Unit-4 JSP (Java Server Pages)

→ Jsp technology is used to Create web application just like Servlet technology.

→ JSP is server side scripting lang. Server side scripting means that Jsp Code is processed on web server.

→ .Jsp extention, build dynamic web applications, an improved extended version of servlet technology.

→ A Jsp page Consist of HTML tags & Jsp tags.

→ used in designing & development, provides exp lang, Custom tags, etc.

→ Easy to maintain.

→ Ctrl + A → to add all external Jar files into the path

Procedure

① Open Eclipse. Create a dynamic web project with project name Jsp CSE-D.

② Click on project name > Build path > Configure build path add the libraries > Apply & finish.

③ [HTML, CSS, Jsp, images can be inserted]. click on Src > main > webapp > create HTML file for Html Codes; Create Jsp file for Jsp, html, Java Content.

④ File is Created. write pgm. Run & Execute.

Hello.JSP

```
<% out.print(2*5); %>
```

Jsp API

① javax.servlet.jsp
② javax.servlet.jsp.tagext } - Packages

lang.
obj
Throwable
Serious logical Error
Error
Exception
Simple logical Error
JVM
Programmer

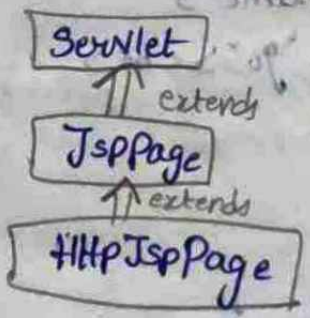
①

↳ 2 interfaces, 6 classes

↳ ① JspPage
② HttpJspPage

↳ * JspError
* JspException
* JspWriter
* PageContext

* JspFactory
* JspEngineInfo



Methods of Jsp Page

① Public void jspInit(): - Same as INIT method of servlet interface
② Public void jspDestroy()

Method of HttpJspPage - Extends javapage interface

① Public void _jspService(): It is used to process req. The underscore signifies that you cannot override this method.

→ In Java class using final class method no one can override the method. → thread, system, string - final class

Jsp Scriplet tag (Scripting Element)

→ In Jsp, java code can be written inside the jsp page using the scriplet tag.

→ The scripting element provides ability to insert java code inside jsp.

→ 3 types → ① Scriplet tag
② expression tag
③ declaration tag

①

→ used to execute java source code in Jsp

Syntax: <% java source code %>

Eg: <% out.print("welcome to jsp"); %>

foom.html

```
<form action="welcome.jsp">
<input type="text" name="uname">
<Input type="submit" value="go"><br>
</form>
```

welcome.jsp

```
<%
String name = req.getParameter("uname");
out.print("welcome"+name);
%>
```

②

Jsp Expression tag

- is written to the output stream of the response.
- not need of out.print()
- mainly used to print values of variable method

Syntax: <% = Statement %>

<% = (10+30) %>

Note: do not end with Semicolon in case of this tag.

Example for Jsp Exp tag

Index.jsp

```
<form action="welcome.jsp">
<input type="text" name="uname"><br>
<input type="submit" value="go"></form>
```

welcome.jsp

```
<%="welcome" +request.getParameter("uname")%>
```

(3) Jsp Declaration tag

Used to declare Variable & methods

Syntax: <%! Variable or method declaration %>

Jsp Scriplet tag can only declare variables not methods	The jsp declaration tag can declare both Variable & methods
declaration is placed inside the ~~jsp~~ _jspService() method.	declaration is placed outside the _jspService() method

index.jsp

```
<%!int data =50; %>
<%="Value of Variable is:" + data %>
<%!
  int cube (int n) {
     return n*n*n;
  }
%>
<%="Cube of 3 is:" + cube(3) %>
```

Q. write a Jsp application to demonstrate the expression tag by using Arithmetic operations.

→

index.jsp

```
<%!
String first = request.getParameter("first");
String second = request.getParameter("second");

int fno=Integer.parseInt(first);
int sno = Integer.ParseInt(second);

%> int k=0;
<% String str=request.getParameter("a1");
   if(str.equals("add"))
      k=fno+sno;
   if(str.equals("sub"))
      k=fno-sno;
   if(str.equals("mul")) k=fno*sno;
   if(str.equals("div")) k=fno/sno;
 %>
  Result is: <%=k%>
```

input.html

```
<form method="post" action="index.jsp">
<h2><center> Mathematical operation </center></h2>
<font size=5 face="Times New Roman">
 <input type="radio" name="a1" value="add" checked>Addition
                                        </input></br>
 <input type="radio" name="a1" value="sub"> Subtraction
                                        </input> <br>
</font>
<table> <tr>
  <td> enter first value: <input type="text" name="first"
                                value=""> </td>
</tr>
<tr> <td> Enter second value: <input type="text" name="second
                                value=""></td></tr>
<br>
<tr>
<td><input type="submit" name="result" value="check result"></td>
</tr>
</table>
</form>
```

(2) Write a Jsp Program to calculate the factorial value for an integer number, while the input is taken from an HTML form.

index.html

```
<form action="factorial.jsp">
enter a number : <input type="int" value="integer">
<input type="submit" value="go"></form>
```

factorial.jsp

```
<%!
    long n, result;
    String str;
    long fact(long n) {
      if(n==0)
          return 1;
      Else
          return n*fact(n-1);
    }
%>
<% str=request.getParameter("val");
   n=Long.parseLong(str);
   result=fact(n);
%>
```

factorial value:
<%=result%>

Fibanocci range b/w 100 & 500

```
<%!
int a=0,
    b=1, c=0;
for(int i=1; i<500; i++)
{
    c=a+b;
    a=b;
    b=c;
    if(c>=100 && c<=500)
      out.println(c);
}
```

100! is 158 digits
can be stored in array

Alternate prime no's

```
int count=0;
boolean ispoime(int n)
{
for(i=2; i<=n/2; i++)
{
   if(n%i==0);

}
}
if(ispoime(6))
  count++;
if(count%2==0)
  out.print(i);
}
```

Jsp Implicit Object

9 Jsp implicit objects ; created by web container that are available to all jsp Pages

	object	Type/className
1	out	JspWriter
2	request	HttpServletRequest
3	response	HttpServletResponse
4	config	ServletConfig
5	application	ServletContext
6	Session	HttpSession
7	pageContext	PageContext
8	page	Object
9	Exception	Throwable

Jsp Out implicit object

Provides implicit object named out. It is object of JspWriter.

② Jsp request implicit object

→ Used to get request information such as parameter, header information remote address, Server name, Server port, Content type, character encoding, etc.

→ also used to set, get & remove attributes from Jsp object.

③ Jsp response implicit Object

→ Used to add or manipulate response such as redirect response to another resource send error etc.

→ Instance of HttpServletResponse is created by web container for each jsp request.

welcome.jsp

```
<%
response.sendRedirect("http://www.google.com");
%>
```

index.html

```
<form>
<input type="submit">
</form>
```

⑥ Session implicit object

Used ~~to~~ this object to set, get or remove attribute or to get session information.

Eg

index.html

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" name="go">
</form>
```

welcome.jsp

```
<%
String name = request.getParameter("uname");
out.print("welcome" + name);
session.setAttribute("user", name);
<a href="second.jsp"> Second jsp page</a>
%>
```

second.jsp

```
<%
String name=(String) session.getAttribute("user");
out.print("Hello" +name);
%>
```

①

→ 4 types of scopes

* Page
* request
* session
* application

Jsp directives

→ they are messages that tell the web container how to translate a Jsp page into the corresponding servlet.

→ 3 types of directives

* Page directive
* include directive
* taglib directive (JSPTL)

Syntax:

```
<%@ directive_name attribute_name="value" %>
```

① Jsp Page directive

→ defines attributes that apply to entire Jsp page

Syntax: `<%@ page attribute="value" %>`

Attributes: import, contentType, extends, info, buffer, language, errorPage, isErrorPage, session, isThreadSafe, pageEncoding.

import

↳ used to import class, interface or all members of package

```
<%@ page import="java.util.Date" %>
Today is: <%= new Date() %>
```

Content Type

<%@ page ContentType = 'application/html' %>

is

② Jsp include directive

→ include directive used to include the content of any resource like html, jsp file; included resource at page translation time

Syntax: <%@ include file = 'resourceName' %>

Eg: <% 'header.html' %>

~~Today~~

Error Page

→ used to define error page, if exception occurs in current page it redirects to another web page.

Exception Handling → Process of handling runtime errors

↳ an object that is thrown at runtime

→ handling exceptions is server side for web developer.

→ 2 ways of exception handling

① By errorPage of isErrorPage attribute of page directive

② By <error-page> element in web.xml file.

index.jsp - for input values

```
<form action = 'process.jsp'>
```

Exception form.jsp

```
<form action = 'ExceptionProcess.jsp'>
No1: <input type = 'text' name = 'n1'><br>
No2: <input type = 'text' name = 'n2'><br>
<input type = 'submit' value = 'divide'>
</form>
```

ExceptionProcess.jsp

```
<%@ page errorPage = "ExceptionError.jsp" %>
<%
   String num1 = request.getParameter("n1");
   String num2 = request.getParameter("n2");
   int a=Integer.parseInt(num1);
   int b = Integer.ParseInt(num2);
   int c=a/b;
   out.print("division of number is "+c);
%>
```

ExceptionError.jsp

```
<%@ page isErrorPage = "true" %>
<h3> sorry an exception occured </h3>
  Exception is : <%= exception %>
```

Jsp Action Tags

→ used to perform some specific tasks

→ used to control the flow between pages & to use JavaBean. (spring, springboot, hybernet)

Jsp Action tags	Description
jsp:forward	forwards the req & res to another resource
jsp:include	includes another resource
jsp:param	sets the parameter value. used in forward & include mostly.
jsp:useBean	creates or locates Bean object.
jsp:setProperty	sets the value of property in bean object

jsp:getProperty	points the value of property of bean

forward Action tag

jsp:forward tag used to forward the req. to another resource like jsp, html, servlet.

Syntax: *without param

```
<jsp:forward page = "relative URL | <%=expression%>" />
```

*with param

```
<jsp:forward page = "relative URL <%=expression%>">
<jsp:param name = "parametername"
        value = "parametervalue" | <%=expression%>" />
</jsp:forward>
```

Eg

① without parameter

forwarding the req. to pointdate.jsp file.

index.jsp (without parameter)

```
<h2>this is index page</h2>
<jsp:forward page = "pointdate.jsp" />
```

Pointdate.jsp

```
<% out.print("Today is: "+java.util.Calendar.getInstance().getTime()); %>
```

② with parameter

index.jsp

```
<h2>this is index page</h2>
<jsp:forward page = "pointdate.jsp">
<jsp:param name = "name" value = "www.google.com" />
</jsp:forward>
```

Index.jsp

```
<jsp:forward page="forwardParamDisplay.jsp" >
<jsp:param name="name" value="chaitanya"/>
<jsp:param name="site" value="BeginnersBook.com"/>
<jsp:param name="tutorialname" value="jsp forward action"/>
<jsp:param name="reqcamefrom" value="forwardParamIndex.jsp"/>
</jsp:forward>
```

forwardParamIndex.jsp

```
<h2>Hello this is a display.jsp page</h2>
My name is: <%=request.getParameter("name")%><br>
The site is: <%=request.getParameter("site")%><br>
the tutorial/Topic is: <%=request.getParameter("tutorialname")%><br>
forward request came from this page: <%=request.getParameter("req
                                    camefrom")%><br>
```

Include Action tag

→ used to include the content of another resource

→ action tag includes the resource at request time so it is better for dynamic web pages because there might be changes in future.

→ jsp:include tag can be used to include static as well as dynamic pages

→ Advantage - Code reusability

→

jsp include directive	Jsp include action
includes resource at translation time	includes at req. time
better for static pages	better for both static & dynamic

JavaBean

→ It is a java class & follows foll. conventions
  * It should have a no-arg Constructor
  * It should provide methods to set & get the value of properties known as getter & setter methods

→ POJO - Plain old Java object. [variables & methods]

→ It is a reusable software component. It encapsulates many objects into one object

JBP > New > package > student
  ↳ > class > studentBean - will be code
rightclick > source > Generate set & getter > select all

```
Package Student;
Public class StudentBean {
int Sno;
String Sname;
String address;
float marks;
Public int getSno() {
  (default code)
}
}
```

⇒

```
Public int getSno() {
    return sno;
}
Public void setSno(int sno) {
    this.sno = sno;
}
Public String getAddress() {
    return address;
}
Public void setAddress(String address) {
    this.address = address;
}
```

Access Bean class

```
Package Student;
Public class StudentBean {
  Public static void main(String args[]) {
    Student Sname = new Student();
    Sname.setName("Arjun");
    System.out.println(Sname.getName());
  }}
```

Properties

→ object need to be created with the help of it the, Properties are accessed. [is a named feature that can be accesses by using object].

→ any datatype, containing the classes that you define

→ readonly, write method.

2 methods in JavaBean's implementation

① getPropertyName() – property name firstName, method name would be firstName()

② setPropertyName() –

syntax

```
<jsp:useBean id="instanceName" scope="page|request|session|applicati
Class="packageName.className" type="packageName.ClassName"
beanName="PackageName.className |<% =expression>" >
</jsp:UseBean>
```

→ bean action tag is used to locate or instantiate a bean class.

→ If bean object is not created, it instantiates the bean.

→ Scope - upto which extent particular bean (data) can be used.

Calculator.java

```
Package bean;
Public class Calculator {
  Public int Cube(int n) { return n*n*n; }
}
```

index.jsp

```
<jsp:useBean id="obj" class="bean.Calculator"/>
```

```
<%
    int m=obj.cube(5);
    out.print("cube of 5 is "+m);
%>
```

jsp:setProperty

Syntax:

```
<jsp:setProperty name="instanceOfBean" property="*" |
  property="propertyName" param="parameterName" |
  property="propertyName" value="{string|<%=expression%>}"
  />
```

→ <jsp:setProperty name="bean" property="*" /> – if you have to set all the values of incoming request in the bean.

→ <jsp:setProperty name="bean" property="username" /> – if you have to set value of incoming specific property

→ <jsp:setProperty name="bean" property="username" value="kumar" /> Set specific value in property.

→ jsp:getProperty action tag – returns the value of of property

Syntax: <jsp:getProperty name="instanceOfBean" property="propertyName" />

Eg: <jsp:getProperty name="beanobj" property="username" />

~~index.html~~

Userform.html

```
<form action="UserProcess.jsp">
Name:<input type="text" name="name">
Password:<input type="password"
          name="password">
Email:<input type="text"
       name="email">
```

Example of bean development in JSP

3 steps

① Userform.html for input values

② UserProcess.jsp file that sets the incoming values to bean object & print one value

③ User.javaBean class have setter & getter method

<input type="submit" value="register">
</form>

UserProcess.jsp

<jsp:useBean id="u" class="org.mrec.User"/>
<jsp:setProperty property="*" name="u"/>
Record:
<jsp:getProperty property="name" name="u"/>
<jsp:getProperty property="password" name="u"/>
<jsp:getProperty property="email" name="u"/>

User.java

```
Package org.mrec;
Public class User {
    Private String name, password, email;
    //setter u getter
}
```

16. b) Create the simple bean & reuse the bean in multiple JSP pages using JSP action tags.

User.java

```
Package UserBean;
Public class User {
    Private String name, password, email;
    Public String getName() {
        return name;
    }
    (default code)
}
```

src/main/java
UserBean
> User.java

BeanIndex.html

```
<form action="BeanProcess1.jsp">
Name:<input type="text" name="name"><br>
Password:<input type="password" name="password"><br>
Email:<input type="text" name="email"><br>
<input type="submit" value="register">
</form>
```

BeanProcess1.jsp

```
<jsp:useBean id="u" class="UserBean.User" scope="session"/>
<jsp:setProperty property="*" name="u"/>
<a href="BeanProcess2.jsp">visit page</a>
```

BeanProcess2.jsp

```
<jsp:useBean id="u" class="useBean.User" scope="session"/>
```

BeanProcess3.jsp

```
<%
 String name="arjun";
%>
<jsp:setProperty property="name" name="b" value="<%=name%>"/>
Record:<br>
<jsp:getProperty property="name" name="u"/><br>
```

15. b) Demonstrate how to use the Cookies in session tracking of Jsp application.

CookieIndex1.html

```
<form action="CookieDemo1.jsp
UserName:<input type="text" name="UserName"><br>
Password:<input type="text" name="UserPassword"><br>
```

```
<input type="submit" value="submit"/>
</form>
```

CookieDemo1.jsp

```
<%
  String username = request.getParameter("userName");
  String password = request.getParameter("Password");
  out.println("Hello " + username);
  out.println("Your password is " + password);

  Cookie c1 = new Cookie("username", username);
  Cookie c2 = new Cookie("password", password);

  response.addCookie(c1);
  response.addCookie(c2);

  out.println("<form action = CookieDemo2.jsp">);
  out.println("<input type='submit' value='go'>");
  out.println("</form>");
%>
```

CookieDemo2.jsp

Ctrl + Shift + / (backslash) → for Comment

```
<%
  Cookie ck[] = request.getCookies();
  out.print("username: " + ck[0].getValue());
  out.print("password: " + ck[1].getValue());
%>
```

BankForm.html

```
<form action = "BankProcess.jsp">
Enter Customer Name: <input type="text" name="cname"><br>
Enter Account number: <input type="text" name="accno"><br>
Enter Email: <input type="text" name="email"><br>
Enter Phone number: <input type="text" name="phone"><br>
Enter Address: <input type="text" name="address"><br>
```

```
<input type="Submit" value="Submit">
</form>
```

BankProcess.jsp

```
<%! int balance=1000;
Public int getBalance()
{
 return balance;
}
Public void deposit(int amount)
{
  balance = balance + amount;
}
Public void withdraw(int amount)
{
   if(balance>amount)
    balance(=balance -amount;
  }
 %>
 <b>
 <%="Customer Name: "+request.getParameter("cname"); %>
 <%="<b> Account number: "+request.getParameter("accno"); %>
 <%="<b> Customer EmailId: "+request.getParameter("email"); %>
 <%="<b> Customer Phonenumber: "+request.getParameter("phone"); %>
 <%="<b> Customer Address: "+request.getParameter("address"); %>

 <b><b>
 <%
  out.println("<b> Initial Balance: "+getBalance());
  deposit(500);
  out.println("<b> Balance after depositing: "+balance);
  withdraw(300);
  out.println("<b> Balance is after withdrawal: "+balance);
  %>
```

15a) Voting system

Voteform.html

```
<form action = "VoteProcess.jsp">
 Enter StudentName : <input type="text" name="sname"><br>
      StudentAge : <input type="text" name="sage"><br>
      Student Mobile : <input type="text" name="smobile"><br>
      Student Address : <input type="text" name="saddress"><br>
 <input type="submit" value="Register">
</form>
```

VoteProcess.jsp

```
<h1> welcome to College Voting system</h1>
<br>
<% int age = Integer.parseInt(request.getParameter("sage"));
 if(age>=18) {
  %>
  <jsp:forward page="Voterage.jsp">
  <jsp:param name="VoterId" value="REC-234"/>
  <jsp:param name="Status" value="Thankyou for voting"/>
  <jsp:param name="msg" value="Have a nice day"/>
  </jsp:forward>
  <% }
  Else {
  %>
  <jsp:include page="VoteMinor.jsp">
  <jsp:param name="Status" value="You are Minor"/>
  <jsp:param name="msg" value="Try after few years"/>
  <jsp:include>
  <% } %>
```

VoteMajor.jsp

```
<%
  String sname = request.getParameter("sname");
  int sage = Integer.parseInt(request.getParameter("sage"));
  String status = request.getParameter("status");
  String VoterId = request.getParameter("VoterId");
  String msg = request.getParameter("msg");
out.print("Hello "+sname);
out.print("<br>Your age is : "+sage);
out.Print("<br> Your VoterId: "+VoterId);
out.Print("<br>"+status);
out.print("<br>"+msg+"Bye");
%>
```

VoteMinor.jsp

```
<%
  String sname = request.getParameter("sname");
  int sage = Integer.parseInt(request.getParameter("sage"));
  String status = request.getParameter("status");
  String msg = request.getParameter("msg");
  out.print("Hello "+sname);
  out.print("<br>Your age is "+sage);
  out.print("<br> Boss "+status);
  out.print("<br>"+msg+" Bye<br><br>");
%>
<a href="Voteform.html">New Voting Registration form</a>
```

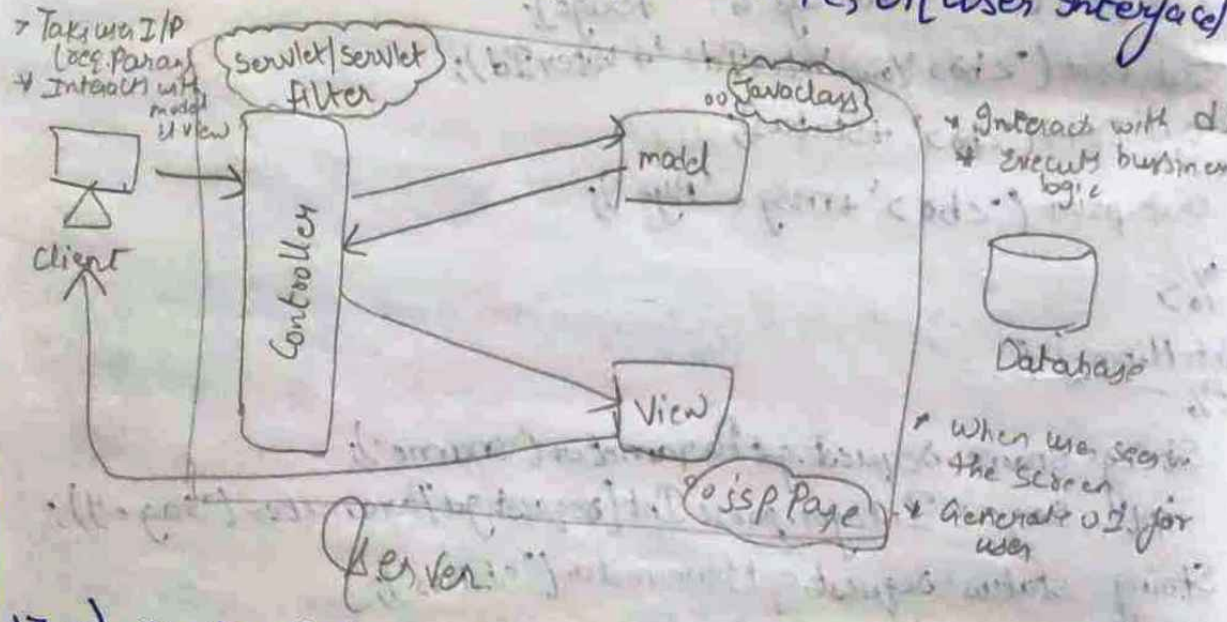
Model View & Controller Architecture (MVC)

→ It is a design pattern that separates the business logic, presentation logic, data.

→ Controller - acts as an interface b/w view & model. Controller intercepts all the incoming requests.

→ Model - represents the state of app. i.e, data. It can also have business logic

→ View - represents the presentation i.e, UI (User Interface).



17. a) Create Student action tags bean class & manage the Student login details using JSP MVC where Java bean class as a model, JSP as a View Component & Servlet as a Controller.

Index.html [A Page that gets input from student or user]

```
<form action="ControllerServlet" method="get">
name<input type="text" name="uname"><br>
Enter password <input type="text" name="password"><br>
<input type="submit" value="Login"><br>
</form>
```

ControllerServlet.java

```
import java.io.PrintWriter;
import java.io.*;
import javax.servlet.http.*;
import javax.servlet.*;
Public class ControllerServlet extends HttpServlet {
    Protected Void doGet (HttpServletRequest req, HttpServletResponse
    res) throws ServletException, IOException {
    res.setContentType("text/html");
    PrintWriter out = response.getWriter();

    String username = req.getParameter("uname");
    String Password = req.getParameter("Password");

    LoginBean bean = new LoginBean();
    bean.setUsername(uname);
    bean.setPassword(Password);
    req.setAttribute("bean", bean);

    boolean Status = bean.validate();
    if(status) {
        RequestDispatcher rd = req.getRequestDispatcher("login-success.jsp")
        .rd.forward(req, res);
    }
    Else {
        RequestDispatcher rd = req.getRequestDispatcher("login-error.jsp");
        rd.forward(req, res);
    }
}
}
```

LoginBean.java

```
Public class LoginBean {
  Private String username, Password;
  Public String getUsername() {
        == setter & getter methods
  }
  Public boolean validate() {
   if (Password.equals("admin")) {
       return true;
   }
   else {
       return false;
   }
  }
}
```

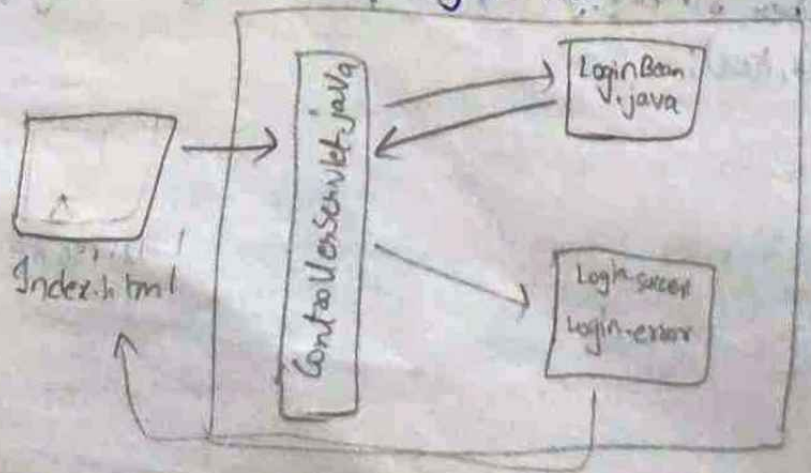
src/main/java
↳ model (Package)
↳ LoginBean.java
↳ Controller
↳ ControllerServlet.java

Login-success.jsp

```
<%@ Page import="bean.LoginBean" %>
<p> You are Successfully logged in! </p>
<%
  LoginBean bean = (LoginBean) request.getAttribute("bean");
  out.print(" welcome, " + bean.getName());
%>
```

Login-error.jsp

```
<p> Sorry! Username or password error </p>
<%@ include file="index.jsp" %>
```



Browser Object Model

JavaScript

alert msg

```
<script>
function msg(){
  alert("Hello Alert Box from MREC");
}
</script>
<input type="button" value="click the box" onclick="msg()">
```

Confirm msg

```
<script>
function msg()
{
  var v= confirm("Are u sure?");
  if(v==true){
    alert("ok");
  }else{
    alert("Cancel");
  }
</script>
<input type="button" value="delete record" onclick="msg()">
```

Prompt

```
<script>
function msg()
{
  var v= prompt("Who are You?");
  alert("I am "+v);
}
</script>
<input type="button" value="Click for Name" onclick="msg()">
```

Open function

```
<script>
function msg(){
  open("http://www.google.com");
}
</script>
```

<input type="button" value="Google Search" onclick="msg()">

History Object

→ represents an array of URL's visited by user. By using this object, you can load previous, forward or any Particular page

③ → forward() : history.forward();
back() : history.back();
go() : history.go(2); // forward next 2nd page
history.go(-2); // forward previous 2nd page

Document Object Model (DOM)

→ we can add dynamic content to our webpage
→ It is object of window
It can be accessed by window document or document.

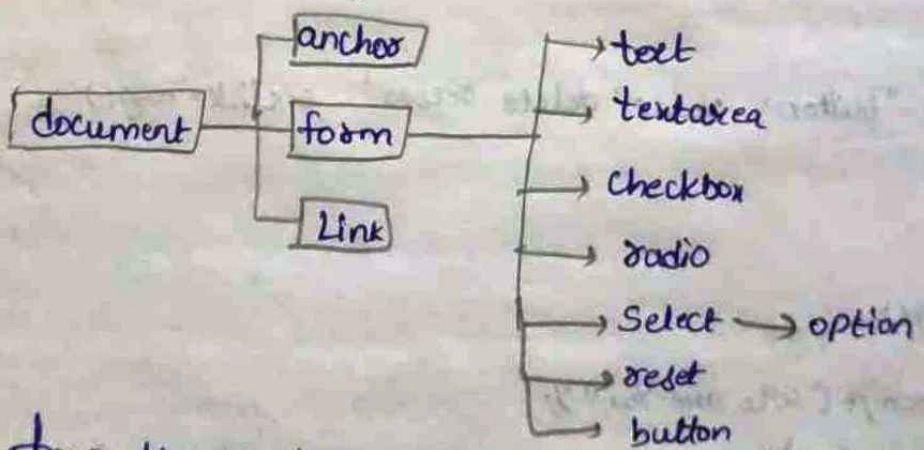


Fig. Properties of document object

Getcube

```
<Script>
function getCube()
{
 var number = document.getElementById('cube').value;
 alert(number * number * number);
}
```

```
</script>
<form>
Enter No:<input type="text" id="cube" name="cube"><br>
<input type="button" onclick="getCube()" value="cubeofn">
</form>
```

form1

```
<script>
function pointValue() {
  var name=document.form1.username.value;
  alert("Welcome: " +name);
}
</script>
<form name="form1">
Entername: <input type="text" name="username">
<input type="button" onclick="pointValue()" value="Pointthevalue">
</form>
```

getElementsByName

```
<script>
function totalbranches()
{
  var branches=document.getElementsByName("branch");
  alert("Total Branches:" + branches.length);
}
</script>
<form>
CSE: <input type="radio" name="branch" value="cse">
ECE: <input type="radio" name="branch" value="ece">
EEE: <input type="radio" name="branch" value="eee">
<br>
<input type="button" onclick="totalbranches()" value="Total Branches">
</form>
```

getElementsByTagName

```
<script>
function countpara()
{
  var totalpara = document.getElementsByTagName('p');
  alert ("total p tags are: " + totalpara.length);
}
</script>
<p> This is paragraph</p>
<h1> This is h1 tag </h1>
<p> we are going to count total no of paragraph </p>
<p> Let's see example</p>
<button onclick="countpara()"> Count Paragraph </button>
```

innerHTML property

→ used to write dynamic html on html document
→ used mostly on webpages to generate the dynamic html. such as registration form, links, etc.

```
<script>
function showcommentform()
{
  var data=" Name: <input type='text' name='name'><br>
            Comment: <br>textarea rows='5' cols='80'></textarea>
            <br><input type='submit' value='post comment'>";
  document.getElementById('mylocation').innerHTML= data;
}
</script>
<form name="myform">
<input type="button" value="comment" onclick="showcommentform()">
<div id="mylocation"></div>
</form>
```