ele=Integer.parseInt(br.readLine());

                stack[++top]=ele;

            }

        }

        catch(IOException e)

        {

            System.out.println("e");

        }

    }

    public void pop()

    {

        if(top<0)

        {

            System.out.println("Stack underflow");

            return;

        }

        else

        {

            int popper=stack[top];

            top--;

            System.out.println("Popped element:" +popper);

        }

    }

    public void display()

    {

        if(top<0)

        {

            System.out.println("Stack is empty");

            return;

        }

        else

        {

            String str=" ";

            for(int i=0; i<=top; i++)

                str=str+"  "+stack[i]+" <--";

            System.out.println("Elements are:"+str);

        }

    }

}

class Link

{

    public int data;

    public Link nextLink;

    public Link(int d)

    {

        data= d;

        nextLink=null;

    }

    public void printLink()

    {

        System.out.print(" --> "+data);

    }

}

class Stack\_List  implements Mystack

{

    private Link first;

    public Stack\_List()

    {

        first = null;

    }

    public boolean isEmpty()

    {

        return first == null;

    }

    public void push()

    {

        try

        {

            BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

            System.out.println("Enter the element");

            int ele=Integer.parseInt(br.readLine());

            Link link = new Link(ele);

            link.nextLink = first;

            first = link;

        }

        catch(IOException e)

        {

            System.err.println(e);

        }

    }

    public Link delete()

    {

        Link temp = first;

        try

        {

            first = first.nextLink;

        }

        catch(NullPointerException e)

        {

            throw e;

        }

        return temp;

    }

    public void pop()

    {

        try

        {

            Link deletedLink = delete();

            System.out.println("Popped: "+deletedLink.data);

        }

        catch(NullPointerException e)

        {

            throw e;

        }

    }

    public void display()

    {

        if(first==null)

            System.out.println("Stack is empty");

        else

        {

            Link currentLink = first;

            System.out.print("Elements are: ");

            while(currentLink != null)

            {

                currentLink.printLink();

                currentLink = currentLink.nextLink;

            }

            System.out.println("");

        }

    }

}

class StackADT

{

    public static void main(String arg[])throws IOException

    {

        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

        System.out.println("Implementation of Stack using Array");

        Stack\_array stk=new Stack\_array();

        int ch=0;

        do

        {

            System.out.println("1.Push 2.Pop 3.Display 4.Exit 5.Use Linked List");

            System.out.println("Enter your choice:");

            ch=Integer.parseInt(br.readLine());

            switch(ch)

            {

            case 1:

                stk.push();

                break;

            case 2:

                stk.pop();

                break;

            case 3:

                stk.display();

                break;

            case 4:

                System.exit(0);

            }

        }

        while(ch<5);

        System.out.println("Implementation of Stack using Linked List");

        Stack\_List stk1=new Stack\_List();

        ch=0;

        do

        {

            System.out.println("1.Push 2.Pop 3.Display 4.Exit");

            System.out.println("Enter your choice:");

            ch=Integer.parseInt(br.readLine());

            switch(ch)

            {

            case 1:

                stk1.push();

                break;

            case 2:

                try

                {

                    stk1.pop();

                }

                catch(NullPointerException e)

                {

                    System.out.println("Stack underflown");

                }

                break;

            case 3:

                stk1.display();

                break;

            default:

                System.exit(0);

            }

        }

        while(ch<5);

    }

}

Output:

Implementation of Stack using Array

1.Push 2.Pop 3.Display 4.Exit 5.Use Linked List

Enter your choice:

1

Enter the element

10

1.Push 2.Pop 3.Display 4.Exit 5.Use Linked List

Enter your choice:

1

Enter the element

15

1.Push 2.Pop 3.Display 4.Exit 5.Use Linked List

Enter your choice:

1

Enter the element

25

1.Push 2.Pop 3.Display 4.Exit 5.Use Linked List

Enter your choice:

3

Elements are: 10 <-- 15 <-- 25 <--

1.Push 2.Pop 3.Display 4.Exit 5.Use Linked List

Enter your choice:

5

Implementation of Stack using Linked List

1.Push 2.Pop 3.Display 4.Exit

Enter your choice:

1

Enter the element

10

1.Push 2.Pop 3.Display 4.Exit

Enter your choice:

1

Enter the element

15

1.Push 2.Pop 3.Display 4.Exit

Enter your choice:

1

Enter the element

20

1.Push 2.Pop 3.Display 4.Exit

Enter your choice:

3

Elements are: --> 20 --> 15 --> 10

1.Push 2.Pop 3.Display 4.Exit

Enter your choice:

2

Popped: 20

1.Push 2.Pop 3.Display 4.Exit

Enter your choice:

3

Elements are: --> 15 --> 10

1.Push 2.Pop 3.Display 4.Exit

Enter your choice:

4

1. Write a Java program that loads names and phone numbers from a text file where the data is organized as one line per record and each field in a record are separated by a tab (\t). It takes a name or phone number as input and prints the corresponding other value from the hash table (hint: use hash tables).

Procedure:

The step-by-step procedure for writing a Java program that accomplishes the task described:

* + Create a Java program that reads data from the given text file.
  + Use a BufferedReader to read each line from the file.
  + Split each line into fields using the tab character (\t) as the delimiter.
  + Store the name and phone number pairs in a data structure (e.g., a hash table).
  + Create a hash table (also known as a map or dictionary) to store the name-to-phone-number mappings.
  + Use the name as the key and the phone number as the value.
  + You can use Java’s HashMap or Hashtable for this purpose.
  + For each line read from the file:
    - Extract the name and phone number from the split fields.
    - Add an entry to the hash table with the name as the key and the phone number as the value.
  + Prompt the user to input either a name or a phone number.
  + If the user provides a name:
    - Look up the corresponding phone number in the hash table.
    - Print the phone number.
  + If the user provides a phone number:
    - Reverse the hash table (swap keys and values).
    - Look up the corresponding name using the reversed hash table.
    - Print the name.

Sample Code:

import java.io.\*;

import java.util.\*;

public class Phonebook

{

public static void main(String args[])

{

try

{

FileInputStream fis=new FileInputStream("//home/gcet/Desktop/myfile.txt");

Scanner sc=new Scanner(fis).useDelimiter("\t");

Hashtable<String,String> ht=new Hashtable<String,String> ();

String[] strarray;

String a,str;

while(sc.hasNext())

{

a=sc.nextLine();

strarray=a.split("\t");

ht.put(strarray[0],strarray[1]);

System.out.println("hash table values are"+strarray[0]+":"+strarray[1]);

}

Scanner s=new Scanner(System.in);

System.out.println("Enter the name as given in the phone book");

str=s.next();

if(ht.containsKey(str))

{

System.out.println("phone no is"+ht.get(str));

}

else

{

System.out.println("Name is not matched");

}

}

catch(Exception e)

{

System.out.println(e);

}

}

}

Myfile.txt

Surya 567

Ravi 456

Sudha 678

Output:

Surya: 567

Ravi: 456

Sudha: 678

Enter the name as given in the phone book

Ravi

phone no is: 456

Enter the name as given in the phone book

Soni

Name is not matched

1. Write a Java program that connects to a database using JDBC and does add, delete, modify and retrieve operations.

Procedure:

* + Ensure you have the following prerequisites:
    - **JDK**: Install the Java Development Kit.
    - **Database System**: Set up a database system (e.g., MySQL, Oracle, PostgreSQL).
    - **JDBC Driver**: Download the JDBC driver for your specific database system (e.g., MySQL Connector/J, Oracle JDBC driver).
    - **IDE or Text Editor**: Use an IDE (e.g., Eclipse, IntelliJ) or a text editor (e.g., Visual Studio Code) to write your Java code.
  + Load the JDBC driver class into memory using Class.forName("your.driver.class.name").
  + For example, if you’re using MySQL, load the MySQL driver class: Class.forName("com.mysql.cj.jdbc.Driver").
  + Use DriverManager.getConnection(url, username, password) to create a connection to your database.
  + Replace url, username, and password with your actual database details.
  + **Create (INSERT)**:
    - Create an INSERT SQL statement.
    - Use a PreparedStatement to execute the statement with appropriate parameters.
    - Example:
  + **Retrieve (SELECT)**:
    - Create a SELECT SQL query.
    - Execute the query using a Statement or PreparedStatement.
    - Process the retrieved data (e.g., iterate through a ResultSet).
    - Example:
  + **Update (UPDATE)**:
    - Create an UPDATE SQL statement.
    - Execute the statement using a PreparedStatement.
    - Example:
  + **Delete (DELETE)**:
    - Create a DELETE SQL statement.
    - Execute the statement using a PreparedStatement.
    - Example:
  + Always close the database connection when done.
  + Use connection.close() to release resources.
  + Wrap your database-related code in try-catch blocks to handle exceptions (e.g., SQLException).
  + Compile your Java program.
  + Run it to test the database operations.

Sample Code:

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.sql.\*;

public class EmployeeDB {

private Connection con=null;

private Statement st=null;

private ResultSet rs=null;

private PreparedStatement pst=null;

EmployeeDB() throws ClassNotFoundException, SQLException ,IOException

{

Class.forName("oracle.jdbc.driver.OracleDriver");

con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","HR","HR");

InputStreamReader isr = new InputStreamReader(System.in);

BufferedReader br =new BufferedReader(isr);

System.out.println("MENU\n1.CREATE\n2.INSERT\n3.UPDATE\n4.DELETE\n");

for(;;)

{

System.out.println("Enter your choice\n");

int ch=Integer.parseInt(br.readLine());

switch(ch)

{

case 1:

System.out.println("Enter your query to create a table\n");

String q1=br.readLine();

st=con.createStatement();

st.execute(q1);

System.out.println("Query executed successfully");

break;

case 2:

System.out.println("Enter your query to Update the table\n");

String q2=br.readLine();

pst=con.prepareStatement(q2);

pst.executeUpdate(q2);

System.out.println("Query executed successfully");

break;

case 3:

System.out.println("Enter your query for selection");

String q3=br.readLine();

st=con.createStatement();

rs=st.executeQuery(q3);

DisplayRS(rs);

System.out.println("Query executed successfully");

break;

case 4:

System.out.println("Enter your query to delete from table\n");

String q4=br.readLine();

st=con.createStatement();

st.executeUpdate(q4);

System.out.println("Query executed successfully");

break;

default:

System.exit(0);

}

}

}

publicstaticvoid main(String[] args) throws ClassNotFoundException, SQLException, IOException

{

final EmployeeDB s1=new EmployeeDB();

System.out.println("MENU\n1.INSERT\n2.UPDATE\n3.DELETE");

}

privatevoid DisplayRS(ResultSet rs) throws SQLException {

ResultSet rs1=rs;

ResultSetMetaData rsm=rs1.getMetaData();

int col=rsm.getColumnCount();

int count=1;

boolean b=rs1.next();

if(!b)

{

System.out.println("Data not found");

}

else

do

{ System.out.println("Record" +(count++)+"=>");

for(int j=0;j<col;j++)

System.out.println(rs1.getString(j+1)+"\t");

System.out.println ("");

} while(rs1.next()); } }

Output:

MENU

1.CREATE

2.INSERT

3.UPDATE

4.DELETE

Enter your choice

1

Enter your query to create a table

create table employee (emp\_id number, emp\_name varchar2(20), emp\_salary number)

Query executed successfully

Enter your choice

2

Enter your query to Update the table

insert into employee values (101, 'Alice', 5000)

Query executed successfully

Enter your choice

2

Enter your query to Update the table

insert into employee values (102, 'Bob', 6000)

Query executed successfully

Enter your choice

3

Enter your query for selection

select \* from employee

Record 1=>

101

Alice

5000

Record 2=>

102

Bob

6000

Query executed successfully

Enter your choice

4

Enter your query to delete from table

delete from employee where emp\_id = 102

Query executed successfully

Enter your choice

3

Enter your query for selection

select \* from employee

Record 1=>

101

Alice

5000

Query executed successfully

Enter your choice

5

1. Write a java program that prints the meta-data of a given table

Procedure:

 the simple steps for a Java program that prints the metadata of a given database table:

* + Load the appropriate JDBC driver for your database system (e.g., MySQL, Oracle).
  + This step ensures that Java can communicate with the database.
  + Create a connection to your database using connection details (URL, username, password).
  + The connection allows your program to interact with the database.
  + Use the DatabaseMetaData interface provided by JDBC.
  + This interface provides information about the database, driver, and available tables.
  + Retrieve details such as driver name, driver version, user name, database product name, and product version.
  + These details help you understand the database environment.
  + Display the retrieved metadata to the user.
  + You can print it to the console or use it in your application as needed.

Sample Code:

ConnectionUtil.Java

package LabProgs;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

public class ConnectionUtil

{

public static Connection getConnection() throws SQLException {

Connection connection = null;

try {

Class.forName("com.mysql.jdbc.Connection");

connection = DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/test",

"root", "system");

} catch (ClassNotFoundException e) {

e.printStackTrace();

} catch (SQLException e) {

e.printStackTrace();

}

return connection;

}

import java.io.\*;

import java.sql.\*;

import java.util.\*;

import java.util.logging.Level;

import java.util.logging.Logger;

public class Prog14 {

public static void main(String args[]) {

try {

Connection conn = null;

conn = ConnectionUtil.getConnection();

Prog14 prog14 = new Prog14();

prog14.printMetaData(conn, "student\_details");

} catch (SQLException ex) {

Logger.getLogger(Prog14.class.getName()).log(Level.SEVERE, null, ex);

}

}

private void printMetaData(Connection conn, String tableName) {

String query = "select \* from " + tableName;

Statement statement;

try {

statement = conn.createStatement();

ResultSetMetaDatametaData = statement.executeQuery(query).getMetaData();

longcolSize = metaData.getColumnCount();

System.out.println("There are "+ colSize + " columns in table

"+tableName);

System.out.println("Columns are:");

for(inti=1;i<=colSize;i++) {

String colName = metaData.getColumnLabel(i);

String colType = metaData.getColumnTypeName(i);

System.out.println(colName+"-->"+colType);

}

} catch (SQLException e) {

e.printStackTrace();

}

}

}

1. Write a java program to implement client-server application

Procedure:

The simple steps for implementing a client-server application in Java:

* + Ensure you have the required libraries or modules for socket programming in Java.
  + These libraries allow communication between the client and server.
  + Decide which part of your application will act as the client and which part will act as the server.
  + The client sends requests, and the server processes those requests.
  + The server creates a socket (a communication endpoint) and listens for incoming connections.
  + The client initiates a connection by creating a socket and specifying the server’s IP address and port number.
  + Define a communication protocol between the client and server.
  + Decide how data will be exchanged (e.g., messages, files, commands).
  + The client connects to the server using its IP address and port.
  + It sends requests (data) to the server.
  + The client can also receive responses from the server.
  + The server accepts incoming connections.
  + It processes client requests (e.g., performs calculations, retrieves data).
  + The server sends responses back to the client.
  + Implement error handling for scenarios like connection failures, timeouts, or unexpected data.
  + Gracefully handle exceptions to prevent crashes.
  + Properly close sockets and release resources when communication is complete.
  + Both client and server should close their sockets.
  + Run the server and client applications separately.
  + Verify that they can communicate successfully.
  + Test different scenarios (e.g., multiple clients connecting simultaneously).
  + Consider deployment options (local, cloud, etc.).
  + Plan for scalability if your application needs to handle more clients.

Sample Code:

import java.net.\*;

import java.io.\*;

import java.util.\*;

public class TcpClient

{

public static void main(String args[])throws IOException

{

try

{

//creating client socket with default ip and port nos and establishing connection with server whose ip is 127.0.0.0 and port 95

Socket con= new Socket("localhost",95);

//creating in object to read data from socket

BufferedReader in=new BufferedReader(new InputStreamReader(con.getInputStream()));

//creating out object to write data into socket

PrintWriter out=new PrintWriter(con.getOutputStream(),true);

while(true)

{

System.out.print("enter the message to the server:");

BufferedReader din=new BufferedReader(new InputStreamReader(System.in));

//reading data from keyboard inorder to send to server

System.out.println("Enter 5 nums");

int a[]=new int[5];

for(int i=0;i<5;i++){

a[i]=Integer.parseInt(din.readLine());

}

String st=Arrays.toString(a);

//writing(sending) data to server

out.println(st);

//reading(recieving) data from server

String s1=in.readLine();

//printing data recived from server on console

System.out.println("from server:"+s1);

if(st.equalsIgnoreCase("bye")||st==null)

break;

}

//closing all the streams opened

in.close();

out.close();

con.close();

}

catch(UnknownHostException e)

{

}

}

}

import java.net.\*;

import java.io.\*;

import java.util.\*;

public class TcpServer

{

public static void main(String args[])throws IOException

{

try

{

//Server scoket creation with default ip address and port no 95

ServerSocket s= new ServerSocket(95);

System.out.println("Server Waiting for the Client");

//waiting for client requests

Socket cs=s.accept();

//Fetching client address and displaying

InetAddress ie=cs.getInetAddress();

String cli=ie.getHostAddress();

System.out.println("Connected to the client with IP "+cli);

//creating in object on client socket for read

BufferedReader in=new BufferedReader(new InputStreamReader(cs.getInputStream()));

//creating out object on clinet socket for write

PrintWriter out=new PrintWriter(cs.getOutputStream(),true);

do

{

//reading(recieving) data from client(if no data from client waits till data is ready)

String st=in.readLine();

//converting into array

String c[]=st.replaceAll("\\[","").replaceAll("]","").replaceAll(" ","").split(",");

int n[]=new int[c.length];

for(int i=0;i<c.length;i++){

n[i]=Integer.valueOf(c[i]);

}

//sorting

TcpServer ts=new TcpServer();

ts.sort(n);

String res=Arrays.toString(n);

//writing(sending) data to client

out.println(res);

if(st.equalsIgnoreCase("bye")||st==null)

break;

//printing data from client on console

System.out.println("from client. "+st);

}while(true);

//closing all streams opened

in.close();

out.close();

cs.close();

}

catch(IOException e)

{

}

}

void sort(int n[]){

int i,j,temp;

int l=n.length;

for(i=0;i<l-1;i++){

for(j=0;j<l-i-1;j++){

if(n[j]>n[j+1]){

temp=n[j];

n[j]=n[j+1];

n[j+1]=temp;

}

}

}

}

}

1. DevelopanddemonstrateaHTML5documentthatillustratestheuseoforderedlist, unordered list, table, borders, padding, color, and the <div>&<span> tag.

Procedure:

Steps for creating an HTML5 document that illustrates the use of ordered lists, unordered lists, tables, borders, padding, color, and the <div> and <span> tags:

* + Start by creating a new HTML file (with a .html extension) using a text editor or an integrated development environment (IDE).
  + Begin your HTML document with the <!DOCTYPE html> declaration.
  + Add the <html> element to enclose the entire content.
  + Inside the <html> element, include the <head> and <body> sections.
  + Use the <ol> tag to create an ordered list.
  + Add list items using the <li> tag within the <ol>.
  + Use the <ul> tag to create an unordered list.
  + Add list items using the <li> tag within the <ul>.
  + Create a table using the <table> tag.
  + Define rows with the <tr> tag.
  + Add table data cells using the <td> tag within each row.
  + Apply borders to elements using CSS (Cascading Style Sheets).
  + Use the border property to set borders around elements.
  + Adjust padding using the padding property to create space inside elements.
  + Set colors for text, backgrounds, and borders using CSS.
  + Use color names, hexadecimal values, or RGB values.
  + Use the <div> tag to create a block-level container.
  + Use the <span> tag to create an inline container.
  + Apply styles or manipulate content within these containers.

Sample code:

<!DOCTYPE html>

<html>

<head>

<title>Lists,Tables</title>

<style>

/\* CSS for styling \*/

body { font-family: Arial, sans-serif; color: #333; }

.container { margin: 20px; padding: 10px; border: 2px solid #ccc; }

table { border-collapse: collapse; width: 100%; }

th, td { border: 1px solid #ddd; padding: 8px; text-align: left; }

th { background-color: #f2f2f2; }

.red { color: red; }

.blue { color: blue; }

</style>

</head>

<body>

<div class="container">

<h1>HTML5 Demonstration</h1>

<h2>Ordered List and Unordered List</h2>

<ol>

<li>College</li>

<li>Name

<ul>

<li>VVIT</li>

<li>GUNTUR</li>

</ul>

</li>

<li>Item 3</li>

</ol>

<h2>Table</h2>

<table>

<tr>

<th>Name</th>

<th>Age</th>

<th>City</th>

</tr>

<tr>

<td>rama</td>

<td>30</td>

<td>Vijayawada</td>

</tr>

<tr>

<td>vinay</td>

<td>25</td>

<td>Guntur</td>

</tr>

</table>

<h2>Using Div and Span</h2>

<div>

<p>This is a <span class="red">red</span> span inside a div.</p>

<p>This is a <span class="blue">blue</span> span inside a div.</p>

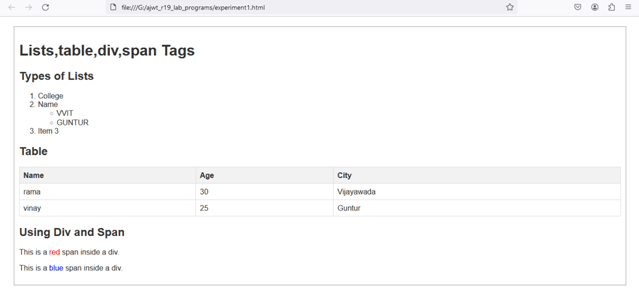
</div>

</div>

</body>

</html>

Output:



1. Write HTML5 code to provide intra and inter document linking.

Procedure:

Steps for creating an HTML5 document that illustrates intra and inter-document linking:

* + **Step 1**: Identify the section or element you want to link to within the same page.
  + **Step 2**: Add an id attribute to that section or element.
  + **Step 3**: Create a link (anchor) that points to the specified id using the <a> tag with the href attribute set to #your\_id.
  + **Step 1**: Identify the external page (document) you want to link to.
  + **Step 2**: Use the <a> tag with the href attribute set to the URL of the external page.
  + **Step 3**: Optionally, specify the target attribute to control how the linked page opens (e.g., in the same window, a new window, etc.).

Sample Code:

<!DOCTYPE html>

<html lang="en">

<head>

<title>Hyper Links Example</title>

</head>

<body>

<h1>Hyper Links</h1>

<p><a href="#part1">Go to Part1</a></p>

<p><a href="#part2">Go to Part2</a></p>

<h2 id="part1">Part1 </h2>

<p>Here is my part1 section.</p>

<h2 id="part2">Part2</h2>

<p>Here is my part1 section.</p>

<h1>Inter-Document Linking</h1>

<p><a href="experiment1.html">Link to Experiment1</a></p>

</body>

</html>

Output:



1. Create a webpage with the following using HTML5:
   1. To embed an image map in a web page
   2. To fix the hot spots
   3. Show all the related information when the hotspots are clicked
   4. Create a web page with all types of Cascading style sheets.

Procedure:

Steps for creating a webpage with the specified features using HTML5:

* + Identify the image you want to use as the map.
  + Define clickable areas (hotspots) within the image using the <map> and <area> tags.
  + Specify the shape (rectangular, circular, or polygonal) and coordinates for each hotspot.
  + Link each hotspot to relevant content or pages.
  + Ensure that the coordinates of each hotspot accurately correspond to the desired clickable area on the image.
  + Test the hotspots to verify that they work as expected when clicked.
  + When a hotspot is clicked, show additional information related to that area.
  + You can achieve this by linking the hotspot to a separate page or by using JavaScript to display a tooltip or modal with relevant details.
  + Create a new HTML file.
  + Add content to the page (text, images, etc.).
  + Apply different types of CSS styles to elements:
    - Inline styles (using the style attribute)
    - Internal styles (within the <style> tag in the <head>)
    - External styles (using an external CSS file linked via the <link> tag)
    - Explore various CSS properties (e.g., font, color, margin, padding, border) to style your content.

Sample code:

<!DOCTYPE html>

<html>

<head>

<title>Image Map</title>

<style>

#info {

display: none;

position: absolute;

background-color: rgba(255, 255, 255, 0.8);

padding: 10px;

border: 1px solid #ccc;

}

</style>

</head>

<body>

<h1>Image Map</h1>

<img src="bird1.jpg" width=400 height=400 alt="Imagemap"

usemap="#hotspots">

<map name="hotspots">

<area shape="rect" coords="0,0,50,50" alt="Hotspot 1"

href="experiment1.html" data-info="hotspot 1">

<area shape="circle" coords="100,100,50" alt="Hotspot 2"

href="experiment1.html" data-info="hotspot 2">

<area shape="poly" coords="150,100,180,50,120,200" alt="Hotspot 3"

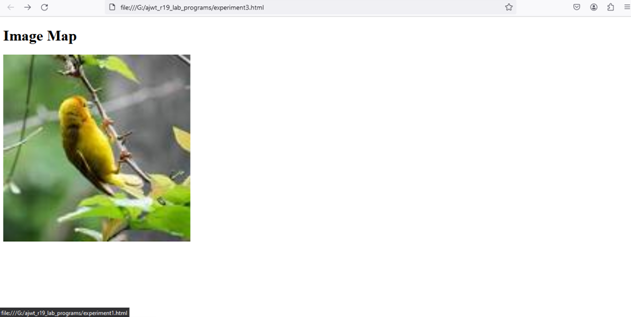
href="experiment3.html" data-info="hotspot 3">

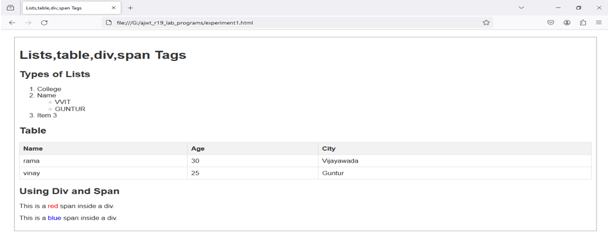
</map>

</body>

</html>

Output:





1. Create a webpage with the following using CSS:
2. Text shadows, rounded corners and box shadows.
3. Linear and Radial gradients.
4. Animation
5. Transitions and Transformations.

Procedure:

* + **Text Shadows**:
    - Apply shadows to text using the text-shadow property.
    - Specify horizontal and vertical offsets, blur radius, and color.
  + **Rounded Corners**:
    - Use the border-radius property to round the corners of elements (e.g., divs, buttons).
    - Set the radius value to create smooth curves.
  + **Box Shadows**:
    - Add shadows to elements using the box-shadow property.
    - Define horizontal and vertical offsets, blur radius, spread distance, and shadow color.
  + **Linear Gradients**:
    - Create smooth color transitions along a straight line.
    - Use the linear-gradient function in CSS.
    - Specify start and end colors, and optionally add color stops.
  + **Radial Gradients**:
    - Create color transitions from a central point outward.
    - Use the radial-gradient function.
    - Set center position, shape, and color stops.
  + Apply animations to elements using CSS.
  + Use the @keyframes rule to define animation steps.
  + Specify animation properties (duration, timing function, delay, iteration count).
  + **Transitions**:
    - Create smooth transitions between property values (e.g., color, size, position).
    - Use the transition property.
    - Specify transition duration and easing function.
  + **Transformations**:
    - Modify element properties (e.g., scale, rotate, skew, translate).
    - Use the transform property.
    - Apply transformations to enhance visual effects.

Sample Code:

<!DOCTYPE html>

<html>

<head>

<title>CSS text animation</title>

<style>

body { font-family: Arial, sans-serif; background-color: #f0f0f0;

margin: 0; padding: 0; }

/\* css text shados,round,. \*/

.text-box { width: 300px; padding: 20px; background-color: #ffffff;

border-radius: 10px; box-shadow: 0 0 10px rgba(0, 0, 0, 0.3);

margin: 50px auto; }

h1 { text-align: center; text-shadow: 2px 2px 4px #888888; }

/\* css linear gradient \*/

.linear-gradient-box { width: 200px; height: 200px;

background: linear-gradient(to right, #ff7e5f, #feb47b); margin: 20px auto;}

/\* css radial gradient \*/

.radial-gradient-box { width: 200px; height: 200px;

background: radial-gradient(circle, #ff5f6d, #ffc371); margin: 20px auto; }

/\* css animate\*/

.animated { width: 100px; height: 100px; background-color: #4CAF50;

position: relative; animation: move 2s infinite alternate; }

@keyframes move {

from { left: 0px; }

to { left: 200px; }

}

/\* Transitions and Transformations \*/

.transition-box {width: 100px; height: 100px; background-color: #008CBA;

margin: 20px auto; transition: transform 0.5s ease-in-out; }

.transition-box:hover { transform: rotate(360deg); }

</style>

</head>

<body>

<div class="text-box">

<h1>CSS example</h1>

<p>This box demonstrates text shadows, rounded corners, and box

shadows.</p>

</div>

<div class="linear-gradient-box"></div>

<div class="radial-gradient-box"></div>

<div class="animated"></div>

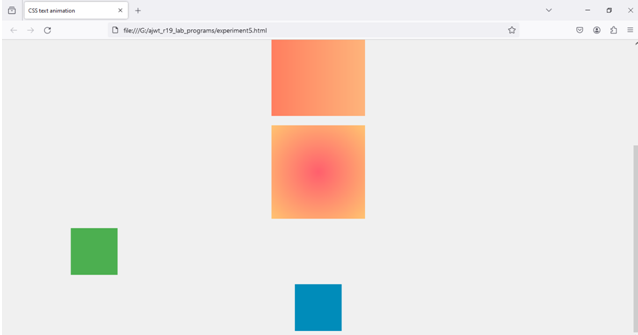
<div class="transition-box"></div>

</body>

</html>

Output:





1. Create a home page for "Cyber book stores" that will display the various books available, the authors and prices of the books. Include a list box that contains various subjects and a "submit" button, which displays information about the books on the subject required by the user.

Procedure:

* + Create an HTML file for your homepage.
  + Use the <html>, <head>, and <body> tags to structure your content.
  + Add a header section that includes the website logo, title, and navigation links (if any).
  + Create a section to display various books available in your store.
  + Include details such as book titles, authors, and prices.
  + You can use tables, lists, or other HTML elements to organize this information.
  + Add a list box (dropdown menu) that contains various subjects (e.g., Fiction, Non-Fiction, Science, History).
  + Populate the list with relevant subjects.
  + Include a “Submit” button next to the subject list box.
  + When the user selects a subject and clicks the button, it should trigger an action (e.g., display books related to the selected subject).
  + Use JavaScript or server-side scripting (e.g., PHP) to handle the form submission.
  + When the user selects a subject and clicks “Submit,” retrieve the relevant book information (titles, authors, prices).
  + Display this information on the page dynamically (without refreshing the entire page).
  + Apply CSS styles to make your homepage visually appealing.
  + Use colors, fonts, and spacing to create an attractive design.
  + Ensure that the layout is user-friendly and responsive.
  + Save your HTML file.
  + Open it in a web browser to verify that the books, subjects, and submit button work as expected.

Sample Code:

Sample Code:

<!DOCTYPE html>

<html>

<head>

<title>Cyber Book Stores</title>

<link rel="stylesheet" href="bookstyle.css">

</head>

<body>

<div class="container">

<h1>Cyber Book Stores</h1>

<form id="bookForm" name="f1">

<label for="subject">Select a Subject:</label>

<select id="subject">

<option value="CSE">CSE</option>

<option value="ECE">ECE</option>

</select>

<br><br>

<input type="submit" value="Submit">

</form>

<h2>Books</h2>

<ul class="book-list" id="bookList">

</ul>

</div>

<script>

document.getElementById('bookForm').addEventListener('submit', function(event) {

event.preventDefault(); // Prevent default form submission

var subject = document.getElementById('subject').value;

displayBooks(subject);

});

function displayBooks(subject)

{

var csebooks = [

{ title: 'C', author: 'Hari', price: '$10.99' },

{ title: 'C++', author: 'Galvin', price: '$12.99' },

{ title: 'HTML', author: 'Dietel', price: '$9.99' }

];

var ecebooks = [

{ title: 'Co', author: 'xyz', price: '$10.99' },

{ title: 'Micro Processor', author: 'Sinha', price: '$12.99' },

{ title: 'VLSI Design', author: 'Sham', price: '$9.99' }

];

var bookList = document.getElementById('bookList');

bookList.innerHTML = ''; // Clear previous book list

var booktype;

if(subject=='CSE')

booktype=csebooks;

else

if(subject=='ECE')

booktype=ecebooks;

// Iterate through the books and create list items to display them

booktype.forEach(function(book) {

var li = document.createElement('li');

li.className = 'book-item';

li.innerHTML = '<strong>Title:</strong> ' + book.title + '<br>' +

'<strong>Author:</strong> ' + book.author + '<br>' +

'<strong>Price:</strong> ' + book.price;

bookList.appendChild(li); // Append the created list item to the book list

});

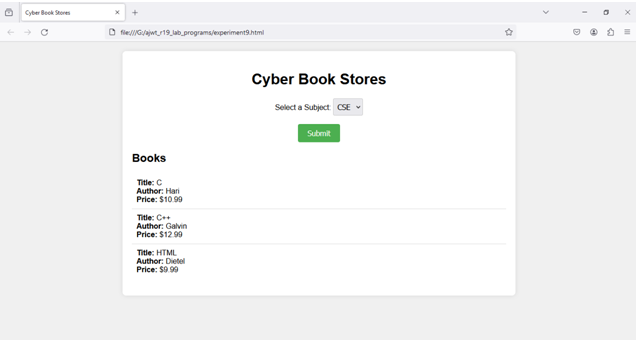
}

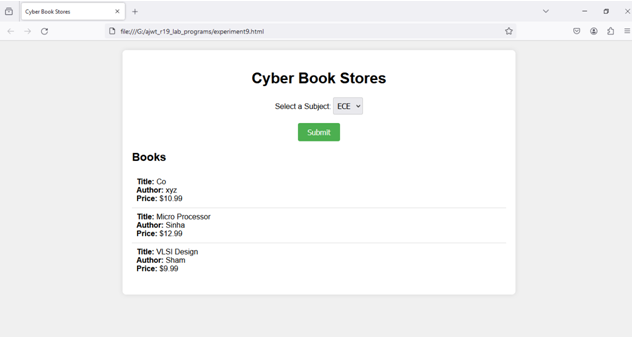
</script>

</body>

</html>

Output:





1. Write an XML file which displays the book details that includes the following:
   1. Title of book
   2. Author name
   3. Edition
   4. Price

Write a DTD to validate the above XML file and display the details in atable (to do this use XSL).

Procedure:

* + Create a new XML file (e.g., books.xml) using a text editor or an XML editor.
  + Define the book details as specified:
    - Each book should have elements for title, author name, edition, and price.
  + Add book details to the XML file.
  + Specify the title, author name, edition, and price for each book.
  + Write a DTD to define the structure of the XML file.
  + Specify the elements, their order, and data types.
  + Ensure that the DTD validates the book details correctly.
  + Add the DTD reference to the XML file:
  + Create an XSL file (e.g., books.xsl) to transform the XML data into a table format.
  + Define the XSL rules to display the book details in a tabular layout.
  + Use an XSLT processor (e.g., browser, command-line tool) to apply the XSL transformation to the XML file.
  + The result should be an HTML table displaying the book details.

Sample Code:

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

<!DOCTYPE Books SYSTEM "books.dtd">

<bookinfo>

   <book>

    <bookname>Java Script</bookname>

    <author>Schildt</author>

    <isbn>ISB-5210</isbn>

    <publisher>TATA Publications</publisher>

    <edition>1st Edition</edition>

    <price>1200</price>

     </book>

    <book>

    <bookname>HTML</bookname>

    <author>Herbert Schildt</author>

    <isbn>ISB-5222</isbn>

    <publisher>Dreamtech</publisher>

    <edition>10th Edition</edition>

    <price>600</price>

     </book>

    <book>

    <bookname>XML</bookname>

    <author>Ditel</author>

    <isbn>ISB-5002</isbn>

    <publisher>Pearson</publisher>

    <edition>20th Edition</edition>

    <price>800</price>

     </book>

</bookifo>

books.dtd

<!ELEMENT bookinfo (book\*)>

<!ELEMENT book (bookname,author,isbn,publisher,edition,price)>

<!ELEMENT bookname (#PCDATA)>

<!ELEMENT author (#PCDATA)>

<!ELEMENT isbn (#PCDATA)>

<!ELEMENT publisher (#PCDATA)>

<!ELEMENT edition (#PCDATA)>

<!ELEMENT price (#PCDATA)>

**2-->Write a DTD to validate the above XML file and display the details in a table (to do this use XSL).**

Xml document validation using DTD

bookinfo.xml

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

<!DOCTYPE bookinfo [

<!ELEMENT bookinfo (book\*)>

<!ELEMENT book (bookname,author,isbn,publisher,edition,price)>

<!ELEMENT bookname (#PCDATA)>

<!ELEMENT author (#PCDATA)>

<!ELEMENT isbn (#PCDATA)>

<!ELEMENT publisher (#PCDATA)>

<!ELEMENT edition (#PCDATA)>

<!ELEMENT price (#PCDATA)>

]>

<bookinfo>

   <book>

    <bookname>Java Script</bookname>

    <author>Schildt</author>

    <isbn>ISB-5210</isbn>

    <publisher>TATA Publications</publisher>

    <edition>1st Edition</edition>

    <price>1200</price>

     </book>

    <book>

    <bookname>HTML</bookname>

    <author>Herbert Schildt</author>

    <isbn>ISB-5222</isbn>

    <publisher>Dreamtech</publisher>

    <edition>10th Edition</edition>

    <price>600</price>

     </book>

    <book>

    <bookname>XML</bookname>

    <author>Ditel</author>

    <isbn>ISB-5002</isbn>

    <publisher>Pearson</publisher>

    <edition>20th Edition</edition>

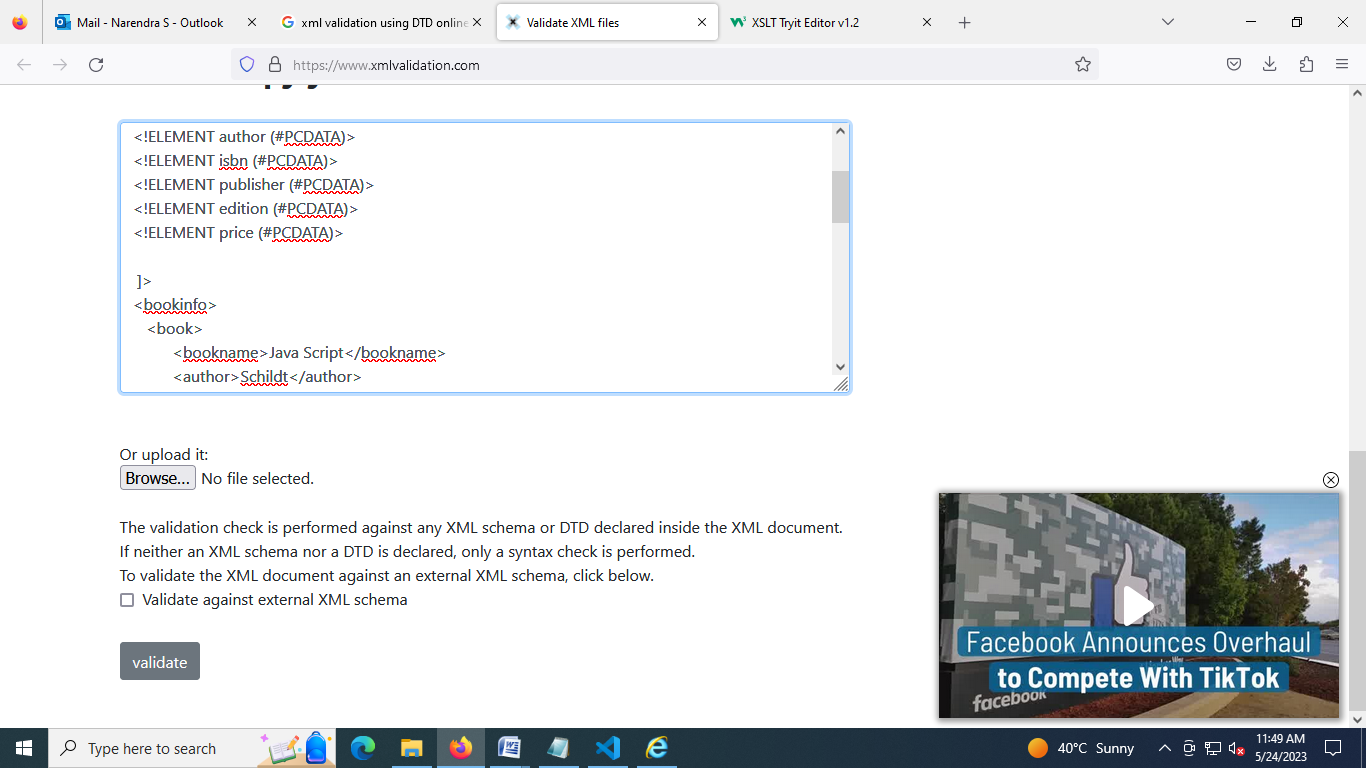
    <price>800</price>

     </book>

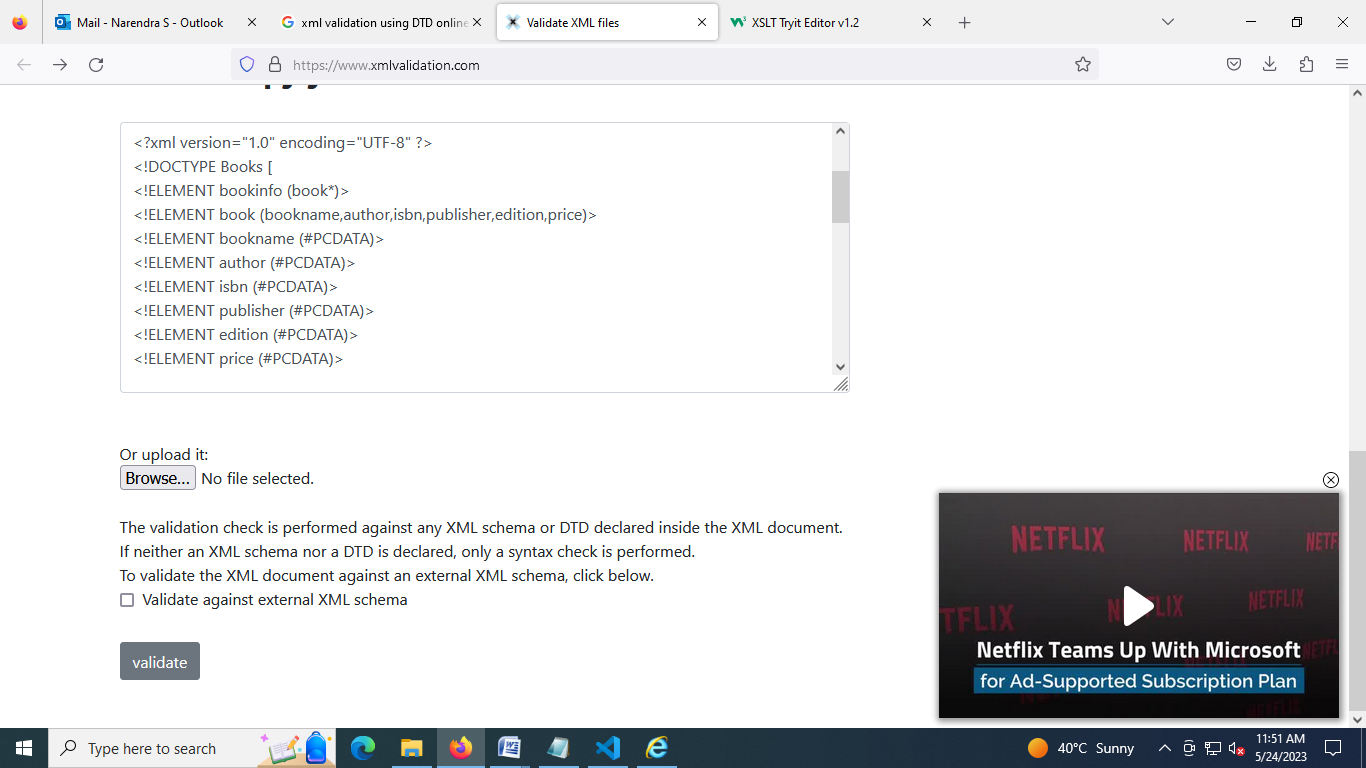
</bookinfo>

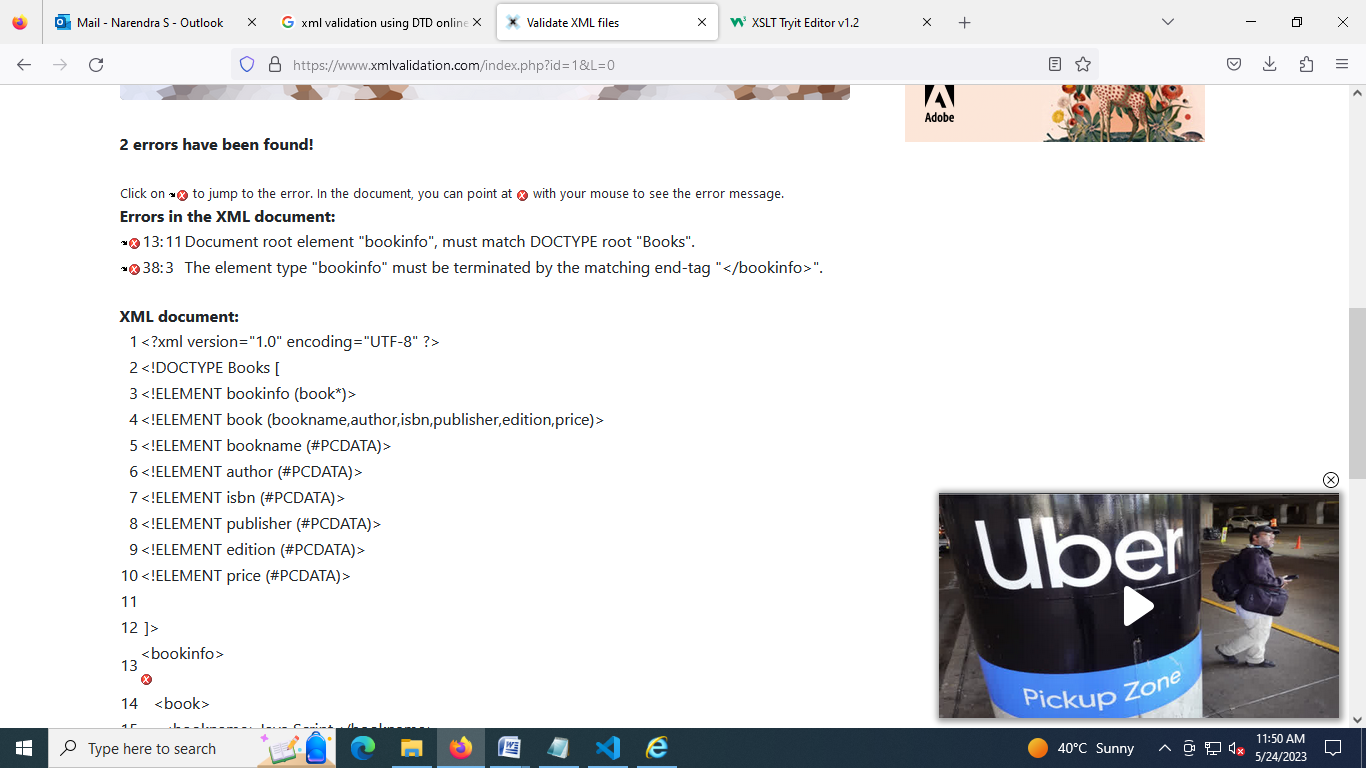
**Use** [**https://www.xmlvalidation.com/**](https://www.xmlvalidation.com/) **to validate xml using DTD or Schema**

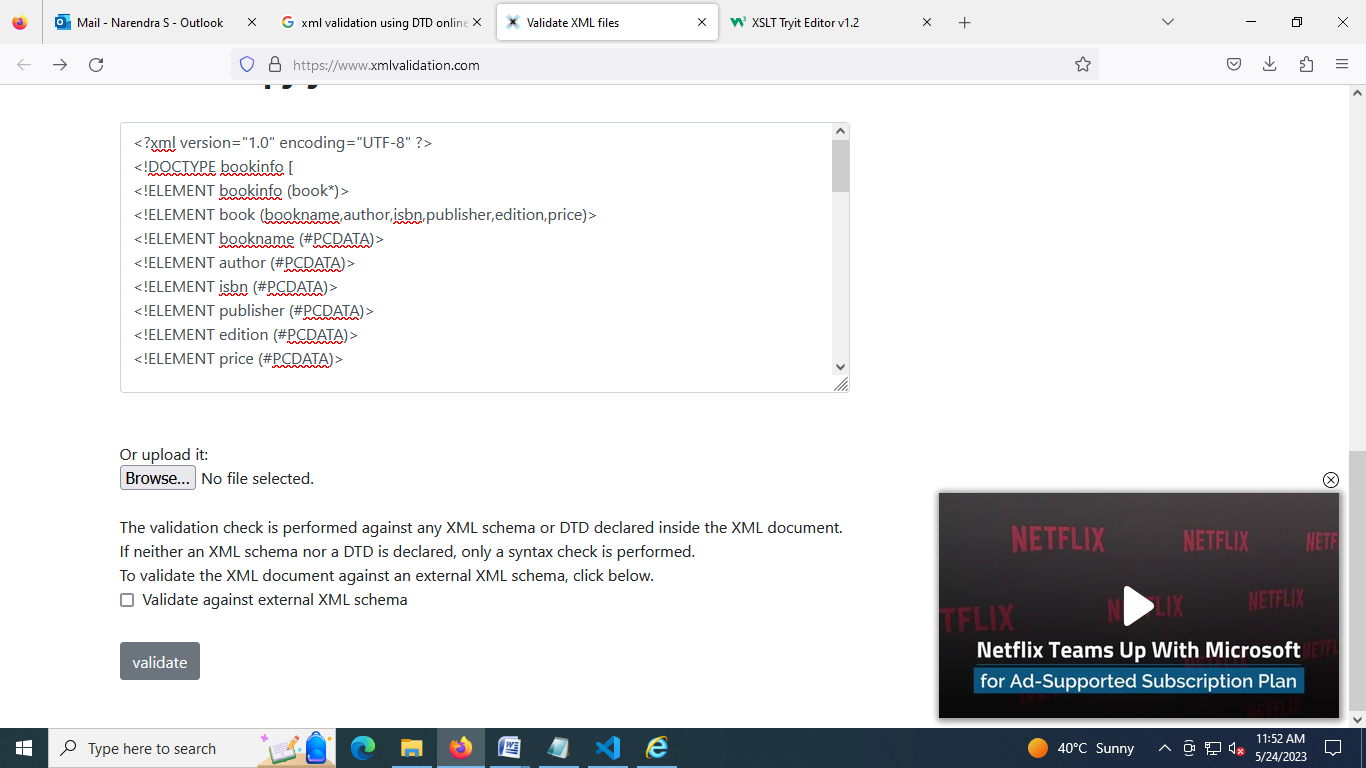
**Change root element name in DOCTYPE element to Books to check validation works**



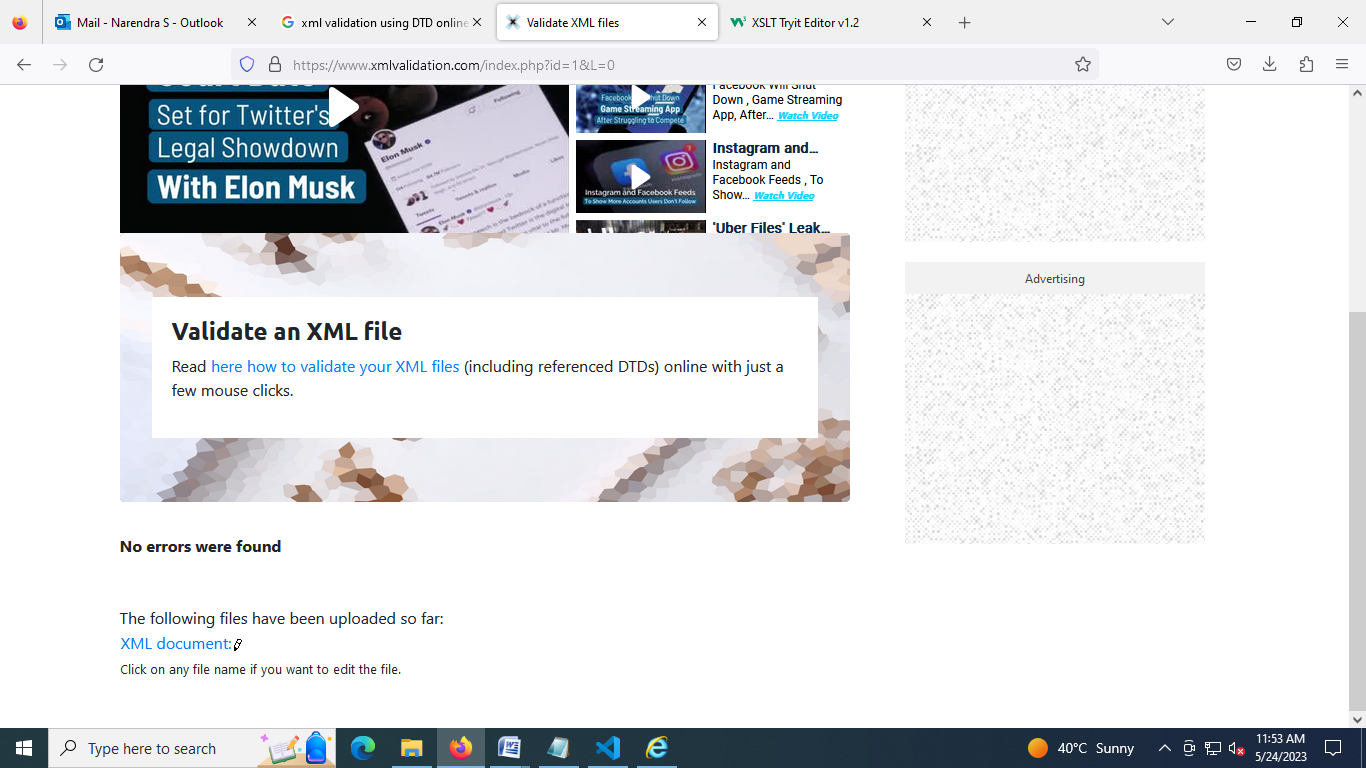
**Output after validation showing error as we given root element name in dtd as books**







Validation after making root element as bookinfo in dtd



**Applying XSL to display xml file content in table form**

book.xml

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

<!DOCTYPE Books SYSTEM "books.dtd">

<?xml-stylesheet type="text/xsl" href="book.xsl"?>

<bookinfo>

   <book>

    <bookname>Java Script</bookname>

    <author>Schildt</author>

    <isbn>ISB-5210</isbn>

    <publisher>TATA Publications</publisher>

    <edition>1st Edition</edition>

    <price>1200</price>

     </book>

       <book>

    <bookname>HTML</bookname>

    <author>Herbert Schildt</author>

    <isbn>ISB-5222</isbn>

    <publisher>Dreamtech</publisher>

    <edition>10th Edition</edition>

    <price>600</price>

     </book>

    <book>

    <bookname>XML</bookname>

    <author>Ditel</author>

    <isbn>ISB-5002</isbn>

    <publisher>Pearson</publisher>

    <edition>20th Edition</edition>

    <price>800</price>

     </book>

</bookinfo>

book.xsl

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

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">

  <html>

    <head>

        <title>XSL Example</title>

    </head>

    <body>

    <table border="1.0" style="color:blue;">

       <tr>

       <th>Book NAME </th>

       <th>AUTHOR</th>

       <th>ISBN</th>

       <th>PUBLISHER</th>

       <th>EDITION</th>

       <th>PRICE</th>

      </tr>

      <xsl:for-each select="bookinfo/book">

     <tr>

      <td style="color:white;background-color:gray;">

<xsl:value-of select="bookname" /> </td>

      <td style="color:white;background-color:gray;">

<xsl:value-of select="author" /> </td>

     <td style="color:white;background-color:gray;">

<xsl:value-of select="isbn" /> </td>

     <td style="color:white;background-color:gray;">

<xsl:value-of select="publisher" /> </td>

     <td style="color:white;background-color:gray;">

<xsl:value-of select="publisher" /> </td>

  <td style="color:white;background-color:gray;">

<xsl:value-of select="price" /> </td>

           </tr>

</xsl:for-each>

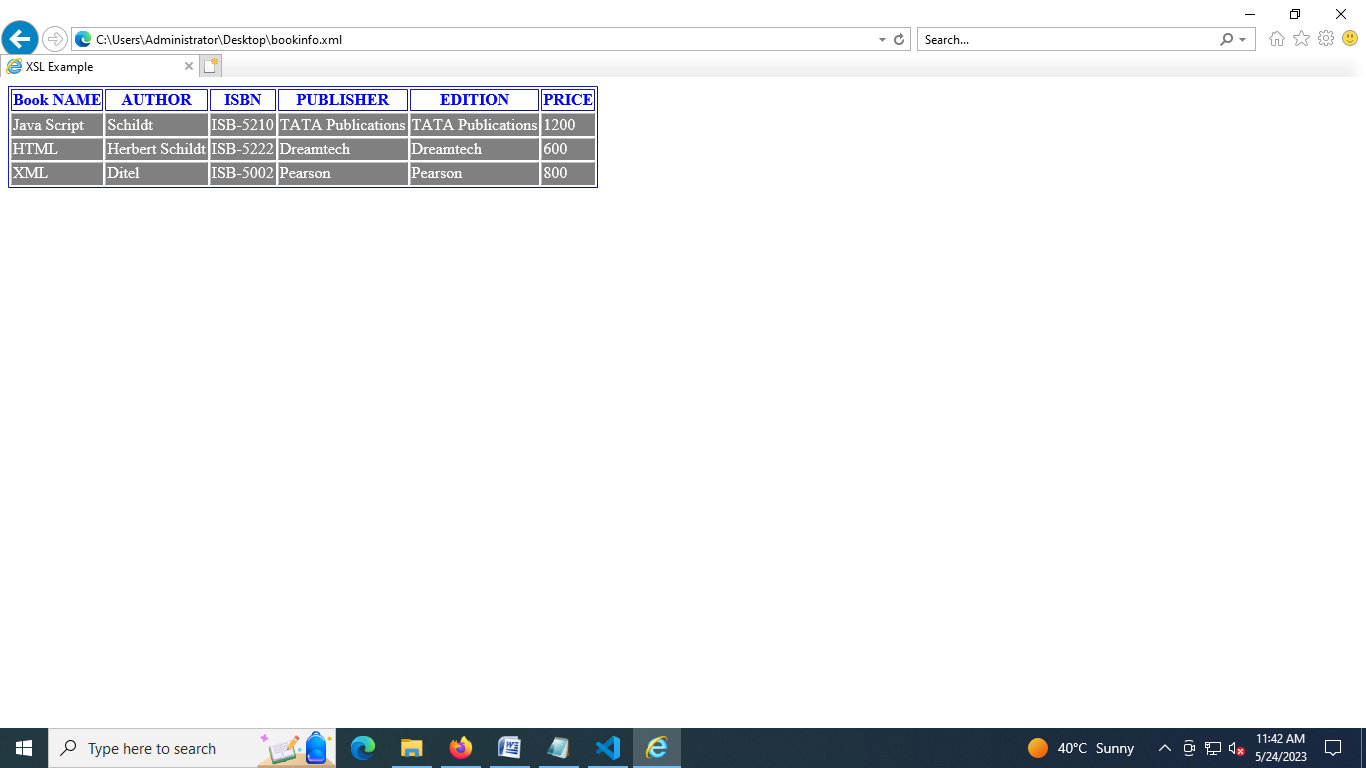
</table>

</body>

</html>

</xsl:template>

</xsl:stylesheet>



1. Design an XML document to store information about a student in an engineering college. The information must include: college id, Name of the College, Brach, Year of Joining, and e- mail id. Make up sample data for 3 students. Create a CSS style sheet and use it to display the document.

Procedure:

* + Create an XML file (e.g., students.xml) using a text editor or an XML editor.
  + Define the structure for student information:
    - Each student should have elements for college ID, college name, branch, year of joining, and email ID.
    - Add sample data for three students.
  + Create a separate CSS file (e.g., style.css) to style the XML content.
  + Define styles for elements within the XML document (e.g., font, color, spacing).
  + In the XML file, add a reference to the CSS stylesheet:
  + Use CSS rules to format the student information.
  + You can style elements like <collegeId>, <collegeName>, etc., based on your design preferences.

Sample Code:

**students.xml**

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

<?xml-stylesheet typ="text/css" href="student.css"?>

<students>

<student>

<sname>k.vinay </sname>

<collegeid>21BQ1A0542</collegeid>

<collegename>VVIT</collegename>

<branch>CSE</branch>

<yearofjoin>2021</yearofjoin>

<email>21BQ1A0542@vvit.net</email>

</student>

<student>

<sname>k.vijay </sname>

<collegeid>21BQ1A0522</collegeid>

<collegename>VVIT</collegename>

<branch>CSE</branch>

<yearofjoin>2021</yearofjoin>

<email>21BQ1A0522@vvit.net</email>

</student>

<student>

<sname>vinod </sname>

<collegeid>21BQ1A0512</collegeid>

<collegename>VVIT</collegename>

<branch>CSE</branch>

<yearofjoin>2021</yearofjoin>

<email>21BQ1A0512@vvit.net</email>

</student>

</students>

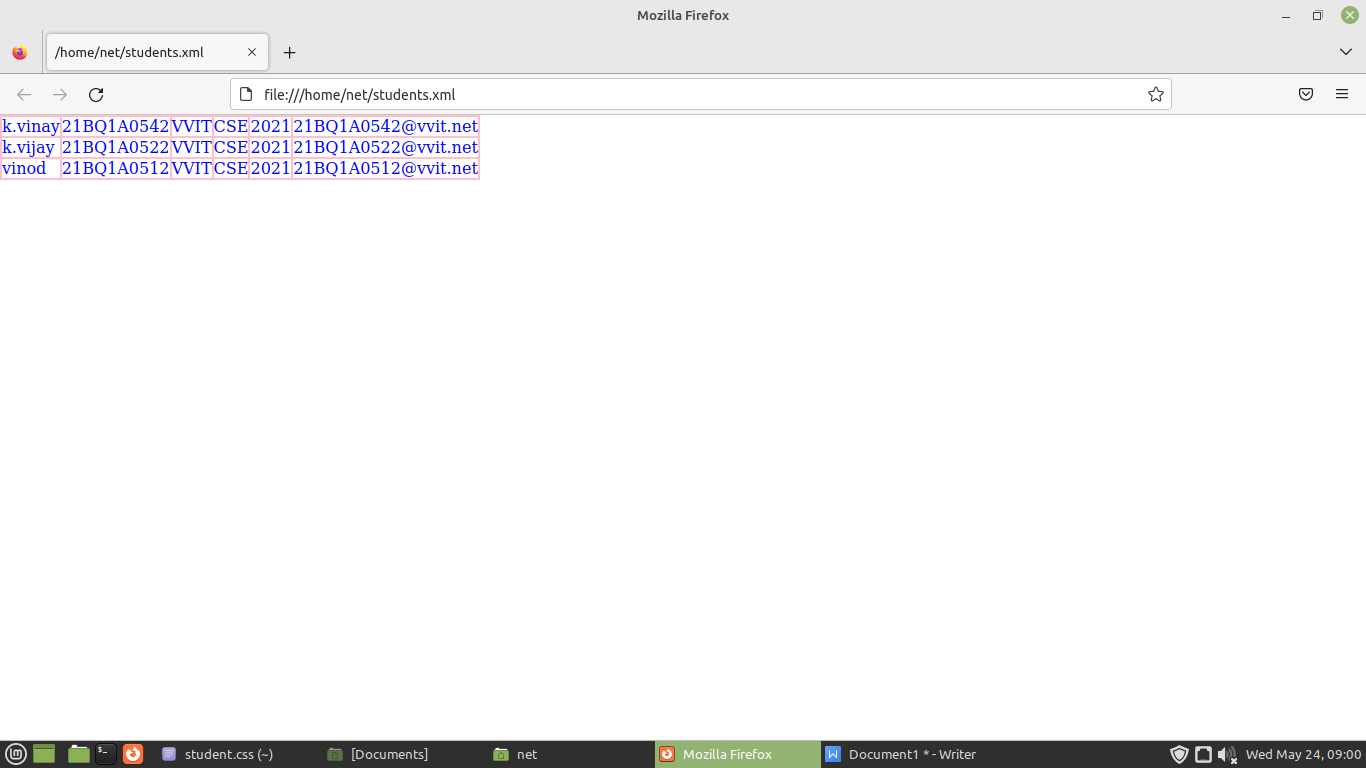
**student.css**

students{display:table;border:solid 1px pink;}

student{display:table-row;border:solid 1px pink;color:blue;}

student{display:table-row;border:solid 1px pink;color:blue;}

sname,collegeid,collegename,branch,yearofjoin,email{display:table-cell;border:solid 1px pink;color:blue;}



1. Write a program to demonstrate XML SAX Parser.

Procedure:

* + The Simple API for XML (SAX) is an event-driven, serial access parser for reading and processing XML files.
  + Unlike DOM (Document Object Model), which creates an internal representation of the entire XML tree, SAX reads the XML sequentially and triggers events for specific elements or text.
  + Create an instance of the SAX parser.
  + Set up event handlers to handle different events (e.g., start of an element, end of an element, character data).
  + Provide an XML file to the SAX parser.
  + The parser will read the file from start to end, calling specific methods for each encountered element or text.
  + Implement event handlers (callbacks) for various SAX events:
    - startDocument(): Called at the start of the XML document.
    - endDocument(): Called at the end of the XML document.
    - startElement(): Called when an opening tag is encountered.
    - endElement(): Called when a closing tag is encountered.
    - characters(): Called with the text contents between start and end tags.
  + Within the event handlers, process the data as needed.
  + For example, extract information from specific elements or attributes.
  + You can print the data, store it in memory, or perform other actions.
  + Handle exceptions or errors that may occur during parsing (e.g., malformed XML, missing elements).
  + Execute your program.
  + Observe how the SAX parser reads and processes the XML data sequentially.

Sample Code:

SaxpDemo.java

import java.io.File;

import javax.xml.parsers.SAXParser;

import javax.xml.parsers.SAXParserFactory;

import org.xml.sax.Attributes;

import org.xml.sax.SAXException;

import org.xml.sax.helpers.DefaultHandler;

public class SaxpDemo

{

public static void main(String[] args)

{

try {

File inputFile = new File("bookinfo.xml");

SAXParserFactory factory = SAXParserFactory.newInstance();

SAXParser saxParser = factory.newSAXParser();

UserHandler userhandler = new UserHandler();

saxParser.parse(inputFile,userhandler);

} catch (Exception e) {

e.printStackTrace();

}

}

}

class UserHandler extends DefaultHandler

{

boolean bname = false;

boolean bauthor = false;

boolean bisbn = false;

boolean bpublisher = false;

boolean bprice = false;

@Override

public void startElement(String uri, String localName, String qName, Attributes attributes) throws SAXException

{

if (qName.equalsIgnoreCase("bookname"))

{

bname=true;

}

else if(qName.equalsIgnoreCase("author"))

{

bauthor=true;

}

else if(qName.equalsIgnoreCase("isbn"))

{

bisbn=true;

}

else if(qName.equalsIgnoreCase("publisher"))

{

bpublisher=true;

}

else if (qName.equalsIgnoreCase("price"))

{

bprice = true;

}

}

@Override

public void endElement(String uri,String localName, String qName) throws SAXException

{

if (qName.equalsIgnoreCase("book")) {

System.out.println("End Element :" + qName);

}

}

@Override

public void characters(char ch[], int start, int length) throws SAXException {

if (bname) {

System.out.println("Book Name: " + new String(ch, start, length));

bname = false;

} else if (bauthor) {

System.out.println("Author Name: " + new String(ch, start, length));

bauthor = false;

} else if (bisbn) {

System.out.println("ISBN : " + new String(ch, start, length));

bisbn = false;

}

else if (bpublisher) {

System.out.println("Publisher: " + new String(ch, start, length));

bpublisher = false;

}

else if (bprice) {

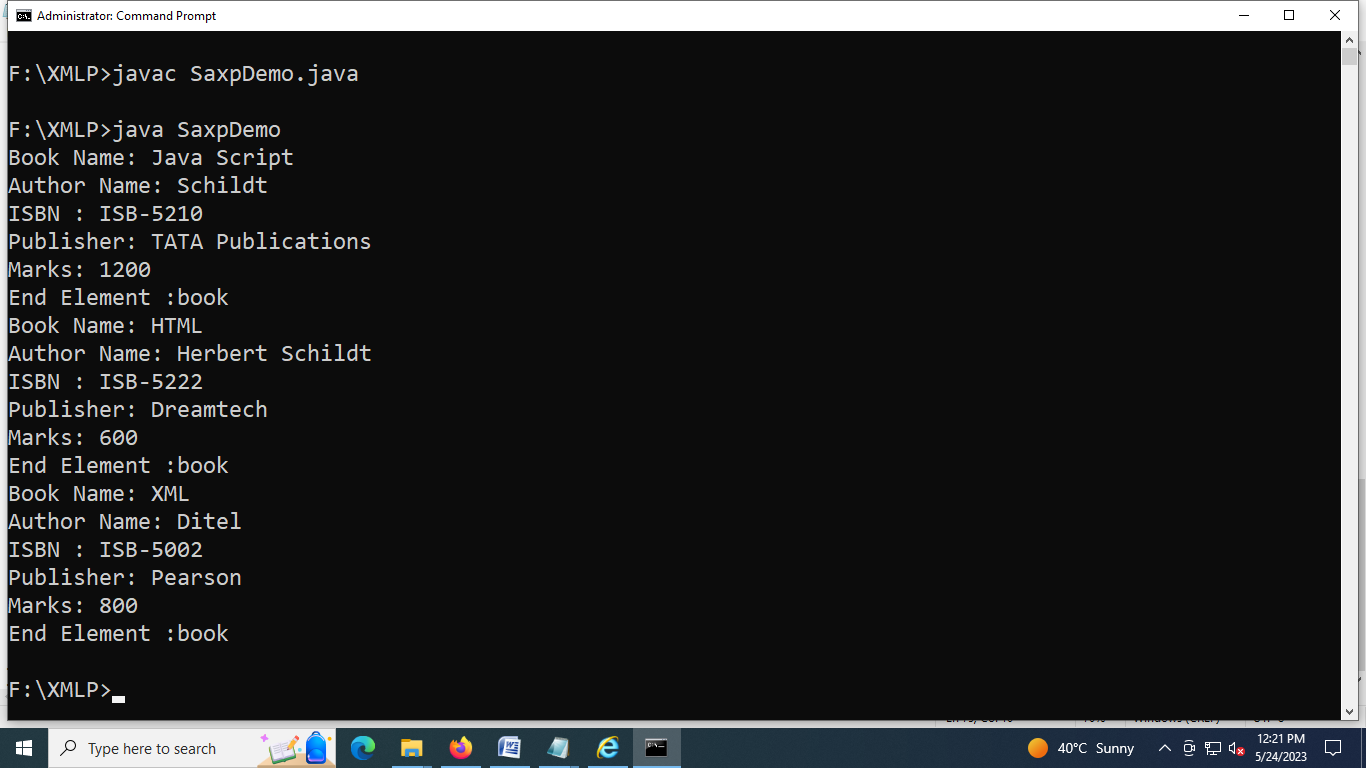
System.out.println("Marks: " + new String(ch, start, length));

bprice = false;

}

}

}

****

1. Write a program to demonstrate XML DOM Parser.

Procedure:

* + The Document Object Model (DOM) is a programming interface for HTML and XML documents.
  + It represents the document as a tree of objects, allowing you to manipulate the content and structure.
  + Create an instance of the DOM parser.
  + Load an XML file into the parser.
  + The DOM parser reads the entire XML document and creates a tree structure in memory.
  + Each element becomes a node in the tree (e.g., elements, attributes, text nodes).
  + Traverse the tree to access specific nodes (e.g., elements, attributes).
  + Extract data from nodes (e.g., get element values, attribute values).
  + Modify nodes (e.g., change text content, add or remove elements).
  + Use the extracted data for your application logic.
  + Perform any necessary operations based on the XML content.
  + Handle exceptions or errors that may occur during parsing (e.g., malformed XML, missing elements).
  + Execute your program.
  + Observe how the DOM parser constructs the tree and allows you to work with XML data.

Sample Code:

import org.w3c.dom.\*;

import javax.xml.parsers.\*;

import java.io.\*;

import org.xml.sax.SAXException;

public class Domj

{

private static final String fname = "bookinfo.xml";

public static void main(String[] args)

{

DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

try {

DocumentBuilder db = dbf.newDocumentBuilder();

Document doc = db.parse(new File(fname));

doc.getDocumentElement().normalize();

System.out.println("Root Element :" + doc.getDocumentElement().getNodeName());

System.out.println("------");

NodeList list = doc.getElementsByTagName("book");

for (int i=0; i < list.getLength(); i++)

{

Node node = list.item(i);

if (node.getNodeType() == Node.ELEMENT\_NODE)

{

Element element = (Element) node;

String bname = element.getElementsByTagName("bookname").item(0).getTextContent();

String auth = element.getElementsByTagName("author").item(0).getTextContent();

String isb = element.getElementsByTagName("isbn").item(0).getTextContent();

String publ= element.getElementsByTagName("publisher").item(0).getTextContent();

String pr= element.getElementsByTagName("price").item(0).getTextContent();

System.out.println("Current Element :" + node.getNodeName());

System.out.println("Name : " + bname);

System.out.println("Author : " + auth);

System.out.println("Isbn : " + isb);

System.out.println("Isbn : " + publ);

System.out.println("price : " + pr);

}

}

}

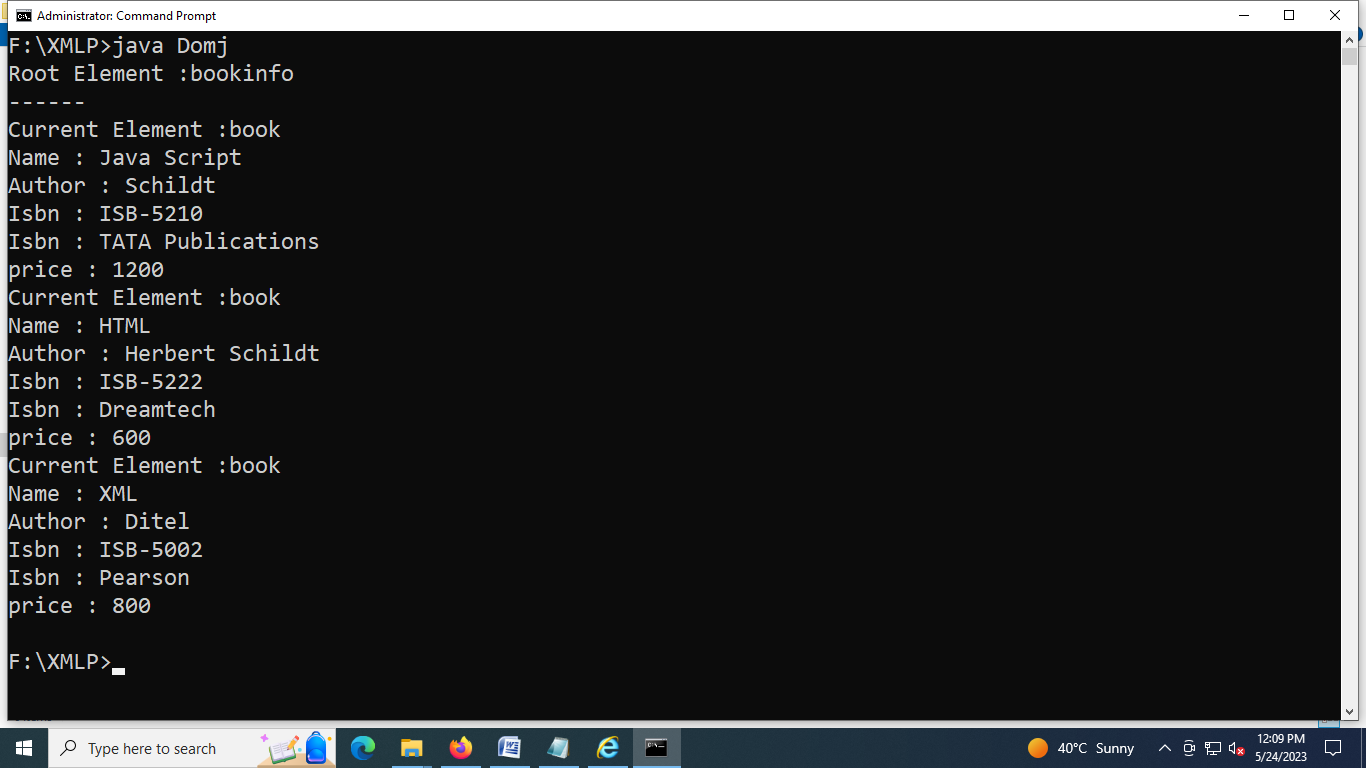
catch(ParserConfigurationException | SAXException| IOException e) {

e.printStackTrace();

}

}

}



1. Create tables in the database which contain the details of items (books in our case Like Book name, Price, Quantity, Amount) of each category.

Modify your catalogue page in such a way that you should connect to the database and extract data from the tables and display them in the catalogue page using JDBC.

Procedure:

* + Ensure you have a database system (e.g., MySQL, PostgreSQL) installed and running.
  + Create tables to store item details (e.g., books).
  + Define columns for book name, price, quantity, and amount.
  + In your Java program (e.g., your catalog page), establish a connection to the database using JDBC.
  + Provide the necessary connection details (URL, username, password).
  + Write SQL queries to retrieve data from the tables.
  + For example:
    - Retrieve book names, prices, quantities, and amounts for each category.
  + Use JDBC to execute the SQL queries against the database.
  + Retrieve the result sets containing the book details.
  + Modify your catalog page to display the retrieved book information.
  + You can format it as a table, list, or any other suitable format.
  + Handle exceptions related to database connectivity, query execution, and data retrieval.
  + Gracefully handle errors to prevent crashes.
  + Run your modified catalog page.
  + Verify that it successfully connects to the database and displays the book details.

Sample Code:

Bookscart.java

import javax.servlet.\*;

import javax.servlet.http.\*;

import java.sql.\*;

import java.io.\*;

import java.lang.\*;

public class Bookscat extends HttpServlet

{

public void doGet(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException

{

PrintWriter out=res.getWriter();

res.setContentType("text/html");

int s,qu;

float pr,am;

String nam;

String cpath="/MyApp/images/cart.png";

try{

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

String ur="jdbc:mysql://localhost:3306/test";

Connection con=DriverManager.getConnection(ur,"root","root");

String q="select \* from books;";

PreparedStatement stmt=con.prepareStatement(q);

ResultSet rs=stmt.executeQuery();

out.println("<html><head><title>Catalogue Page</title></head><body></center><table>");

out.println("<tr><th>Book image</th><th>Name</th><th>Price</th><th>Quantity</th><th>Amount</th><th></th></tr>");

out.println("<h1> Book Catalogue</h1>");

while(rs.next())

{

String imgurl="/MyApp/images/b"+rs.getInt(1)+".jpg";

nam=rs.getString(2);

pr=rs.getFloat(3);

qu=rs.getInt(4);

am=rs.getFloat(5);

out.println("<tr><td><img src="+imgurl+" width=40 height=40></td><td>"+nam+"</td>");

out.println("<td>"+pr+"</td><td>"+qu+"</td><td>"+am+"</td>");

out.println("<td> <button type='button'><img src="+cpath+" width=20 height=20></button></td></tr>");

}

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

stmt.close();

con.close();

out.close();

}

catch(ClassNotFoundException c)

{

out.println(c);

}

catch(SQLException se)

{

out.println(se);

}

}

}

Web.xml

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

<!--

Licensed to the Apache Software Foundation (ASF) under one or more

contributor license agreements. See the NOTICE file distributed with

this work for additional information regarding copyright ownership.

The ASF licenses this file to You under the Apache License, Version 2.0

(the "License"); you may not use this file except in compliance with

the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software

distributed under the License is distributed on an "AS IS" BASIS,

WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and

limitations under the License.

-->

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"

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

xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee

http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"

version="3.1"

metadata-complete="true">

<display-name>Welcome to Tomcat</display-name>

<description>

Welcome to Tomcat

</description>

<servlet>

<servlet-name>Hello</servlet-name>

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

</servlet>

<servlet-mapping>

<servlet-name>Hello</servlet-name>

<url-pattern>/sayhello</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>Registration</servlet-name>

<servlet-class>Register</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>Registration</servlet-name>

<url-pattern>/regis</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>loginv</servlet-name>

<servlet-class>Validate</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>loginv</servlet-name>

<url-pattern>/valid</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>jspserv</servlet-name>

<jsp-file>/Readinit.jsp</jsp-file>

<init-param>

<param-name>userid</param-name>

<param-value>user1</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>jspserv</servlet-name>

<url-pattern>/Readinit.jsp</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>jspcontext</servlet-name>

<jsp-file>/Readcontext.jsp</jsp-file>

</servlet>

<servlet-mapping>

<servlet-name>jspcontext</servlet-name>

<url-pattern>/Readcontext.jsp</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>bookcat</servlet-name>

<servlet-class>Bookscat</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>bookcat</servlet-name>

<url-pattern>/bookscat</url-pattern>

</servlet-mapping>

<context-param>

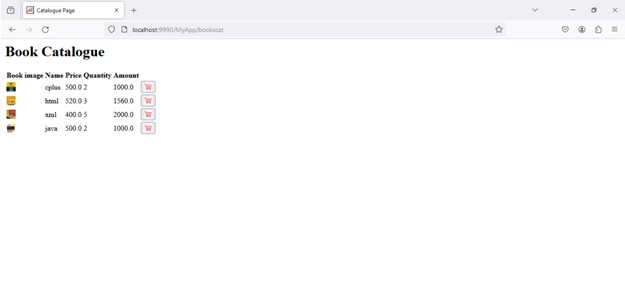
<param-name>userid</param-name>

<param-value>user1</param-value>

</context-param>

</web-app>

Output:



1. Using java servlets and JDBCs to red and retrieve the following information from a database:
2. Name
3. Password
4. Email id
5. Phone number

Procedure:

* + **Java Servlets**: Servlets are Java classes that handle HTTP requests and responses. They run on a web server and provide dynamic content.
  + **JDBC (Java Database Connectivity)**: JDBC is a Java API for connecting to databases, executing SQL queries, and retrieving data.
  + Set up a database system (e.g., MySQL, PostgreSQL).
  + Create a database and tables to store user information (name, password, email, phone number).
  + Create Java servlets to handle HTTP requests related to user data.
  + Implement methods to connect to the database using JDBC.
  + Retrieve user information based on the specified criteria (name, password, email, phone number).
  + Configure servlet mappings in the web.xml file.
  + Define URL patterns that map to your servlets.
  + In your servlets, execute SQL queries to retrieve user data.
  + Use prepared statements to prevent SQL injection.
  + Extract the retrieved data (name, password, email, phone number).
  + Format it as needed (e.g., display in a table, JSON response, etc.).
  + Deploy your servlets on a web server (e.g., Tomcat, Jetty).
  + Access the servlets via URLs to verify that they retrieve the specified information from the database.

Sample Code:

<!DOCTYPE html>

<html>

<head>

<title>Input Form</title>

<style>

body { font-family: Arial, sans-serif; margin: 0; padding: 20px; }

form {

max-width: 400px; margin: 0 auto; padding: 20px; border: 1px solid #ccc;

border-radius: 5px; background-color: #f9f9f9; }

input[type="text"],

input[type="date"],

input[type="submit"] { width: 100%; padding: 10px; margin-bottom: 10px; border: 1px solid #ccc;

border-radius: 5px; box-sizing: border-box; }

input[type="submit"] { background-color: #4CAF50; color: white; cursor: pointer; }

input[type="submit"]:hover {background-color: #45a049; }

.output { margin-top: 20px; padding: 10px; border: 1px solid #ccc; border-radius: 5px;

background-color: #fff; }

</style>

</head>

<body>

<p align="center" style="color:green;font-size:50px;">Student Info Form</p>

<form id="f1" onsubmit="return displayInfo()">

<label for="firstName">First Name:</label>

<input type="text" id="firstName" name="firstName" required>

<label for="lastName">Last Name:</label>

<input type="text" id="lastName" name="lastName" required>

<label for="dob">Date of Birth:</label>

<input type="date" id="dob" name="dob" required>

<input type="submit" value="Submit">

</form>

<div id="block1" class="output" style="display: none;">

<h2>User Information</h2>

<p><strong>First Name:</strong> <span id="FName"></span></p>

<p><strong>Last Name:</strong> <span id="LName"></span></p>

<p><strong>Date of Birth:</strong> <span id="PDOB"></span></p>

</div>

<script>

function displayInfo() {

var firstName = document.getElementById('firstName').value;

var lastName = document.getElementById('lastName').value;

var dob = document.getElementById('dob').value;

document.getElementById('FName').innerText = firstName;

document.getElementById('LName').innerText = lastName;

document.getElementById('PDOB').innerText = dob;

document.getElementById('block1').style.display = 'block';

return false;

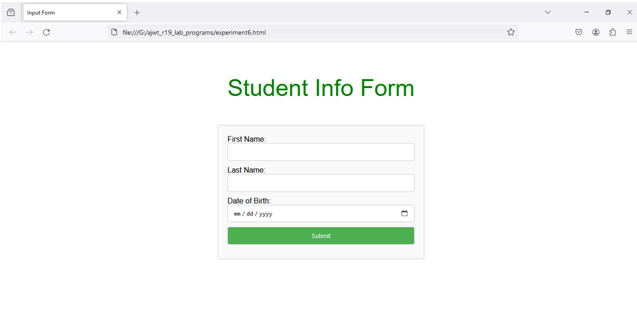
}

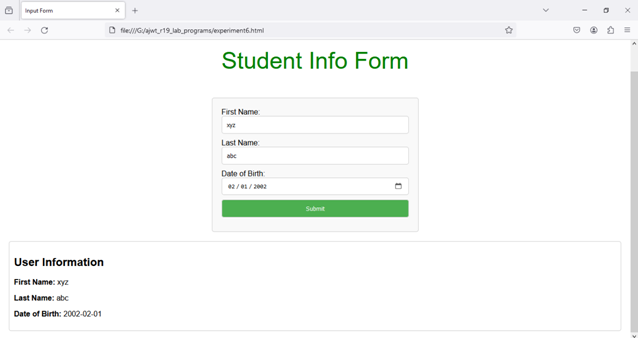
</script>

</body>

</html>

Output:





1. Demonstrate Cookie and Session Managementin Servlets.

Procedure:

* + **Cookies?**:
    - Cookies are small pieces of data stored on the client-side (browser) by the server.
    - They contain information such as user preferences, session IDs, or tracking data.
    - Cookies are used to maintain state between multiple requests from the same client.
  + **Types of Cookies**:
    - **Non-persistent (Session) Cookies**:
      * Valid for a single session (until the browser is closed).
      * Removed when the user closes the browser.
    - **Persistent Cookies**:
      * Valid across multiple sessions.
      * Not removed when the browser is closed.
      * Typically used for features like “Remember Me.”
  + **Advantages of Cookies**:
    - Simplest technique for maintaining state.
    - Cookies are stored at the client-side.
    - Useful for personalized experiences (e.g., remembering user preferences).
  + **Disadvantages of Cookies**:
    - Won’t work if cookies are disabled in the browser.
    - Limited to storing textual information.
    - Security concerns (e.g., sensitive data in cookies).
  + Sessions are used to maintain state across multiple requests.
  + A session is a logical connection between the client and server.
  + Servlets use the HttpSession interface for session management.
  + Store session-specific data using session attributes (e.g., user authentication, shopping carts).
  + Test your servlets with different browsers and cookie settings.
  + Verify that session attributes persist across requests.

Sample Code:

**Cookiev.java**

import javax.servlet.\*;

import javax.servlet.http.\*;

import java.sql.\*;

import java.io.\*;

import java.lang.\*;

public class Cookiev extends HttpServlet

{

public void doPost(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException

{

    PrintWriter out=res.getWriter();

    res.setContentType("text/html");

    String usr=req.getParameter("uid");

    String pw=req.getParameter("pwd");

    int s=0;

    Cookie c = null;

    Cookie[] cookies = null;

    cookies=req.getCookies();

    if(cookies!=null)

    {

        for (int i = 0; i < cookies.length; i++)

        {

            c=cookies[i];

           String u=c.getName();

           String v=c.getValue();

             if( u.equals(usr)&&v.equals(pw))

               {

                out.print(" Login success..Welcome user...<br/>");

                s=1;

                break;

            }

        }

        if(s==0)

          out.print(" Login failed.. check userid and password.....<br/>");

    }

    else

    {

        Cookie c1=new Cookie("user1","pwd1");

        Cookie c2=new Cookie("user2","pwd2");

        res.addCookie(c1);

        res.addCookie(c2);

        if((usr.equals(c1.getName())&&pw.equals(c1.getValue()))||(usr.equals(c2.getName())&&pw.equals(c2.getValue())))

        out.print(" Login success..Welcome user...<br/>"+usr);

        else

        out.print(" Login failed..check userid and password...<br/>");

    }

}

}

Loginc.html

<!DOCTYPE html>

<html>

  <head>

      <title>Login Form</title>

   </head>

<body>

          <form name="f" method="post" action="http://localhost:9990/myproject/loginc">

        <table align="center">

            <caption>LOGIN HERE</caption>

            <tr><td> user name....:</td><td><input type = "text" name = "uid"></td></tr>

            <tr><td> pass word....:</td><td><input type = "password" name = "pwd"></td></tr>

            <tr><td></td><td><input type = "submit" name = "submit" value = "Login">

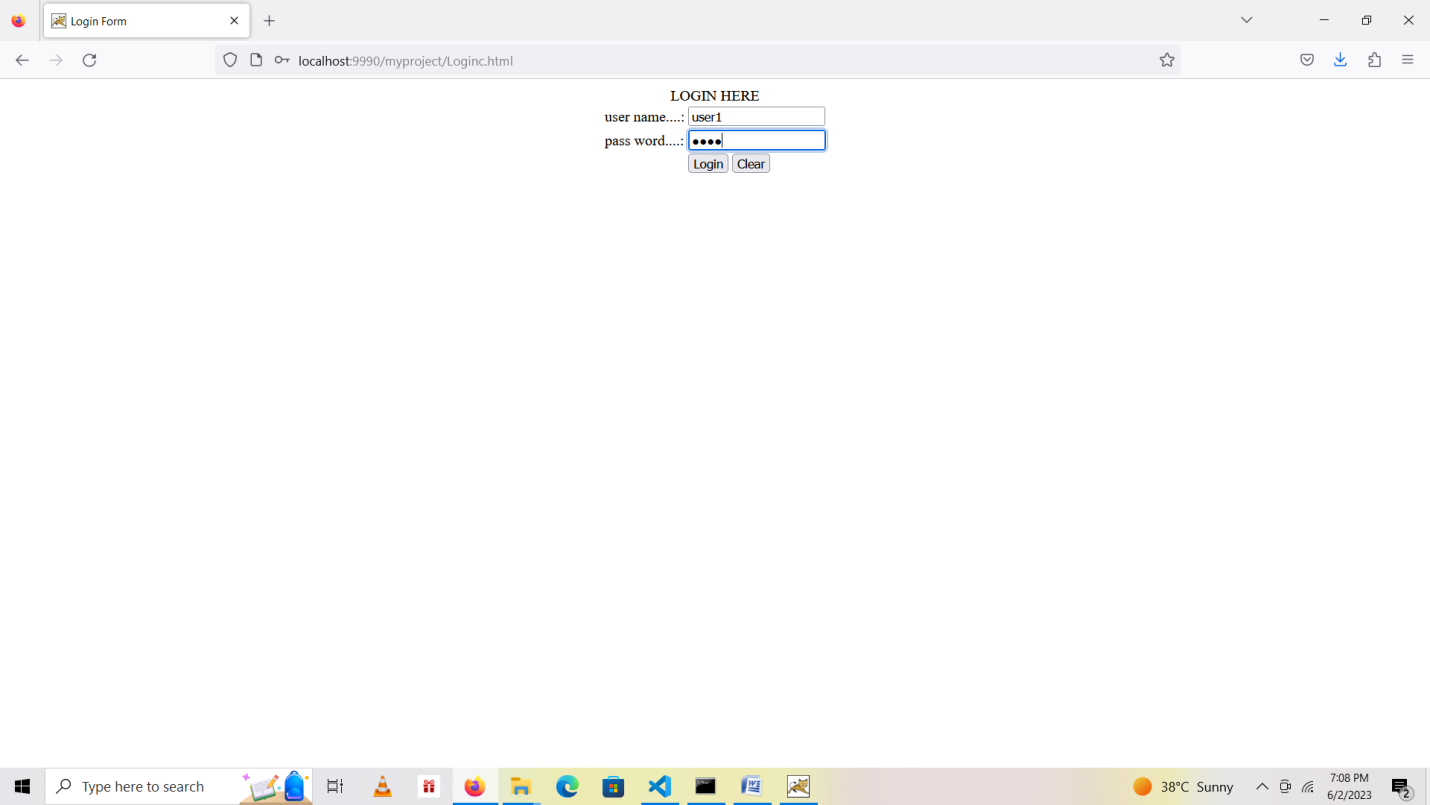
                <input type = "reset" name = "reset"  value = "Clear" > </td></tr>

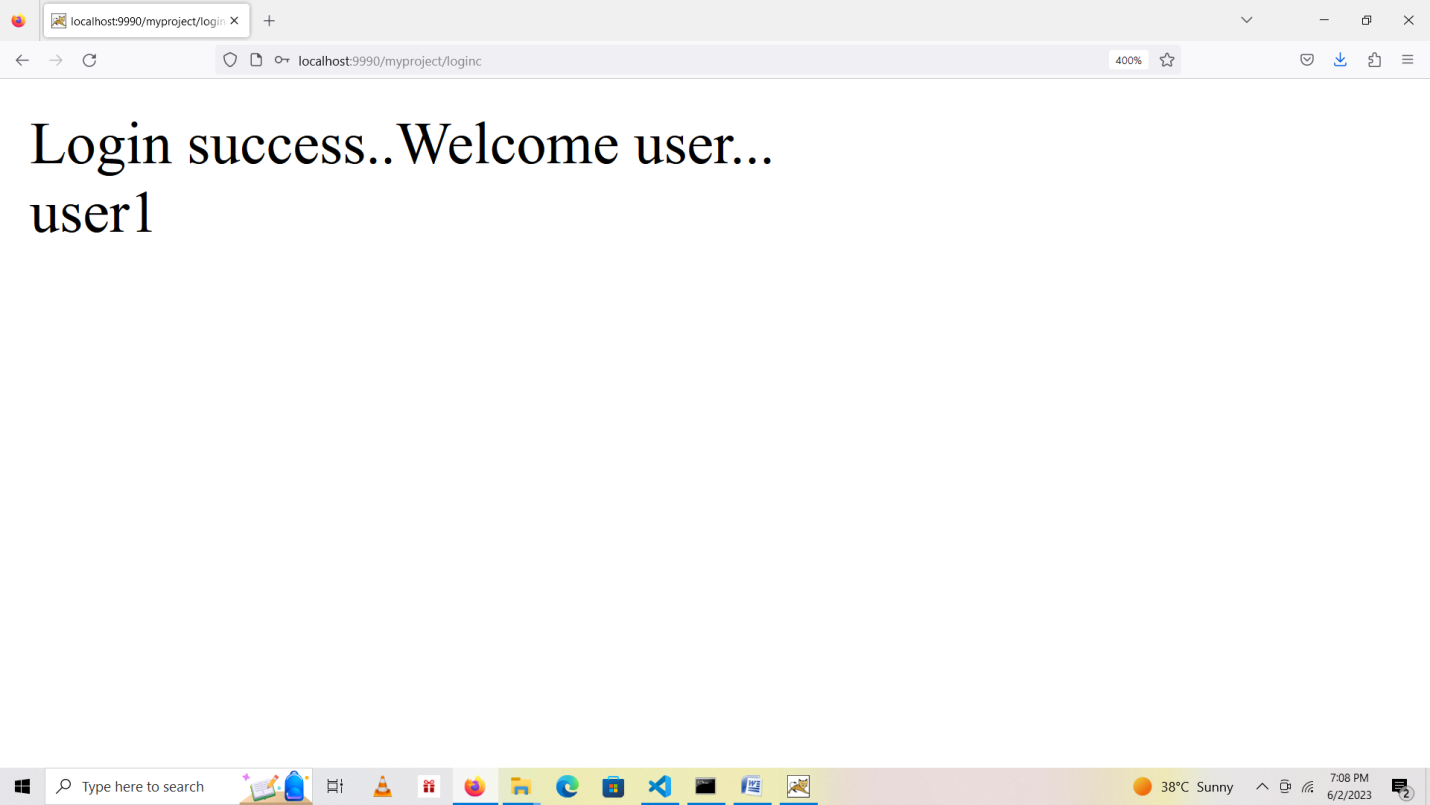
        </table>

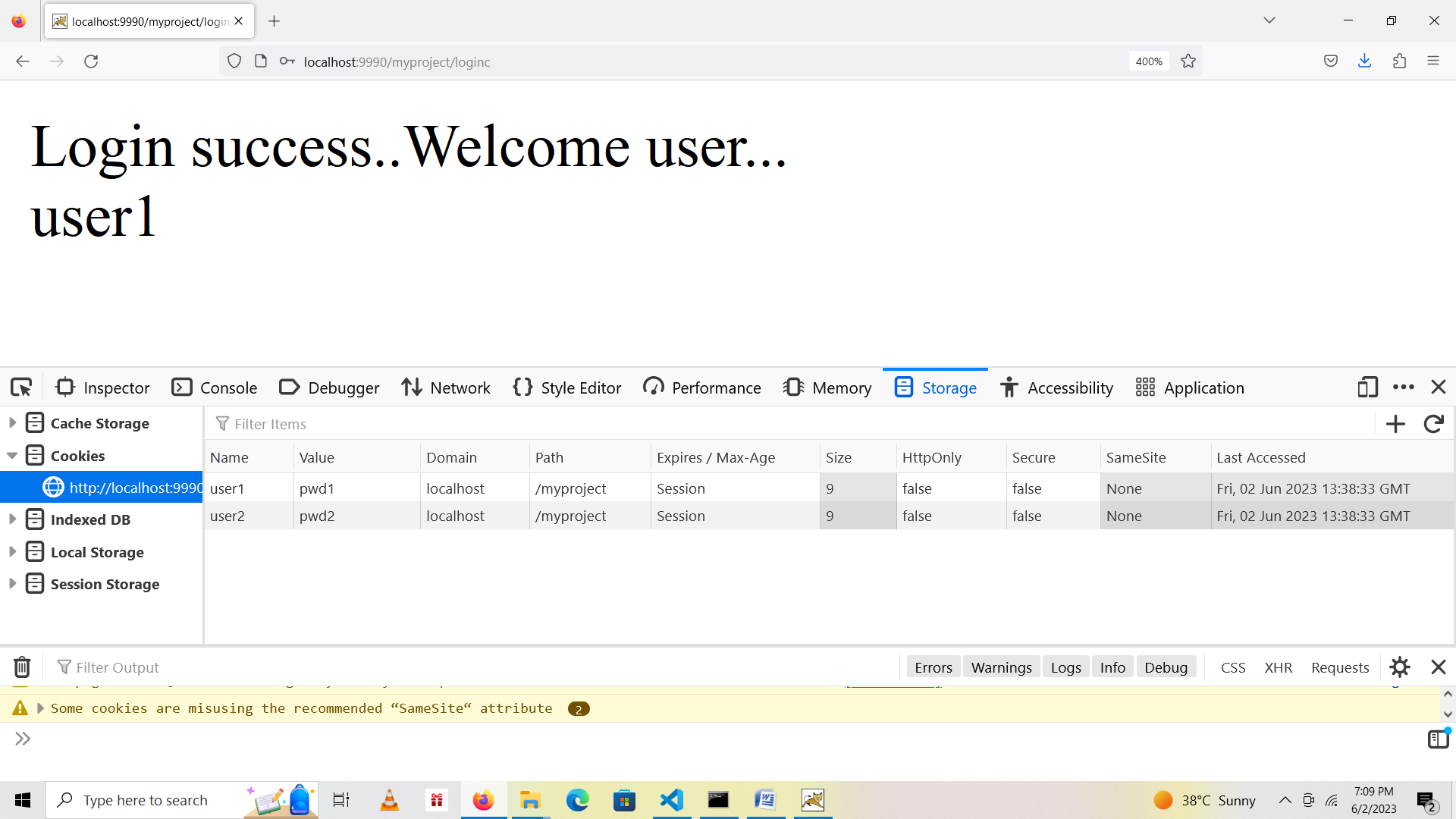
     </form>

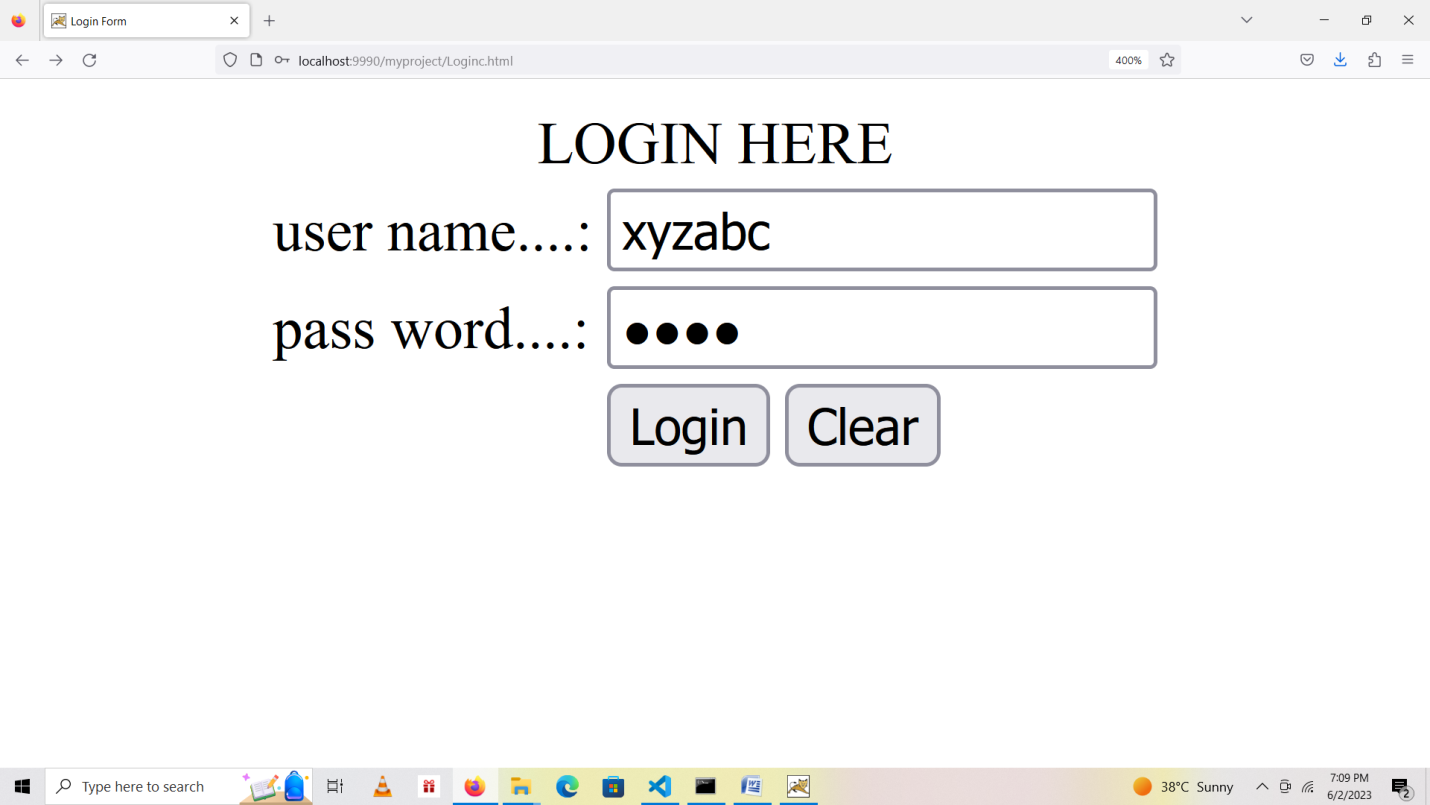
 </body>

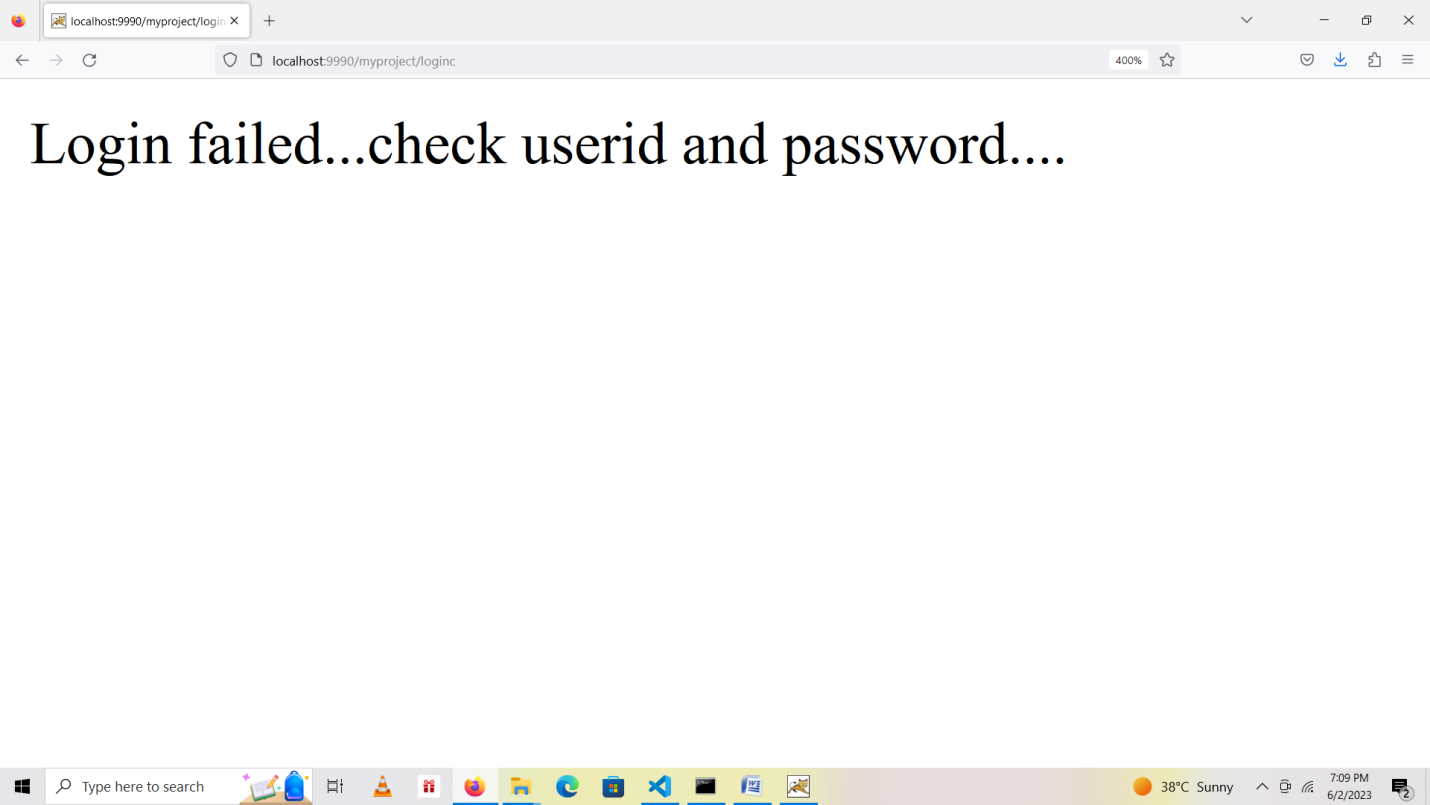
</html>











Accessing user data using session object in multiple servlets

SessionServ.java

import javax.servlet.\*;

import javax.servlet.http.\*;

import java.sql.\*;

import java.io.\*;

import java.lang.\*;

public class SessionServ extends HttpServlet

{

public void doPost(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException

{

    PrintWriter out=res.getWriter();

    res.setContentType("text/html");

    HttpSession se=req.getSession(true);

    String usr=req.getParameter("uid");

    se.setAttribute("username",usr);

    out.println("<br> redirecting..<br>");

    RequestDispatcher re=req.getRequestDispatcher("welc");

    re.forward(req,res);

   }

}

SessionMsg.java

import javax.servlet.\*;

import javax.servlet.http.\*;

import java.sql.\*;

import java.io.\*;

import java.lang.\*;

public class SessionMsg extends HttpServlet

{

public void doPost(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException

{

    PrintWriter out=res.getWriter();

    res.setContentType("text/html");

    HttpSession se=req.getSession();

    String uname=(String)se.getAttribute("username");

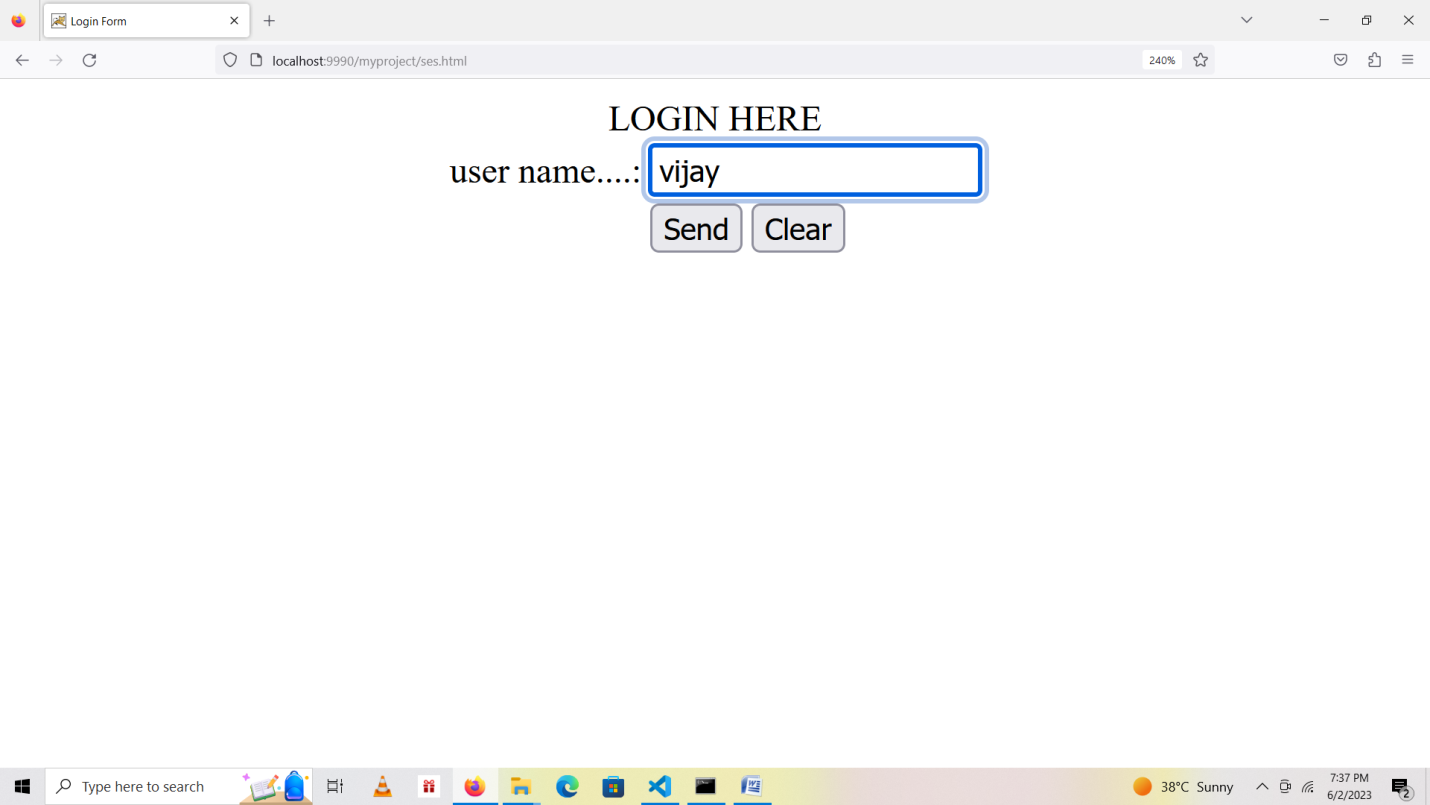
    out.println("<br> welcome user.."+uname);

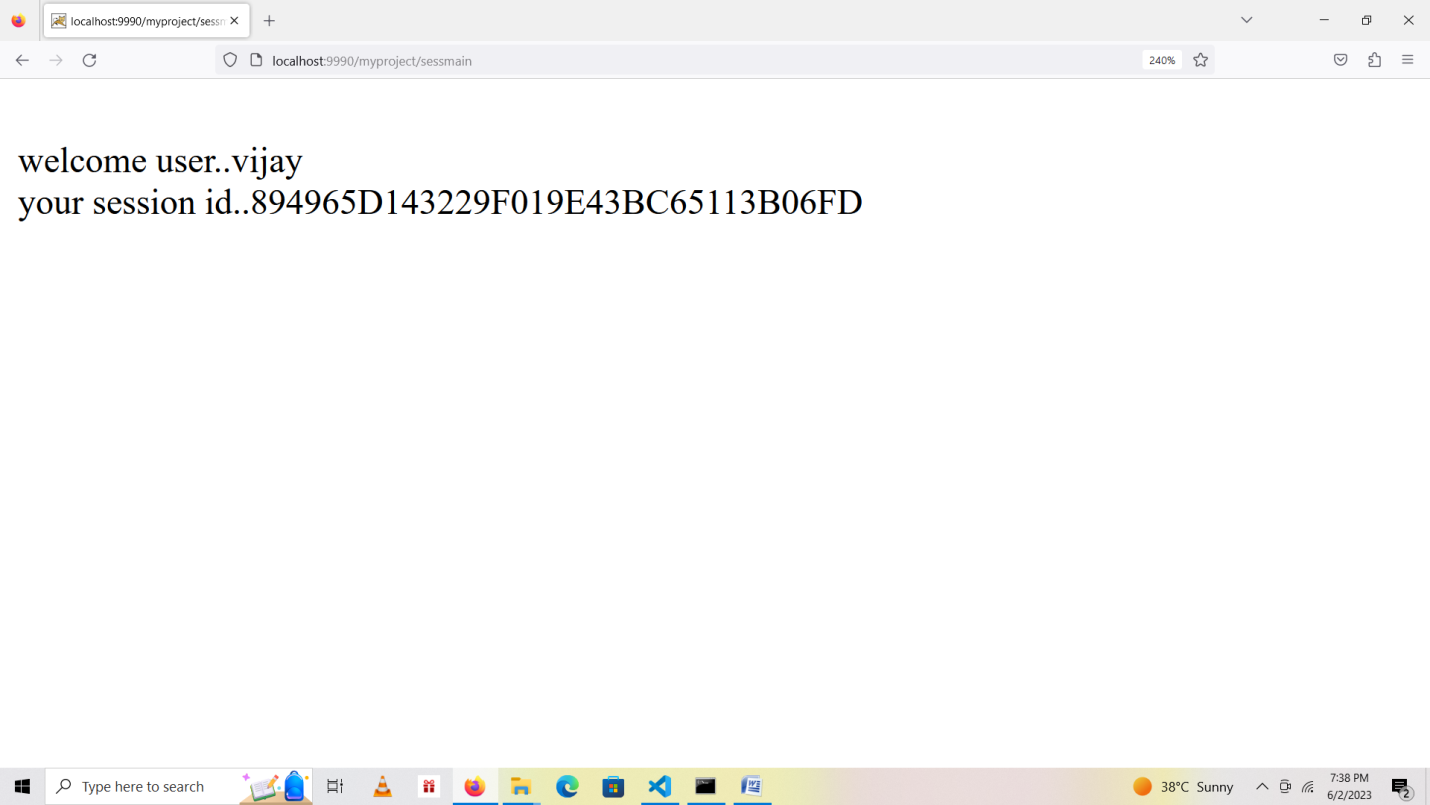
    out.println("<br> your session id.."+se.getId());

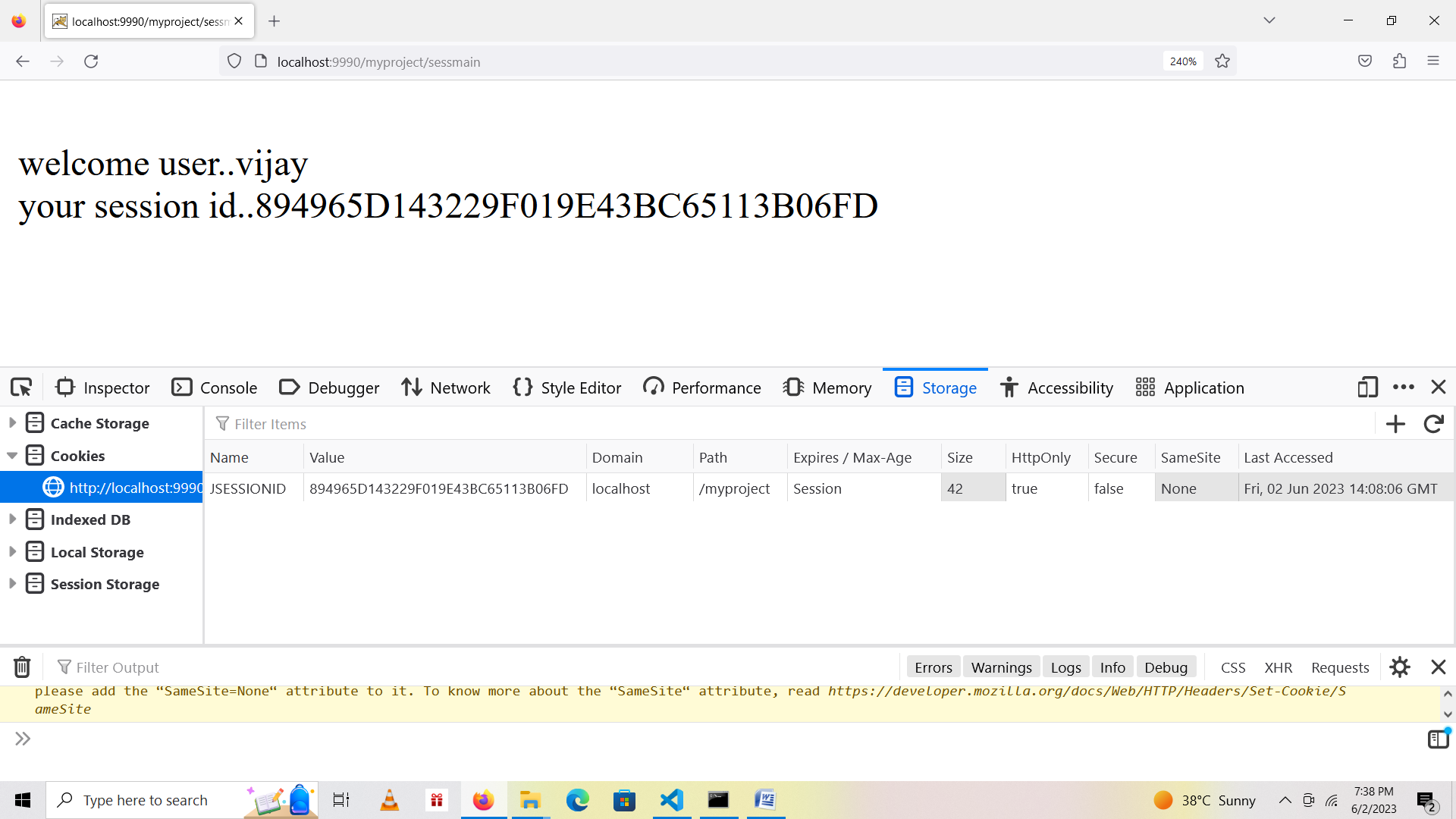
    out.close();

       }

}







1. Write a program to demonstrate Java Bean using JSP Implicit objects.

Procedure:

* + Java Beans are reusable software components written in Java.
  + They follow specific conventions (e.g., having a no-argument constructor, providing getter and setter methods) to be easily integrated into applications.
  + Java Beans can be used for various purposes (e.g., data storage, business logic, UI components).
  + JSP provides several built-in objects (implicit objects) that are automatically available in JSP pages.
  + These objects are created by the web container and are accessible without explicit declaration.
  + Examples of implicit objects include out, request, session, application, etc.
  + Create a Java Bean class (e.g., BookBean) that encapsulates book-related data (e.g., book name, price, quantity).
  + Set up getter and setter methods for each property.
  + In your JSP page, use implicit objects (e.g., request, session) to interact with the Java Bean:
    - Set Java Bean properties using request.setAttribute("bookBean", bookBeanInstance).
    - Retrieve Java Bean properties using ${bookBean.propertyName}.

Sample Code:

I. <jsp:useBean>

1. create a java bean
2. compile the java bean
3. go to C:\Program Files\Apache Software Foundation\Tomcat 9.0\webapps\myproject\WEB-INF\classes

and create a folder with the name same as package name present in java bean

ex:

C:\Program Files\Apache Software Foundation\Tomcat 9.0\webapps\myproject\WEB-INF\classes\dateb

Copy java bean class file in to the above folder.

1. Set class path to the folder that contains java bean class file

Ex:

C:\Program Files\Apache Software Foundation\Tomcat 9.0\webapps\myproject\WEB-INF\classes\dateb

Datebean.java

package dateb;

import java.util.\*;

public class Datebean

{

Date d=new Date();

public Date getDate()

{

return d;

}

public void setDate(Date t)

{

this.d=t;

}

}

1. Write jsp code and import javabean

Usebean.jsp

<%@ page import="dateb.Datebean,java.util.\*"%>

<html>

<head>

<title>Usebean</title>

</head>

<body>

<h1 align="center">Usebean Example</h1>

<b> <jsp:useBean id="datebean" class="dateb.Datebean"/>

<% Date dt=datebean.getDate(); %>

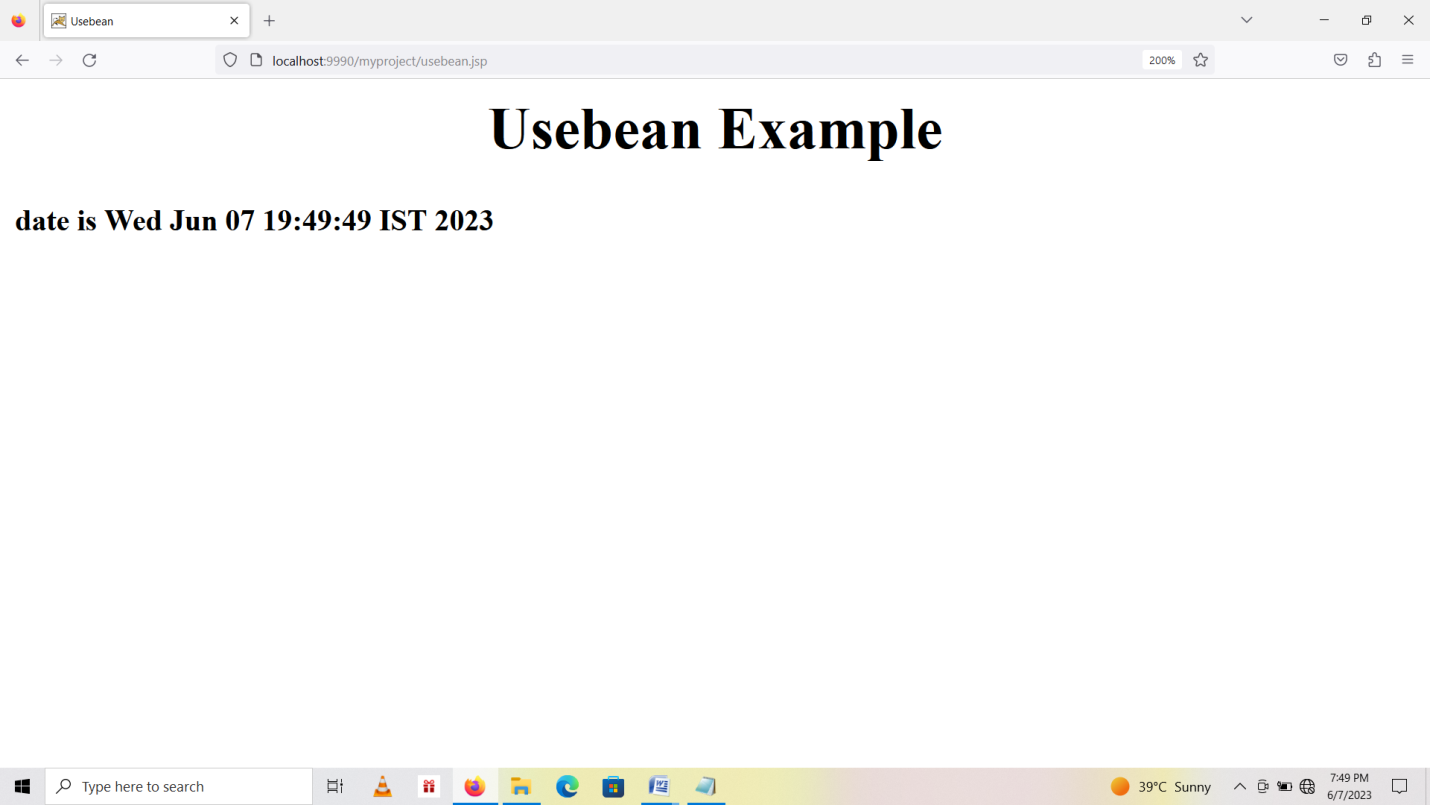
<%= "date is "+dt %>

</b>

</body>

</html>

Output:



1. Write a JSP program to conduct online examination and to display student mark list available in a database.

Procedure:

* + Set up a database system (e.g., MySQL, PostgreSQL).
  + Create tables to store student information (e.g., roll number, name, marks).
  + Create JSP pages for conducting online examinations:
    - Design an interface for students to log in and take exams.
    - Display questions, collect answers, and calculate scores.
    - Store student responses in the database.
  + Retrieve student marks from the database.
  + Display the mark list on a separate JSP page.
  + Include details such as roll number, name, and total marks.
  + Test your JSP pages by simulating student logins, exam attempts, and mark calculations.
  + Verify that student data is correctly stored and displayed.

Sample Code:

**onlinequiz.html**

<!DOCTYPE html>

<html>

   <head>

      <title>Online Quiz</title>

   </head>

    <body>

    <center>

        <h1>Quiz 1</h1>

        <p>Subject: Advanced Java Programming</p>

        <p><b><i>Answer all the Questions</i></b></p>

        <p>Each question carry 1 mark</p>

      <form name="examform" method="post" action="/MyApp/evaluate.jsp">

        Enter your roll no..:<input type="text" name="rollno">

        <table>

            <tr><td>1</td><td>HTML stands for?</td></tr>

            <tr><td></td><td><input type = "radio" name = "q1" value="o1">High Text Machine Language </td></tr>

            <tr><td></td><td><input type = "radio" name = "q1" value="o2">Hyper Text Machine Language </td></tr>

            <tr><td></td><td><input type = "radio" name = "q1" value="o3">Hyper Text Markup Language </td></tr>

            <tr><td></td><td><input type = "radio" name = "q1" value="o4">None</td></tr>

            <tr><td>2</td><td>The correct sequence of HTML tags for starting a webpage is?</td></tr>

            <tr><td></td><td><input type = "radio" name = "q2" value="o1">Head,Title,HTML,body</td></tr>

            <tr><td></td><td><input type = "radio" name = "q2" value="o2">HTML,Body,Title,Head</td></tr>

            <tr><td></td><td><input type = "radio" name = "q2" value="o3">HTML,Head,Title,Body</td></tr>

            <tr><td></td><td><input type = "radio" name = "q2" value="o4">HTML, Head, Title, Body</td></tr>

            <tr><td>3</td><td>Which of the following element is responsible for making the text bold in HTML?</td></tr>

            <tr><td></td><td><input type = "radio" name = "q3" value="o1"> &lt;pre&gt;</td></tr>

            <tr><td></td><td><input type = "radio" name = "q3" value="o2">&lt;b&gt;</td></tr>

            <tr><td></td><td><input type = "radio" name = "q3" value="o3">&lt;a&gt;</td></tr>

            <tr><td></td><td><input type = "radio" name = "q3" value="o4">&lt;br&gt;</td></tr>

            <tr><td>4</td><td>Which of the following tag is used for inserting the largest heading in HTML?</td></tr>

            <tr><td></td><td><input type = "radio" name = "q4" value="o1"> &lt;h3&gt;</td></tr>

            <tr><td></td><td><input type = "radio" name = "q4" value="o2">&lt;h1&gt;</td></tr>

            <tr><td></td><td><input type = "radio" name = "q4" value="o3">&lt;h5&gt;</td></tr>

            <tr><td></td><td><input type = "radio" name = "q4" value="o4">&lt;h6&gt;</td></tr>

            <tr><td>5</td><td>Which of the following tag is used to insert a line-break in HTML?</td></tr>

            <tr><td></td><td><input type = "radio" name = "q5" value="o1"> &lt;br&gt;</td></tr>

            <tr><td></td><td><input type = "radio" name = "q5" value="o2">&lt;h1&gt;</td></tr>

            <tr><td></td><td><input type = "radio" name = "q5" value="o3">&lt;b&gt;</td></tr>

            <tr><td></td><td><input type = "radio" name = "q5" value="o4">&lt;i&gt;</td></tr>

            <tr><td>6</td><td>How to create an ordered list (a list with the list items in numbers) in HTML?</td></tr>

            <tr><td></td><td><input type = "radio" name = "q6" value="o1"> &lt;ol&gt;</td></tr>

            <tr><td></td><td><input type = "radio" name = "q6" value="o2">&lt;ul&gt;</td></tr>

            <tr><td></td><td><input type = "radio" name = "q6" value="o3">&lt;b&gt;</td></tr>

            <tr><td></td><td><input type = "radio" name = "q6" value="o4">&lt;i&gt;</td></tr>

            <tr><td>7</td><td>Which of the following tag is used to define options in a drop-down selection list?</td></tr>

            <tr><td></td><td><input type = "radio" name = "q7" value="o1"> &lt;list&gt;</td></tr>

            <tr><td></td><td><input type = "radio" name = "q7" value="o2">&lt;select&gt;</td></tr>

            <tr><td></td><td><input type = "radio" name = "q7" value="o3">&lt;dropdown&gt;</td></tr>

            <tr><td></td><td><input type = "radio" name = "q7" value="o4">&lt;i&gt;</td></tr>

            <tr><td>8</td><td>The  &lt;hr&gt; tag in HTML is used for </td></tr>

            <tr><td></td><td><input type = "radio" name = "q8" value="o1"> under line</td></tr>

            <tr><td></td><td><input type = "radio" name = "q8" value="o2">vertical line</td></tr>

            <tr><td></td><td><input type = "radio" name = "q8" value="o3">horizontal line</td></tr>

            <tr><td></td><td><input type = "radio" name = "q8" value="o4">column line</td></tr>

            <tr><td></td><td></td></tr>

            <tr><td></td><td></td></tr>

            <tr><td></td><td><input type="submit" name="submit" value="Submit">

            <input type="reset" name="reset"  value="Clear"> </td></tr>

        </table>

        </form>

    </center>

   </body>

</html>

**evaluate.jsp**

<%@ page import="java.io.\*,java.sql.\*,java.util.\*" %>

<html>

<head>

<title>Database Connectivity</title>

</head>

<body style="color:aqua">

<center>

<%!

String dr="com.mysql.jdbc.Driver";

String ur="jdbc:mysql://localhost:3306/test";

String q="select \* from answers";

String q2="insert into result values(?,?)";

String user="root";

String pwd="root";

String param,value,qid,ans,rollno;

int marks=0,status;

Object obji=null;

%>

<%

try{

Class.forName(dr);

Connection con=DriverManager.getConnection(ur,user,pwd);

Statement stmt=con.createStatement();

PreparedStatement pst=con.prepareStatement(q2);

ResultSet rst=stmt.executeQuery(q);

Enumeration en=request.getParameterNames();

Object obji=en.nextElement();

param=(String)obji;

value=request.getParameter(param);

rollno=value;

while(en.hasMoreElements() && rst.next())

{

obji=en.nextElement();

param=(String)obji;

value=request.getParameter(param);

qid=rst.getString("qid");

ans=rst.getString("answer");

if(param.equals(qid)&&value.equals(ans))

marks=marks+1;

}

pst.setString(1,rollno);

pst.setInt(2,marks);

pst.executeUpdate();

out.println("marks obtained ...for roll no"+rollno+" is"+marks);

rst.close();

stmt.close();

pst.close();

con.close();

}

catch(ClassNotFoundException c)

{

out.println(c);

}

catch(SQLException s)

{

out.println(s);

}

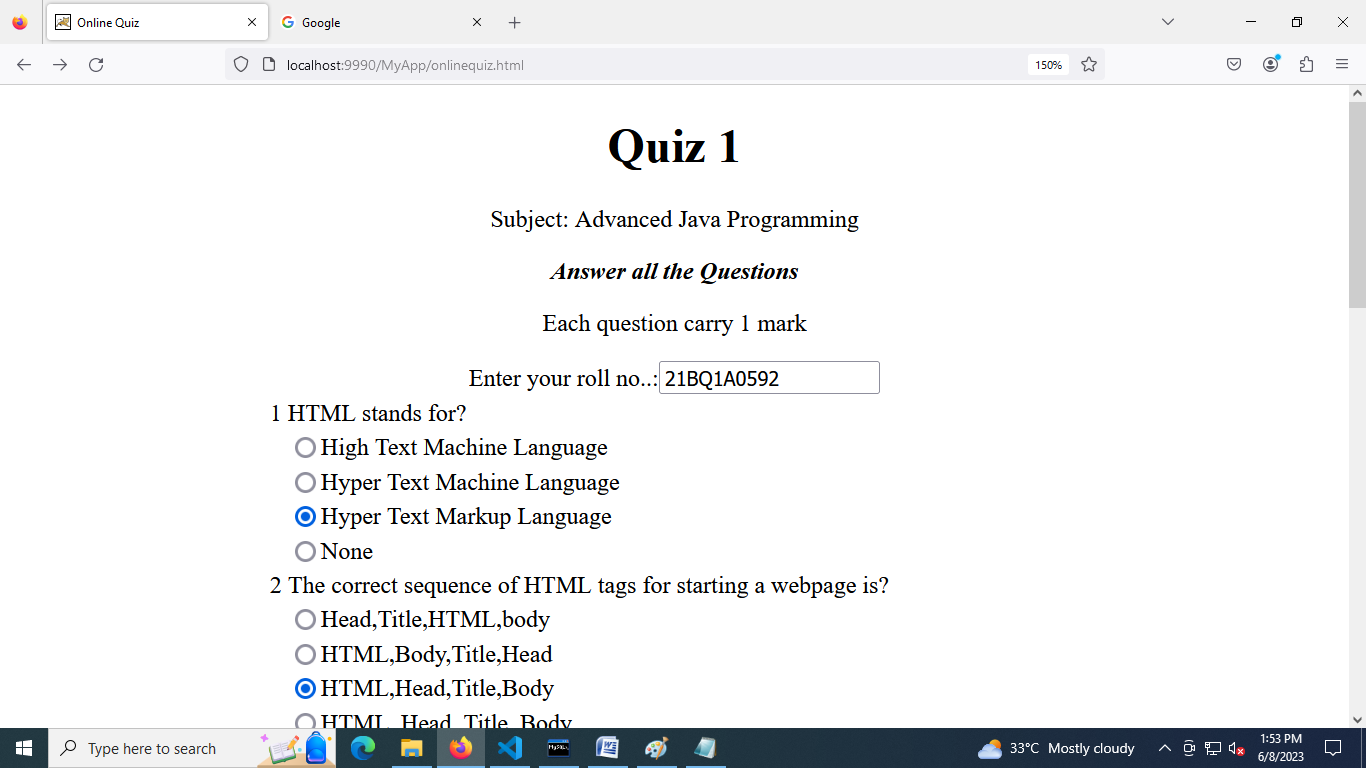
%>

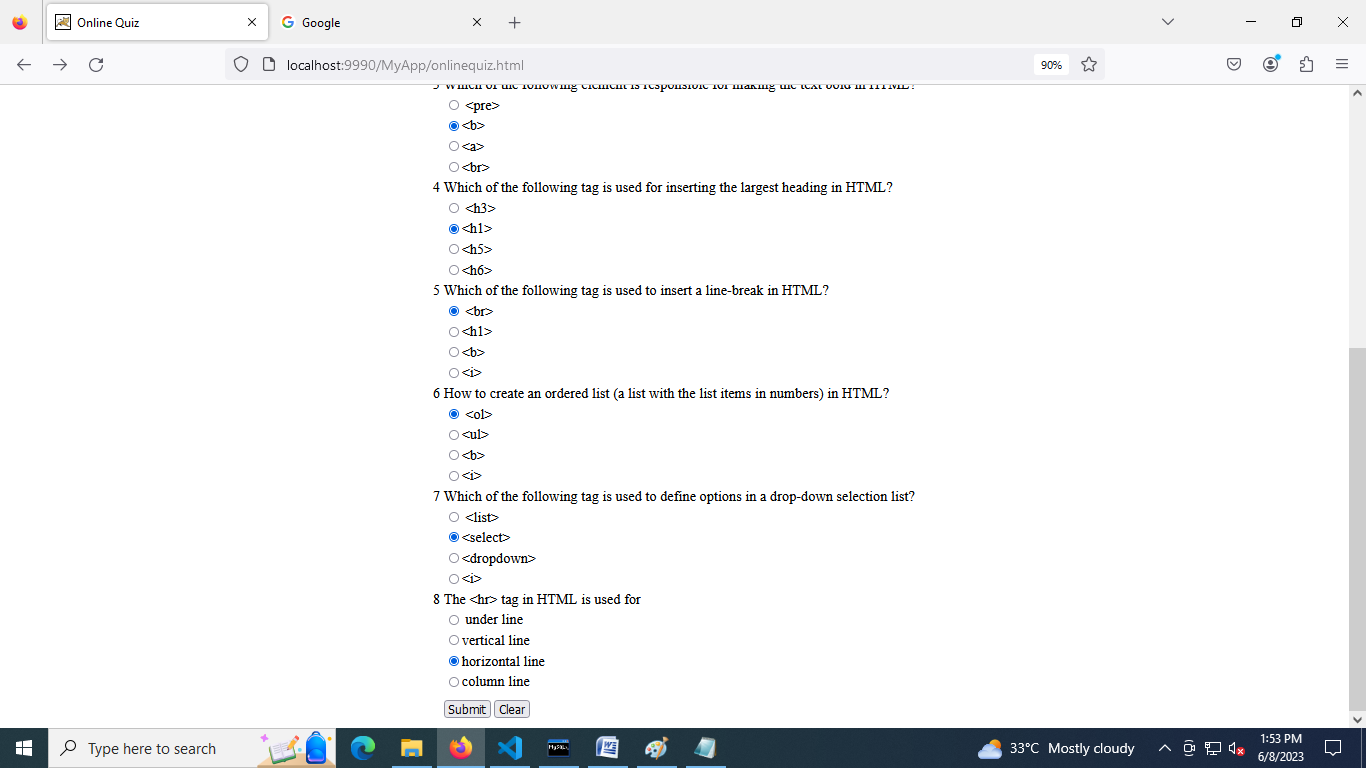
</center>

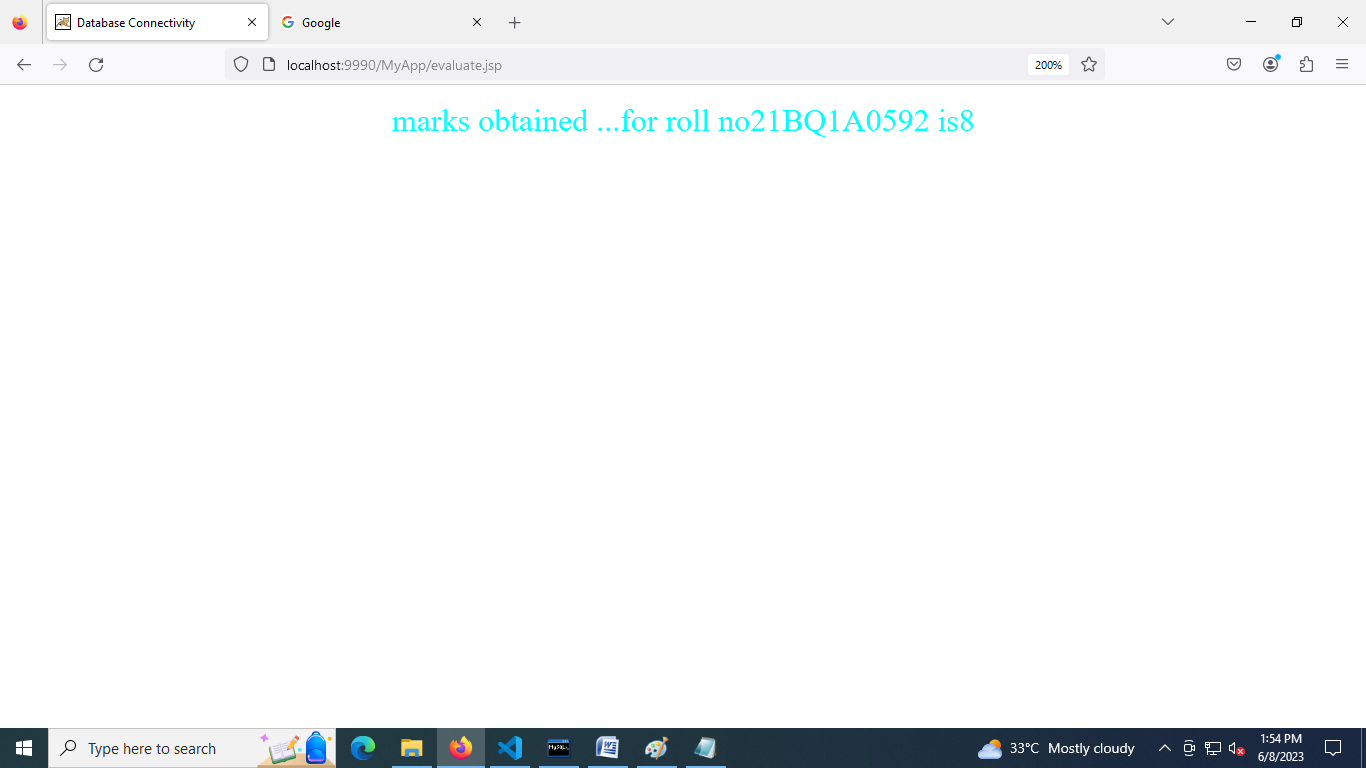
</body>

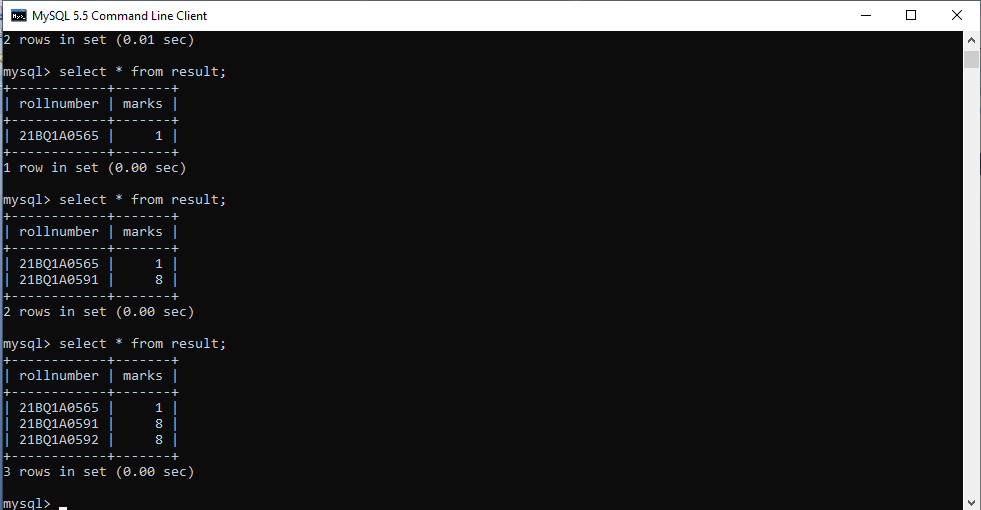
</html>

OUTPUT:









1. Write a program to demonstrate cookie &Sessions using JSP.

Procedure:

* + **Cookies**:
    - Cookies are small pieces of data stored on the client-side (browser) by the server.
    - They contain information such as user preferences, session IDs, or tracking data.
    - Cookies are used to maintain state between multiple requests from the same client.
  + **Sessions**:
    - Sessions are used to maintain state across multiple requests.
    - A session is a logical connection between the client and server.
    - Servlets use the HttpSession interface for session management.
    - Store session-specific data using session attributes (e.g., user authentication, shopping carts).
  + Create JSP pages that demonstrate cookie and session management:
    - Set and retrieve cookies using implicit objects (e.g., request, response).
    - Use HttpSession to store and retrieve session attributes.
    - Verify that data persists across multiple requests.
  + Test your JSP pages with different browsers and cookie settings.
  + Verify that session attributes persist across requests.

Sample Code:

**user.html**

<html>

<head> <title>Cookie Example</title> </head>

<body>

  <form action="cookieex.jsp" method="GET">

                Username: <input type="text" name="username"> <br>

                Email: <input type="text" name="email"/><br>

                <input type="submit" value="Submit"/>

  </form>

</body>

</html>

**cookieex.jsp**

<%

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

  String em=request.getParameter("email");

  Cookie username = new Cookie("username",uname);

  Cookie email = new Cookie("email",em);

   username.setMaxAge(60\*60\*10);

   email.setMaxAge(60\*60\*10);

   response.addCookie(username);

   response.addCookie(email);

%>

<html>

<head><title>Cookie example</title> </head>

  <body>

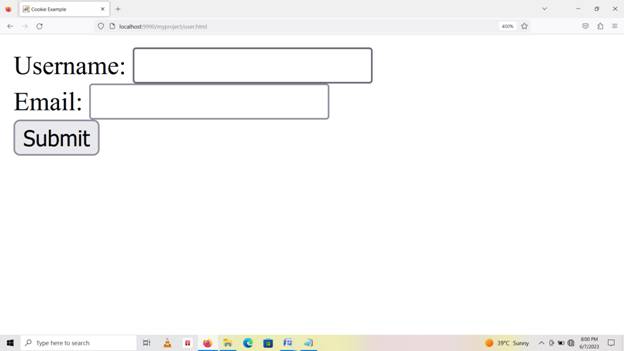
                <b>Username:</b>  <%= request.getParameter("username")%>

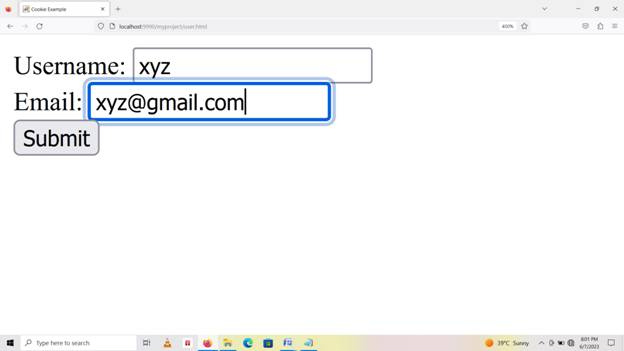
                <b>Email:</b>    <%= request.getParameter("email")%>

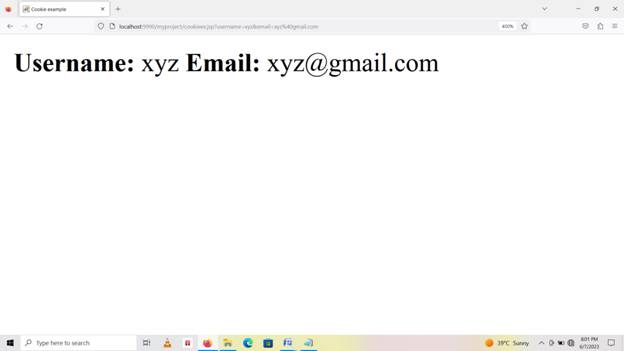
  </body>

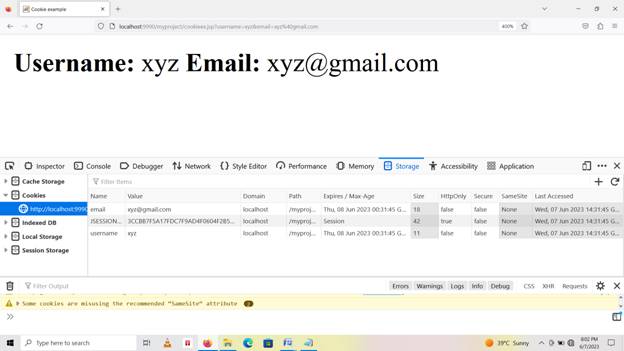
</html>

Output:









III. Session Handling using Jsp

**userdata.html**

<html>

<head> <title>Session Example</title> </head>

<body>

  <form action="sessionex.jsp" method="GET">

                Username: <input type="text" name="username"> <br>

                Email: <input type="text" name="email"/><br>

                <input type="submit" value="Submit"/>

  </form>

</body>

</html>

**sessionex.jsp**

<html>

<head><title>Session Example</title> </head>

  <body>

                <%

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

           String em=request.getParameter("email");

           session.setAttribute("username",uname);

           session.setAttribute("email",em);

        %>

                <jsp:forward page="sessionmsg.jsp"/>

  </body>

</html>

**sessionmsg.jsp**

<html>

  <head>

       <title>Session Message</title>

  </head>

  <body>

                <%

                 String uname=(String)session.getAttribute("username");

                 String em=(String)session.getAttribute("email");

                 out.println("<br> welcome user.."+uname);

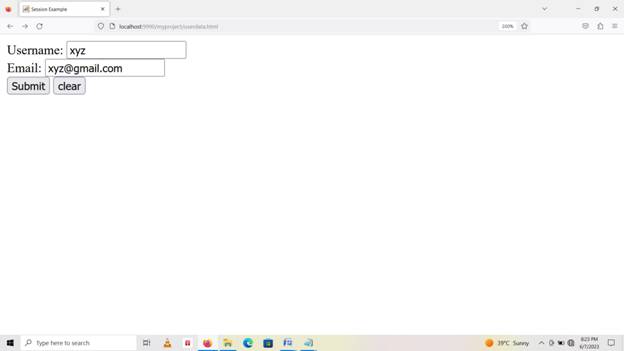
                 out.println("<br> your email id.."+em);

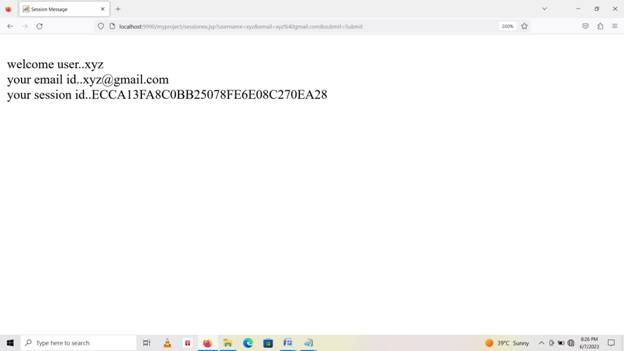
                 out.println("<br> your session id.."+session.getId());

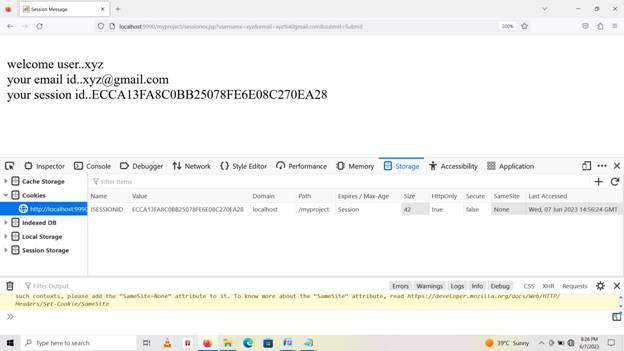
       %>

 </body>

</html>







**CO-PO Mapping:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Course Outcome** | **PO1** | **PO2** | **PO3** | **PO4** | **PO5** | **PO6** | **PO7** | **PO8** | **PO9** | **PO 10** | **PO 11** | **PO 12** | **PSO1** | **PSO2** |
| **CO1** | **-** | **1** | **1** | **2** | **-** | **-** | **-** | **-** | **-** | **-** | **-** | **-** | 2 | 1 |
| **CO2** | **-** | **1** | **2** | **2** | **-** | **-** | **-** | **-** | **-** | **-** | **-** | **-** | 2 | 1 |
| **CO3** | **-** | **2** | **2** | **2** | **-** | **-** | **-** | **-** | **-** | **-** | **-** | **-** | 2 | 1 |
| **CO4** | **-** | **2** | **2** | **2** | **-** | **-** | **-** | **-** | **-** | **-** | **-** | **-** | 2 | 1 |