

→ INTRODUCTION TO HTML:-

HTML (Hyper text Markup language) is a scripting language for web pages. and output of programs can be seen after using browser. The HTML documents are plain text files which the user can create using a text editor like Notepad/ Edit.

SGML (Standard Generalized Markup language) was used as the basis of HTML. SGML is not a Markup language, but a Markup metalanguage, which means it is a language for defining markup languages; it allows you to define your own tags, each of which is used to make a particular sort of document element, such as

Paragraph, section heading/list.

HTML is only a markup lang. which provides a set of tags suitable for making up webpages but does not provide any way of defining new tags.

→ HyperText Language:-

HTML is hyper text lang. becoz it supports font styled text pictures, graphics and animations and also provides hyper links that used to brows internet easily. Text becomes hypertext with the addn of links that connects other hypertext documents. Hypertext is text and links - pointers to other pieces of text, possibly else where

in same doc. (internal linking) or
in other doc. (external linking).

→ Markup language:-

HTML is a Markup lang. becoz any text simply placed in file is the data content, and text is shown as it is on browser.

→ Tags In HTML:-

HTML is a tag-based system. A tag is a special instrn for browser which indicates that given info. after tag should be rendered according to given instrn.

Tags are lexically distinguished from text proper, by being enclosed in angle brackets or beginning with backslash character.

A tag is made up of a left operator (<) and a right operator (>) and a tag name b/w these two operators. optionally tag name contain one or more parameter. If you forget to mention right operator (>) or if you give any space b/w left operator and tag name browser not considers it as tag. At the same time if browser not understands the tag name it just ignores it, browser won't generate any error.

Syntax:-

<tag name [parameter(s) [= value]]>

left operator

Right operator.

A number of points should be noted about HTML tags: 1-④

- a) tags are delimited by angled brackets
- b) they are not case sensitive
- c) styles must be switched off by an end tag.
- d) white spaces, tabs and newlines are ignored by browser.
- e) if a browser doesn't understand a tag it will usually ignore it.

→ Structure of HTML document:-

All HTML docs follow same basic structure. They have a head which contains ctrl info. used by browser and server and a large body. The body contains content that display on screen & tags which ctrl how that content is formatted by browser.

The basic document is:

```
<html>
  <head>
    <title> A HTML document </title>
  </head>
  <body> ... </body>
</html>
```

→ Some Basic HTML Tags:-

① ~~<html>~~ ... </html>

In order to identify by browser for a document whether it is html code or not it verifies <html> tag in that file and then considers that file as html and displays accordingly.

Ex: <html>

 This is simple HTML
 </html>

② `<head>... </head>`

head is first part of HTML document which contains webpage title.

③ `<title>... </title>`

Specifies title of document.

Ex: `<html>`

`<head>`

`<title> My first document`

`</title>`

`</head>`

`</html>`

④ `<body>... </body>`

Second part of HTML doc. and it is largest part. All the remaining info abt webpage content is included in `<body>` tag. Body tag has different parameters which include back-

- ground color, link color, text color etc.

- Background parameter specifies background image for webpage.

- BGcolor parameter specifies the background color for webpage.

- Text parameter specifies text color.

⑤ Blocks:- (`<P>...</P>`, header tag)
The two major blocks of text in HTML documents are paragraph and heading.

(i) `<P [align = "left" / "center" / "right"] >`
...`</P>`

Specifies a paragraph.

(ii) headers / predefined tags:

headers are simple forms of text formatting that vary text sizes based on header's level.

The size varies from one browser to another browser.

Ex: `<html>`
 `<head>`
 `<title>` Blocks in HTML `</title>`
 `</head>`
 `<body bgcolor= blue>`
 `<h1>` Web Technologies `</h1>`
 `<h2>` " " `</h2>`
 `<h3>` " " `</h3>`
 `<h4>` " " `</h4>`
 `<h5>` " " `</h5>`
 `<h6>` " " `</h6>`
 `</body>`
 `</html>`.

⑥ Text :-

The text on a HTML page can be altered in a number of ways: the actual font used can be changed to

to attempt to force browser to use a specific font and look of text can be changed for emphasis.

(i) `<basefont size="n">`

Lets you specify a minimum font size for basic text but not for headings. The size argument takes an integer from 1 to 7.

(ii) `<font size="[+/-n]" color="#rrggbb">`

sets font size relative to either default value or to any size set by `<basefont>`. Absolute font sizes can be forced by using an integer from 1 to 7; relative font sizes are set by using +/- 1 to 7.

(iii) `<b>.. </b>`

bolds the text

(iv) `<u>` .. `</u>`

Underlines text.

(v) `<i>` .. `</i>`

Italic text.

(vi) `<em>` .. `</em>`

for emphasis (new standard for italic)

(vii) `<strong>` .. `</strong>`

for strong. (new standard for bold)

(viii) `<s>` .. `</s>`

Strikes the text.

Ex: `<html>`

`<head>`

`<title>` Text `</title>`

`</head>`

`<body>`

Contd..

<h1>Changing Font Sizes </h1>

<basefont size="3">

<p>Here is some text in size three

<p>And here is some

 larger

 t

 e

 x

 t

</basefont>

<U><I> Text Styles </I></U>

 Web technologies

<U>

<I>

<S>

</body>

</html>.

(ix) Subscripts & Superscripts

<sub>

<sup>

CO₂

O²

Ex:

<html>

<body>

 Formula for water

H <sub>2

</sub> O

 Basic Mathematical formula

(a+b) <sup>2

</sup> = a <sup>2

</sup> + b <sup>2

</sup> +
2 * a * b.
</body>

</html>

(x) Character ~~exp~~ escape sequences:

& < > " ©
& < > ©

(X) Horizontal Rules <hr>

<hr {width=n} [size=n]

[align = {left/right/center}] [Noshade]>

⑦

Linking in HTML:-

→ The most important capability of HTML is its ability to create hyperlinks. Hyperlinks can be applied either to text/images.

Anchor tag is used to create links. Minimum it requires a parameter 'href' which indicates destination document.

Syntax:

Ex: One.html

```
<html>
```

```
<body>
```

```
  welcome!
```

```
</body>
```

```
</html>
```

two.html

```
<html>
```

```
<body>
```

```
<a href = "One.html"> Click here </a>
```

```
</body>
```

```
</html>
```

(8)

LISTS:-

List is a collection of items and they may be ordered or unordered.

HTML provides 3 types of list:

(i) ordered or numbered list

① - -

(i) unordered or bulleted list

 ..

(ii) definition list.

<dl> .. </dl>

Ex: <html>

<head>

<title> List </title>

</head>

<body>

<h2> Unordered list </h2>

 ABC

 DEF

<ul type="square">

 GHI

<h2> Ordered list </h2>

 abc

<ol type='i'>

 def

<ol type='A' start=5>

 ghi

<dl>

<dt> widget </dt>

<dd> provided in 3 sizes </dd>

</dt>

</body>

</html>.

⑨

Tables :-

Two uses:

- a) structuring pieces of info.
- b) Structuring the whole web page.

`<table> .. </table>`

indicates start of a table.

`<thead> .. </thead>`

starts header part of table, and it should be followed by `<th>`

`<th> .. </th>`

Used to give table heading (column header).

`<tbody> .. </tbody>`

starts main area of table, says actual data is started.

`<tr> .. </tr>`

Indicates starting of a table row.

`<td> .. </td>`

Actual table data for a cell, for each cell we should give `<td>`

`<tfoot> - </tfoot>`

Used to define foot info. of table.

`<caption> - </caption>`

to insert caption into table, inserts caption data directly above table,

`<table [align="center"/"left"/"right"]
[border [=n]] [cellpadding="n"]
[width="n%"] [cellspacing="n"]>
- </table>`

cellpadding— in pixels determines how much space there is b/w contents of a cell and its border.

cellspacing— sets amount of white space b/w cells.

width— sets amount of screen that table will use.

```

Ex: <html>
    <head>
        <title>table</title>
    </head>
    <body>
        <table border=1>
            <caption>Comparison Statement</caption>
            <thead>
                <tr>
                    <th rowspan="2" bgcolor=tan>features
                    <th colspan=2>Companies
                </th>
            </tr>
            <tr>
                <th>compay <th> laser
            </tr>
        </thead>
        <tbody>
            <tr>
                <td> price <td> 11,340 <td> 12,000
            </tr>
            <tr>
                <td> warranty <td> 2yrs <td> 1yr
            </tr>
        </tbody>
    </table>
</body>
</html>

```


10) Images

Images play an important role in web pages. The total look of a webpage is improved using images.

Images can be embedded inside document using `<img>` tag.

Syntax:

```

```

Image as hyper link

```
<a href=a.html>  </a>
```

11) Forms :-

`<form>` tag is used to create html form. This tag has several attributes and important among them is Method and action.

Action parameter indicates path to this script; when user submits form where it should go is given in this parameter

Method parameter specifies how data entered in form is sent to destination.

There are two methods of sending info; One is post other is get.

Post	get.
1) Info. sent along body. 2) Data is not visible while sending. 3) provides security 4) Can send any no. of characters 5) spl characters are possible	- Info sent along URL - Data visible in site address - No security. - limited characters - Not possible.

User Interface elements in forms:-

(i) Label:-

You may observe each and every user interface has some heading which is called label.

(ii) Text Fields:-

This can be created using `<input>` tag.

```
<input type="text" [size=n] [name=" "]
      [maxlength=n] [value=" "]>
```

(iii) Buttons:-

3 types of Buttons. Button, Submit button, reset button.

- Button creates a simple button.

```
<input type="button" [value=""]>
```

- Submit button is clicked, then form action should be invoked.

```
<input type="submit" [value=""]>
```


- Reset button is clicked then it clears all entries user has entered in form.

`<input type="reset" [Value=" "]>`

(iv) Password field :-

`<input type="password" [size=10]
[Name=" "]>`

(v) Radio Button :-

Only one selection at a time.

`<input type="radio" [Name=" "]
[Value=" "]>`

(vi) checkboxes :-

Multiple selections are possible

`<input type="checkbox" [Name=" "]>`

(vii) Combo Boxes & list boxes :-

Combo box: `<select [Name=" "]>
<option> ...
<option> ...
</select>`

list box :
 <select [size=2] [name=]
 <option> . selected > ..
 <option> ..
 </select>

(vii) text area:-

Sized by rows and columns

<textarea [name=" "] [rows=]
 [cols=] > .. </textarea>

Ex!

<html>

<head>

<title> Admission form </title>

</head>

<Body bgcolor=tan>

<form name=form1>

<center> Admission form </center>

<table border=1 align=center>

<tr>

<td> S No

<td> <input type=text>

</tr>

<tr>

<td> Name

<td><input type="text">

</td>

<tr>

<td> Password

<td><input type="password">

</td>

<tr>

<td> Sex

<td><input type="radio"> Male,

<input type="radio"> Female

</td>

<tr>

<td><input type="checkbox">

<td> Require hostel facility

<input

</td>

<tr>

<td> Address

<td><textarea rows="3" cols="20">

</textarea>

</td>

<tr>

<td> Course

<td> <select>

<option [selected]> BTech </option>

<option> MTech </option>

<option> others </option>

</select>

</tr>

<tr>

<td>

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

<td> <input type="reset" value="reset">

</td>

</table>

</form>

</body>

</html>.

⑫ Frames:-

A set of frames are defined using `<frameset>` tag which ends with `</frameset>` tag.

Attributes are:

a) row / column

b) size

The size of frames are mentioned in any one of following units

a) pixel units

b) Percentage units

c) Fraction unit.

a) pixel units :-

represent no. of pixels in each frame. Commas must separate numbers.

Ex: `<frameset cols="150, 70, 70">`
`</frameset>`

b) percentage Unit:

Divides window according to specified percentages.

ex: `<frameset rows="70%,30%">`
`</frameset>`

The defn of frame is given using `<frame>` tag. The attributes are:

- i) source html address (src)
- ii) name of frame (name)
- iii) Margin width (Margin width)
- iv) scrolling button (scrolling)
- v) Can it be resized (noresize)

ex: `<html>`
`<head>`
`<title> Frames </title>`
`</head>`

<body>

<frameset rows = "45, 55">

<frame name = f1 src = one.html>

<frame name = f2 src = two.html>

</frameset>

</body>

</html>

Nested Frameset :-

Frameset within another

frameset

Ex: <frameset rows = "20%, *">

<frame name = "f1" src = One.html>

<frameset cols = "25%, *">

<frame name = f2 src = two.html>

<frame name = f3 src = three.html>

</frameset>

</frameset>

→ CASCADING STYLE SHEETS :-

A style is simply a set of formatting instructions that can be applied to a piece of text. There are three mechanisms by which we can apply styles to our HTML documents:

- a) Style can be defined in basic html tag.
- b) styles can be defined in `<head>` section and applied to whole document.
- c) styles can be defined in external files called stylesheets which can then be used in any document by including stylesheets via a URL.

Ex: <html>

<head>

<title> Simple stylesheet </title>

<style>

h1{

color: red;

border: thin groove;

}

h2{

color: green;

margin-left: 60%;

}

• myid{

text-align: right;

color: purple;

}

• myid2{

align: right;

}

</style>

</head>

→ Defining Own Style:-

① Rule:-

A style rule has two parts:-
a selector and a set of declaration.

- selector used to create link b/w
rule and html tag.

- declaration has two parts :
property & value.

• Selector { property : value ; Property : Value - }

Ex: body {
background-color : #eebd2;
}
h1 {
color : #eeebd2;
background-color : #d8a29b;
font-family : "Book Antiqua",
Times, serif;
border : thin groove #9baab2;
}

Selector → body {
h1 {
Property → color : #eeebd2;
background-color : #d8a29b;
font-family : "Book Antiqua",
Times, serif;
border : thin groove #9baab2;

<body>

<h1> Simple style sheet </h1>

<p> The first paragraph is left
unaltered </p>

<p style="margin-left: 10%;
border: ridge;
background: #ffffcc">

But this paragraph undergoes some
fairly radical alterations </p>

<h2> Here's is a heading </h2>

<p class="myid"> Followed by some
text </p>

<h3 class="myid2"> A label </p>

</body>

</html>.

②

Classes:-

If you only want to apply a style to some paragraphs, for instance you have to use classes.

selector.classname {property: value; ... }

<selector class=classname>

Ex: h1.fred {

color: #eebdbl;

background-color: #d8a29b;

font-family: "Book antiqua"
Times, serif;

border: 1px groove #9baab2;

}

<h1 class="fred"> Simple heading.</h1>

Anonymous classes:-

<html>

<head>

<title> Anonymous classes </title>

<style>


```
•fred {  
  color: #eeebe2;  
  background-color: #d8a29b;  
  font-family: 'Book Antiqua', Times,  
               serif;  
  border: thin groove #9baab2;  
}
```

</style>

</head>

<body>

<h1 class="fred"> A simple heading

</h1>

<p class="fred"> Applying style fred
to a paragraph of text

</p>

</body>

</html>.

③ Including stylesheets:

<link rel="stylesheet" href="url"
type="text/css" ~~media="screen"~~>

Ex:

<link rel="stylesheet" href="style.css"
type="text/css">

↳ Properties & Values in styles:-

① Fonts:-

font-family: <family-name> [<generic-family>]

font-style: normal / italic / ~~bold~~ / oblique.

↓
slanted.

font-weight: normal / bold / bolder / lighter

font-size: [small / medium / large] / [smaller / larger] / <length> / <percent>

② Background & Colors:-

Color: <value>

background-color: <value>/transparent

background-image: URL/none.

- Color of any attribute can be changed.
- Values should be given as hexadecimal values.
- If you set background-color you should set background-image to none.

③ Text:-

text-decoration: none/underline/overline
/ line-through

text-transformation: none/capitalize/uppercase/
lowercase.

text-align: left/right/center/justify

text-indentation: length/percentage.

④ Boxes:-

margin: length/percentage/auto {1, 4}

border-width: thin/thick/medium/length {1, 4}

padding: length/percentage {1, 4}

border-color: value {1, 4}

border-style: none/dotted/solid/double/groove/
ridge {1, 4}

width: length/percentage/auto

height: length/auto

→ For Matting blocks of information:-

a) classes:-

A class is a defn. of a set of styles which can be applied as you choose.

```
<html><head>  
  <title> Changing the body </title>  
  <style> body {  
    color: white;  
    background-color: red;  
  }  
</style>  
</head>
```

```
<body>
  <h1> A New Page </h1>
</body>
</html>
```

(or)

```
<body bgcolor="red" text="white">
```

② Divisions:-

Any formatting that needs adding is placed inside the `<div>` tag.

thus,

```
<div class="anyelement">
  <p>...</p>
  <h2>...</h2>
</div>
```

③ Spans:-

The HTML standard no longer supports idea of modifying individual items in place. This is to remove problems that can arise with indiscriminate use of colors and font tags.

spans are used as follows:

`<p><span class="anyelement">The</span> span tag`

②

Layers:-

z-index: n

This specifies the layer
if $n=1$, then layer appears on top of
background.

Position: absolute/relative.

left: n

top: n

width: n

height: n

Ex: `<html>`

`<head>`

`<title> Layering Text </title>`

`</head>`

`<body>`

`<h1> Layering Text </h1>`

`<div style="z-index: 2; left: 50px;`

`top: 200px; position: absolute;`

`color: red; text: white; font-size: 36pt;`

`border: thin-groove; ">`

`<p> This is higher layer </p> </div>`

<tbody>

<div style="z-index: 4; left: 10%; top: 30px;
width: 100px; position: absolute;
background-color: yellow;
color: black; font-size: 18pt;">

<p> Some text placed in a box
that doesn't go right across
the screen </p>

</div>

</body>

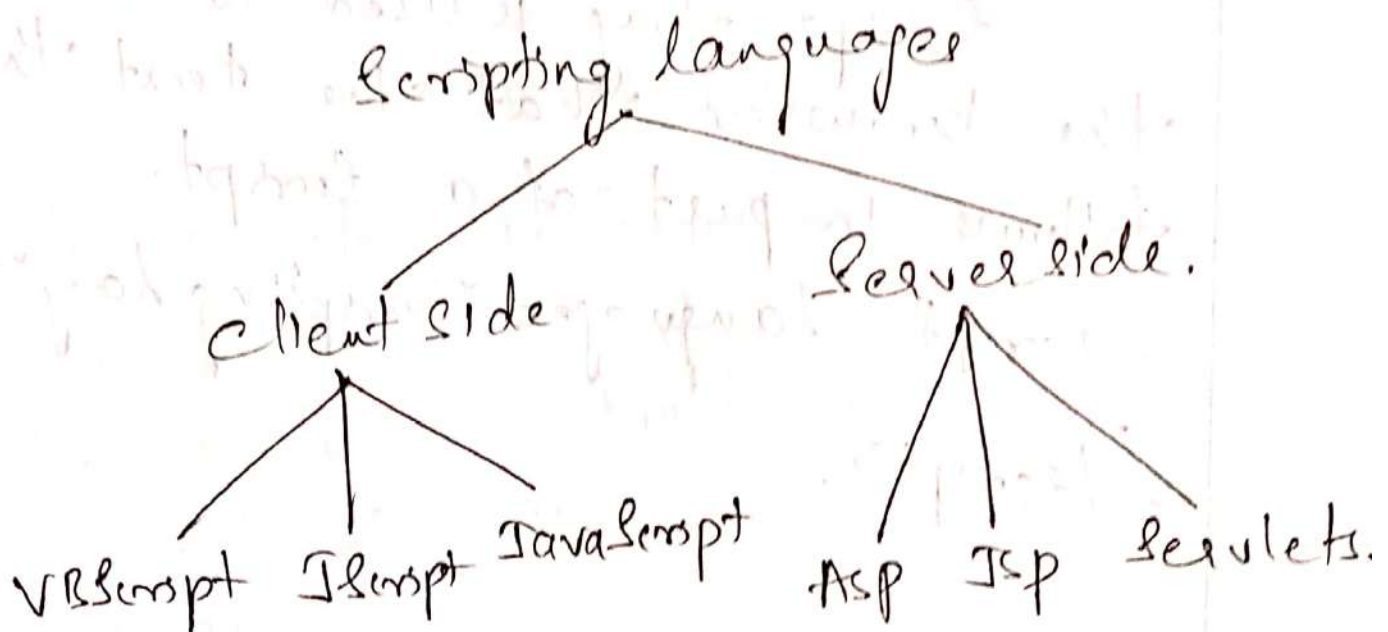
</html>

SCRIPTING LANGUAGES

There are languages some time referred to as a "little languages". They are very easier to understand.

Scripting languages are of two types

- Client Side Scripting used for verifying simple validations at client side.
- Service Side Scripting used for database verification.



Importance of Scripting languages:-

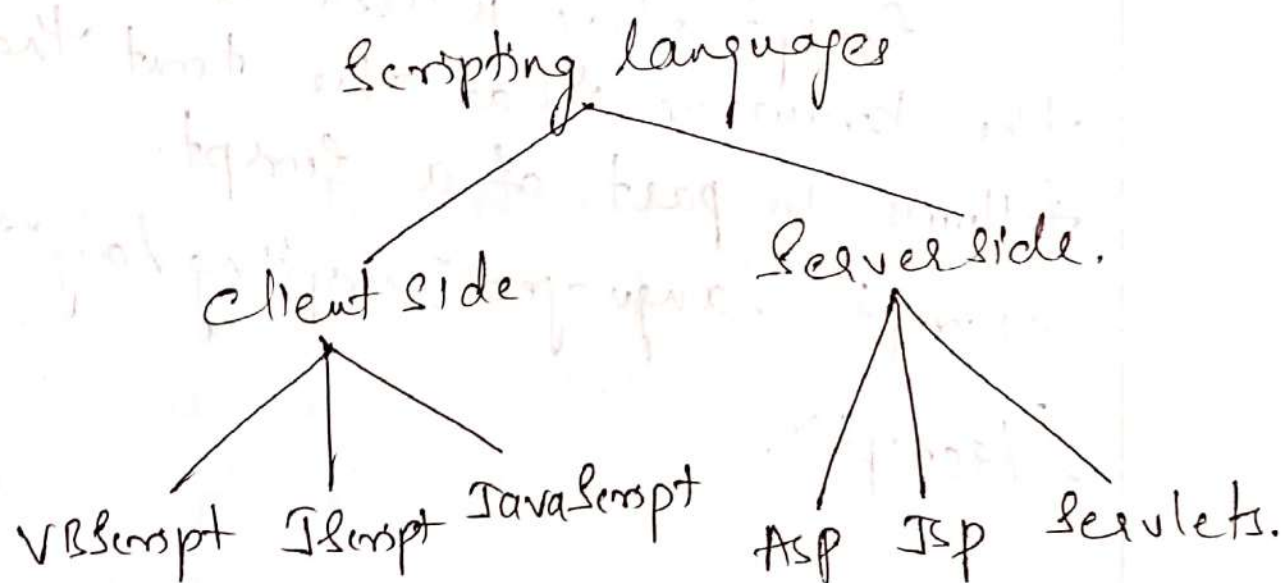
- ① They facilitate a disciplined approach to design computer programs that enhance functionality & appearance of webpages.

→ SCRIPTING LANGUAGES

These are languages sometimes referred to as a "little languages". They are very easier to understand.

Scripting languages are of two types:

- Client side scripting used for verifying simple validations at client side.
- Server side scripting used for database verification.



→ Importance of Scripting languages:-

- ① They facilitate a disciplined approach to design computer programs that enhance functionality & appearance of webpages.

- (2) They are specialised programming lang. that are inserted on webpage to control different elements of page including elements, frames and browser interface.
- (3) They modify document's content dynamically. etc.

<script> Tag :-

Script tag is used to indicate the browser that the text that follows is part of a script.

`<script language="Scripting lang">`

`</script>.`

→ JAVA SCRIPT - INTRODUCTION

Javascript is an object-based scripting language designed to enhance webpage that are constructed with html documents.

webpages are of two kinds:

- a) Static web pages: here no specific interaction with client.
- b) dynamic web pages: Interactions with client as well as validation can be added.

↳ Similarities b/w Javascript & Java:-

- 1) Both have same kind of operators
- 2) Similar control structures
- 3) Concepts of Objects & Methods are similar.

↳ Differences b/w Javascript & Java:-

- 1) Java is object-oriented, where Javascript is Object based programming lang.
- 2) Java source code is first compiled and

and client interprets the code.

Javascript code is not compiled only interpreted.

(3) Inheritance, polymorphism, Threads are not available in Javascript.

↳ Benefits of Javascript:-

(1) Widely supported in web Browsers

(2) It gives easy access to document objects and can manipulate most of them.

(3) Javascript can give interesting animations without long download time associated with many multimedia datatypes;

(4) Web Surfer's don't need a special Plug-in to use your scripts.

(5) Javascript is relatively secure -

Javascript can neither read from your local hard drive nor write to it and you can't get a virus infection directly from Javascript.

↳ Problems with JavaScript:-

- (1) access to objects differ from browser to browser.
- (2) If your script doesn't work then your Page is useless.
- (3) Because of problems of broken scripts many web surfers disable JavaScript support in their browser.
- (4) Scripts can run slowly & complex scripts can take a long time to startup.

↳ Key Points that you need to apply in all scripts:-

- (1) Each line code is terminated by a semicolon.
- (2) Blocks of code that are to be executed together as a unit.
- (3) Functions have parameters which are passed inside paranthesis
- (4) Variables are declared using keyword var.
- (5) Scripts require neither a main function nor an exit condition.

→ VARIABLES:-

A Variable is a named value that you use in your programs.

↳ Variable Names:-

Rules governing how you name your variables in JavaScript:

- a) Names must begin with a letter, digit, or underscore.
- b) you cannot use spaces in names;
- c) names are case-sensitive, so ~~that~~ ~~for~~ ~~ed~~,
- d) you cannot use a reserved word as a variable name.

↳ Data Types:-

JavaScript has four data types.

- a) Numeric → basic numbers
- b) Strings
- c) boolean
- d) null.

↳ Creating Variables:-

we use keyword var before the variable name.

Ex: var a;

↳ Using Variables in JavaScript

→ OBJECTS IN JAVASCRIPT:-

'new' operator is used to create objects.

↳ Document Object:-

To display some info. on the screen.

Syntax: document.write ("any msg");

(or)

document.writeln ("any msg");

↳ Window Object:-

It is a small window dialog that is having a title bar, message, a default icon, with one or more command buttons.

Syntax:

```
window.alert ("String message",  
              "default Value");
```

Ex:

```
<script language="JavaScript">  
    var username;  
    username = window.prompt ("Enter username");  
    document.writeln ("Welcome to"  
                      + username);  
</script>
```

Ex: Program for adding two no's:-

```
<html>  
<head>  
<title> Addition of two no's </title>  
<script language="JavaScript">  
    var fnum, snum, num1, num2, sum;  
    fnum = window.prompt ("Enter 1st no", "0");  
    snum = " " " " " 2nd " "  
    num1 = parseInt (fnum);  
    num2 = parseInt (snum);
```

```

sum = num1 + num2;
document.writeln("<h1>The sum is " +
sum + "</h1>");
</script>
</head>
<body>
<p>Click for running script
again </p>
</body>
</html>

```

→ Math Object :-

used to perform mathematical calculations. Methods are below:

Name	Purpose	Example.
Math.abs(x)	Returns absolute value.	Math.abs(-43)=43
Rounding & Truncation Functions.		
Math.ceil(x)	Gives nearest integer not less than x.	5.8 = 6 5.0001 = 6 -9.8 = 9
Math.floor(x)	Gives nearest integer not greater than x.	5.8 = 5 5.0001 = 5 -5.8 = 6
Math.round(x)	Rounds x closer to nearest Integer	5.8 = 6 5.4 = 5

Name	Purpose	Example.
Comparison Functions		
Math.max(x,y)	Gives max. value of x & y.	30, 50 = 50
Math.min(x,y)	Gives min value of x & y.	30, 50 = 30.
Logarithmic Related Functions.		
Math.log(x)	Natural logarithm of base e of x.	Math.log(2).
Math.exp(x)	exponential of given no.	Math.exp(2).
Math.pow(base, exp)	Base exp.	2, 2 = $2^2 = 4$
Trigonometric Functions		
Math.sin(x)	Sine value of x	Sin 0.0 = 0
Math.cos(x)	Cosine value of x	Cos 0.0 = 1
Math.tan(x)	tangent value of x.	tan 0.0 = 0.
Miscellaneous Functions		
Math.sqrt(x)	Square root of x	9 = 3.
Math.random(x).	generates a random No. b/w 0 & 1.	

→ String Object

A string is a collection of characters.

Function Name	Description
Conversion Functions.	
str.toLowerCase()	Converts to lower case.
str.toUpperCase()	Converts to upper case.
str.toString()	Converts to string.
Searching Methods.	
str.charAt(pos)	Returns character at given posn.
str.charCodeAt(pos)	Returns character code at given posn.
str.indexOf(str1)	find position of str1 in str from starting posn.
str.indexOf(str1, n)	finds posn of str1 in str at nth posn.
str.lastIndexOf(str1)	finds posn of str1 in str for last occurrence, start from end to str.
Obtaining Strings.	
str.substr(m, n)	Extracts n chars from mth posn.
str.substring(m, n)	Extracts chars from m to n but not including nth character.

→ Date Object :-

Method	description.
toString()	gives string rep. of date.
valueOf()	time in no. of ms from Jan 1 1970.
getDate()	Returns 1 to 31, day of Months.
getDay()	Returns 0 to 6, sun to sat-day.
getMonth()	Returns 0 to 11, Jan to dec.
getFullYear()	Returns 4 digit year no.
getHours()	Returns 0 to 23
getMinutes()	0 to 59.
getSeconds()	0 to 59.
setDate(v)	sets date, day, Month, year
setDay(v)	
setMonth(v)	
setFullYear(y, m, d)	
setHours(v)	set hours, minutes, seconds, time or milliseconds.
setMinutes(v)	
setSeconds(v)	
setTime(v)	
setMilliseconds(v)	

↳ Userdefined Objects:-

2- (8)

- ① new - used to create Objects
- ② this - to differentiate global Variables and those which are part of an Object but may have same name.
- ③ . (dot) - when referring to a property of an Object, whether a method or a variable a dot is placed b/w object name and Property.

Ex: <html>

<head>

<script language="JavaScript">

function Objdemo(){

popup("Hello");

myhouse = new house("Din Hacking", 2, 4);

alert(myhouse.name + "Has" +
myhouse.floors + "Floors and" +
myhouse.rooms() + "Rooms");

myhouse.leave("farewell");
}


```
function house(name, floors, beds)
```

```
{
```

```
  this.name = name;
```

```
  this.floors = floors;
```

```
  this.bedrooms = beds;
```

```
  this.rooms = floors;
```

```
  this.leave = popup;
```

```
}
```

```
function frooms() {
```

```
  var groundfloor = 3;
```

```
  var utilities = 2;
```

```
  var total = 0;
```

```
  if (this.floors <= 0) { total = 0; }
```

```
  else {
```

```
    if (this.floors == 1) {
```

```
      total = this.bedrooms + utilities;
```

```
    }
```

```
    else {
```

```
      total = (this.floors * utilities);
```

```
      total += groundfloor;
```

```
      total += this.bedrooms;
```

```
    }
```

```
    { return total; }
```

```
}
```

2

```
function popup(msg) {  
    alert(msg);  
}  
</script>  
</head>  
<body onload="Objdemo1">  
</body>  
</html>.
```

→ OPERATORS IN JAVASCRIPT :-

↳ Arithmetic Operators :-

Addition (+)

Subtraction (-)

Multiplication (*)

Division (/)

Modulus (%)

↳ Relational Operators:-

> → greater than

< → less than

>= → greater than or equal to

<= → less than or equal to

== → equals

!= → not equals.

Ex: Write a pgm to display result of various relational operators by comparing values 35 & 45. Display result in tabular form.

```
<html>
  <head>
    <script language="JavaScript">
      var x=35, y=45;
      document.writeln("<table border=1>
                          <tr>");
      document.writeln("<th>Comparison
                          operators");
      document.writeln("<th>==<th>!=
                          <th>>");
```

```

document.writeln("<th>> <th> <th> <th>");
document.writeln("<tr> <tr> <tr>");
document.writeln("x=" + x + ", y=" + y + "<td>"
    + (x==y) + "<td>" + (x!=y)
    + "<td>");
document.writeln("(x>y) + "<td>" + (x==y) +
    "<td>" + (x<y) + "<td>" +
    (x<=y) + "<tr>");

```

```

x=45, y=35;

```

```

document.writeln("<tr> <tr>");
document.writeln("x=" + x + ", y=" + y + "<td>"
    + (x==y) + "<td>" + (x!=y) +
    "<td>");
document.writeln("(x>y) + "<td>" + (x==y) +
    "<td>" + (x<y) + "<td>" +
    (x<=y) + "<tr>");

```

```

document.writeln("<table>");

```

```

</script>

```

```

<body> </body>

```

```

</html>.

```


→ Assignment Operators:-

$+=$

$-=$

$*=$

$/=$

$\%=$

Syntax:

Variable operator = expr.

↓

Variable = Variable operator
expr.

→ Increment & decrement Operator:-

$++$, $--$ Ex: if $a=10$.

Operator	Called	Ex.	Result.
$++$	Preincrement	$S = ++a$	$S=11, a=11$
$++$	Postincrement	$S = a++$	$S=10, a=11$
$--$	Predecrement	$S = --a$	$S=9, a=9$
$--$	Postdecrement	$S = a--$	$S=10, a=9$

→ Logical Operators:-

boolean1	boolean2	logical AND &&	logical OR 	logical Not !
F	F	F	F	T
F	T	F	T	
T	F	F	T	
T	T	T	T	F

→ CONTROL STRUCTURES IN JAVASCRIPT

Selection Structures

if, if-else, switch.

Repetition Structures

while, For, Do-while.

↳ Simple if :-

if (condn)
 stmts;

Ex: pgm for finding 2 nos are equivalent.

```
<html> <head>
  <script language="javascript">
    val a,b;
    a= parseInt(window.prompt("enter A", "0"));
    b= parseInt(window.prompt("enter B", "0"));
    if (a==b)
      window.alert("both are equal");
  </script>
</head> </html>
```


↳ if-else:-

```
if (condn)
    true stmts;
else
    false stmts;
```

↳ while Control Structure:-

```
while (expr)
    { stmts; }
```

Ex: pgm for finding sum of 10 no's.

<html>

<head>

<script language="JavaScript">

var sum=0, i=1;

while (i<=10)

{

sum = sum + i;

i++;

}

document.writeln ("sum of first 10
integers " + sum);

</script>

</head>

</html>

2. (12)
→ for statement:-

for (initialisation; condition; incr/decr);

Ex: write a program to display table for given numbers.

```
<html>
```

```
<head>
```

```
<script language="JavaScript">
```

```
var n, i;
```

```
n = parseInt(window.prompt("Enter  
which table you want to  
display", "0"));
```

```
document.writeln("Table for " + n +  
" <br>");
```

```
for (i = 1; i <= 10; i++)
```

```
document.writeln(n + " * " + i + " = " +  
(i * n) + "<br>");
```

```
</script>
```

```
</head>
```

```
</html>.
```

→ do while statement:-

```
do { stmts }
```

```
while (condn);
```


→ switch statement:-

```
switch (expr)
{
    case 1: stmt; break;
    :
    case n: stmt; break;
}
```

→ FUNCTIONS IN JAVASCRIPT:-

Functions allows you to modularize a program. When you are divided into modules, then it is very easy to manage entire programs.

Functions are of 2 types:

- a) User defined functions
- b) Predefined / library functions.

→ User defined functions:-

```
function fun_name (parameter list)
{
    declarations;
    statements;
    :
    return [expression];
}
```

Ex: Write a script to display square & cube of given no.

<html>

<head>

<script language="JavaScript">

var a;

a = parseInt(window.prompt("Enter no", "0"));

document.writeln("square of given no is" + square(a));

document.writeln("Cube of given no is" + cube(a));

function square(k).

{ return k*k;

}

function cube(k)

{ return k*k*k;

}

</script>

</head>

</html>

→ Predefined / Library Functions:-

~~fun~~
eval - takes a string arg. representing JavaScript code to be executed. The interpreter evaluates code & executes dynamically.

isNaN - Not a Number - true
 Number - false.

parseFloat - Converts to floating value.

parseInt - Converts to integer value.

→ ARRAYS IN JAVASCRIPT:-

declaring syntax:-

var var-name; ex: var num;

allocating memory:-

num = new Array(12);

initializing Array:-

num = new Array(10, 20, 30, 40);

(or)

var num = [10, 20, 30, 40];

```

Ex: <html>
    <head>
        <script language="JavaScript">
            var ar;
            ar = new Array (5,10,15,20);
            for (var i=0; i<ar.length; i++)
            {
                document.writeln ("Array [" + i + "] → "
                                   + ar[i] + "<br>");
            }
        </script>
    </head>
</html>

```

→ EVENT HANDLING IN JAVASCRIPT:

Event	Handler	Description.
blur	onBlur	The c/p focus is moved from Object, usually when moving from a field or from the form itself.
change	onChange	The value of field in a form has been changed by user entering/deleting data.

Event	handler	Description.
click	onClick	Mouse is clicked over an element of a page.
dblclick	onDbClick	When element or a link is clicked twice.
dragdrop	onDragDrop	A system file is dragged with a mouse and dropped onto browser.
focus	onFocus	Input focus is given to an element opp. to blur.
keydown	onKeyDown	A key is depressed but not released.
keypress	onKeyPress	A key is pressed.
keyup	onKeyUp	A depressed key is released.
load	onLoad	A page is loaded into browser.
mousedown	onMouseDown	A mouse button is pressed.
mousemove	onMouseMove	mouse/cursor is moved.
mouseout	onMouseOut	mouse pointer moves off an element.
mouseover	onMouseOver	mouse pointer is moved over an element.

Event	Handler	Description.
Mouse up	onMouseUp	Mouse button is released.
move	onMove	A window is moved / Maximised / Restored.
Resize	onResize	Window is resized using script.
select	onSelect	A field on form is selected by clicking mouse / tabbing keyboard.
submit	onSubmit	A form is submitted.
unload	onUnload	user leaves webpage.

Javascript Objects and Event Handlers.

Object	Event handler
Window	onLoad, onUnload, onBlur, onFocus
Link	onClick, onMouseOut, onMouseOver
image	onLoad.
form	onReset, onSubmit.
text	onBlur, onChange, onFocus, onSelect.
textarea.	onBlur, onChange, onFocus, onSelect.
Pass word	onBlur, onChange, onFocus, onSelect.

Object	EventHandler
button	onClick
reset	onClick
submit	onClick
radio	onClick
checkbox	onClick
select	onBlur, onChange, onFocus

⇒ DYNAMIC HTML WITH JAVASCRIPT

DHTML is well structured HTML code which adheres to the standards, it is stylesheets used to present neutral formatting control and finally it is use of script to make text, images and style elements active.

→ DATA VALIDATIONS

1-(16)

Validation is process of ensuring that some data might be correct data for a particular application. Broadly speaking data validation is process of ensuring that users submit only set of characters which you require. It is not the process of ensuring that data is in any way accurate.

Ex: <html>

<head>

<title> Validation Example </title>

<script language="JavaScript">

function validate (var a)

{
~~var x = document.forms.~~

if (a.value == '')

alert("please enter all fields");

}

</script>

</head>

<body>

<center> Login form </center>

<table>

<tr> <td> Username

<td> <input type="text" name="Username" />

</td>

<tr> <td> Password

<td> <input type="password"

name="password"

</td>

onfocus="validate(Username)"

<tr> <td>

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

onfocus="validate(password)"

<input type="reset" value="Reset" />

</td>

</table>

</body>

</html>

↳ opening a new window:-

2- (17)

<html>

<head>

<script language="JavaScript">

function load(url){

var next = url;

newwin = open("url", "newwin",

'status=0, toolbar=0, resizable=0,
width=256, height=132');

}

</script>

</head>

<body>

<p>

show next page </p>

</body>

</html>.

↳ Messages & Confirmations

ex: `prompt("Enter your name", "");`
`confirm("Are you sure");`
`alert("A warning");`

↳ status Bar :-

```
<html>
  <head>
    <script language="javascript">
      function init() {
        self.status = "Message is";
      }
    </script>
  </head>
  <body onload="init()">
    <h1>and status bar says</h1>
  </body>
</html>
```

↳ Writing to a Different Frame

↳ Writing to a Different Frames

↳ Rollover Buttons :-

while browsing websites our eyes get often struck at certain buttons which carries certain images on them. Whenever the mouse cursor is brought over them, the image on these buttons gets changed. such buttons are called as Rollover buttons.

Ex:

```
<html>
<head>
<title> Rollover Buttons </title>
<script language="javascript">
if (document.images)
{
button1 = new Image;
button2 = new Image;
```


→ BASIC XML

XML (extensible Markup language) facilitates users to create their own tags.

XML is a meta language that describes contents of document. So, these tags can be called as self-describing data tags.

↳ Uses of XML

- 1) Simplifies data exchange procedure.
- 2) Easy to organize document.
- 3) Tags/document elements are reusable.
- 4) Provides consistency in display of info.

↳ Applications of XML

- 1) Electronic Commerce
- 2) Financial Funds Transfer
- 3) Multimedia Messages & Messaging Xchange
- 4) Better environment for data transfer.

XML	HTML
1) User defined tags 2) User has ctrl on tags 3) Separates content from presentation 4) Self describing data can be possible 5) we can generate new markup languages using XML 6) It is case sensitive 7) Root element is user defined and only one root element is allowed.	1) predefined tags 2) No ctrl 3) specifies presentation 4) Not possible. 5) No such possibility 6) Not case sensitive 7) Root element is <html>

XML declaration always starts with XML keyword.

<?xml Version="1.0" ?>

Ex: <?xml version="1.0"?>

<authorlist>

<name>

<firstname> abc </firstname>

<lastname> def </lastname>

</name>

<name>

<firstname> ghi </firstname>

<lastname> jkl </lastname>

</name>

</authorlist>

Handling special characters in XML

character	Name	Reference.
&	ampersand	& amp;
<	less than	& lt;
>	greater than	& gt;
"	quote	& quot;
'	apostrophe	& apos;

↳ XML elements :-

- ① Nesting tags
- ② Case sensitive
- ③ Empty tags
ex: `<ingredient />`
- ④ Attributes
- ⑤ Control Information
- ⑥ Comments
`<!-- comment -->`
- ⑦ Processing Instrn.
`<?xml version="1.0" ?>`
- ⑧ Document type definition.
- ⑨ Entities.

→ DOCUMENT TYPE DEFINITION (DTD):-

DTD defines structure of XML document. The relationship b/w root elements, container element and sub elements, is given in DTD. DTD are started with DOCTYPE. DTD's are optional in XML, but recommended for clarity purpose.

Syntax:

<DOCTYPE name_doc SYSTEM "path">

↳ elements in DTD:-

The XML document is composed of a no. of elements.

Syntax:

<!ELEMENT name & defn/type>

Ex:

<!ELEMENT authorlist (name*)>

* → appears zero or more times.

, → separates sequence of items.

None → item appears only one.

() → Encloses a group of items

| → separates a set of alternative

+ → item appears at least once.

? → appears once or not at all.

↳ Attributes:-

Syntax:

<[ATTLIST element attr-name type
Modifier]>

element → element list of XML doc.

attr-name → name of attributes
for element

type → CDATA (character data)
ID (identifier) etc.

modifiers → #IMPLIED → attribute not required/not specific

#REQUIRED → attribute must be present

#FIXED 'val' → attribute must be present and always contains this value.

Ex: <!Element slideshow (<slide+*)>

<!Attlist slideshow

title CDATA #REQUIRED

data CDATA #IMPLIED.

author CDATA "Unknown">

<!Element slide (title,item*)>

Save as slideshow1.dtd.

<?xml version='1.0'>

<!DOCTYPE Slideshow SYSTEM "slideshow1.dtd">

<Slideshow

title="Sample Slideshow"

data="Date of publication"

author="Yours truly">

<slide type="all">

<title>wakeup</title>

</slide>

<slide type="all">

<title>Overview</title>

<item>why this</item>

<item>Good</item>

</slide>

</Slideshow>

↳ Entities in DTD:-

Some markup elements can contain complex data. These elements are called entities.

1) Internal Entities:- Included in XML file.

These are used to create small pieces of data which you want to use repeatedly throughout your schema.

Ex: `<!ENTITY pos "pinch of salt">`

used as

`<item> finally add &pos; </item>`

2) External Entities:-

Ex: `<!ENTITY myimage SYSTEM "abc.jpg" NDATA JPG>`

SYSTEM - keyword shows that we have created data for our local application.

NDATA - tells the processor that we have created a notation for this type of data.

↳ Namespaces in DTD :-

A namespace is a way of keeping names used by applications separate from each other. No duplication of names will be there.

A namespace is included in XML in same way as DTD.

Ex: <staff>

<name> Chris Bates </name>

<dept>

<name> School of CSE </name>

</dept>

<room>

Here duplication of <name> is there.

In order to avoid we use namespaces.

This can also be written as:

<?xml version="1.0"?>

<!xml: namespace ns="url//namespaces/
staff"

prefix="name1">

<!xml: namespace ns="url//namespaces/dept"

prefix="name2">

<staff>

<name1: name> Chris Bates

</name1: name>

<dept>

<name2: name> School of CME

</name2: name>

</dept>

<room> 2322 </room>

</staff>

→ XML SCHEMA

The limitations of DTDs are overcome by XML Schema.

An XML Schema defines elements and attributes of the language and it is alternative to DTD. An XML Schema contains metadata or data dictionary that is used by an XML processor to transform XML document to another kind of document. These schema files are having extension .xsd.

Syntax:

```
<xsd:schema xmlns:xsd="...">  
</xsd:schema>.
```

This schema defines a namespace for its elements. Elements are some attributes those which

defines datatypes, all exist in that namespace. XML elements are defined within XML Schema as being either simple types/complex types.

- A Complex type contains other element or has attributes within a complex type the individual components forms a sequence which must be defined in schema documents.

Elements & attributes may have a datatype which is defined as an attribute of element decln.

- A Simple type defines an element which is a container for data.

Ex: Prepare XML Schema for:

Cookbook

└ Recipe

a) └ name

b) └ Ingredients

1) └ quantity

2) └ amount

3) └ unit

2) └ item.

<?xml version="1.0"?>

<xsd:schema xmlns:xsd="URL">

<xsd:element name="cookbook">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="recipe">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="name"

type="xsd:string"/>

<xsd:element name="ingredient">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="qty">

<xsd:complexType>

<xsd:attribute name="amount"

type="xsd:decimal"/>

<xsd:attribute name="unit"

type="xsd:string"/>

</xsd:complexType>

```

</xsd:element>
<xsd:element name="item"
  type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>.
```


→ PRESENTING XML (EXTENSIBLE STYLE SHEET)

Extensible Style Sheets is a language used to express style sheets which are then used to present XML document.

XSL stylesheets are not the same as HTML cascading stylesheets. Rather than create a style for a specific XML element or class of elements, with XSL a template is created. This template is used to format XML elements which match a specified pattern.

XSL elements:-

Syntax:

xsl:apply-templates.

3-9
1) xsl:apply-templates:

A style sheet can contain a no. of templates. Each of these can be directed toward a different output format.

2) xsl:attribute:

Creates an attribute node.

3) xsl:copy

A CATA section is added to o/p doc.

4) xsl:choose

The condition of an element can be tested.

ex: xsl:when.

5) xsl:comment

A comment is added to o/p document.

6) xsl:copy

copies target node from i/p src to o/p

7) xsl:define-template-set

A set of templates is defined.

- 8) `xsl: element`
creates an element in o/p
- 9) `xsl: entity-ref`
creates an entity reference in o/p.
- 10) `xsl: eval`
Evaluates a piece of text.
- 11) `xsl: for-each`
A single template is applied to a set of xml elements.
- 12) `xsl: if`
Boolean conditions can be tested
- 13) `xsl: node-name`
Name of current node is inserted into the output.
- 14) `xsl: otherwise`
Used for conditional testing of element (or) attribute values.
- 15) `xsl: script`
Global variables & functions can be declared within a template.

16) JSL: when

Used in condnal testing.

→ USING XML PROCESSORS

The Java API for XML processing (JAXP) is for processing XML data using applications written in Java Programming languages. JAXP leverages the parser standards SAX (Simple API for XML parsing) and DOM (Document Object Model) so that you can choose to parse your data as a stream of events or to build an Object rep. of it.

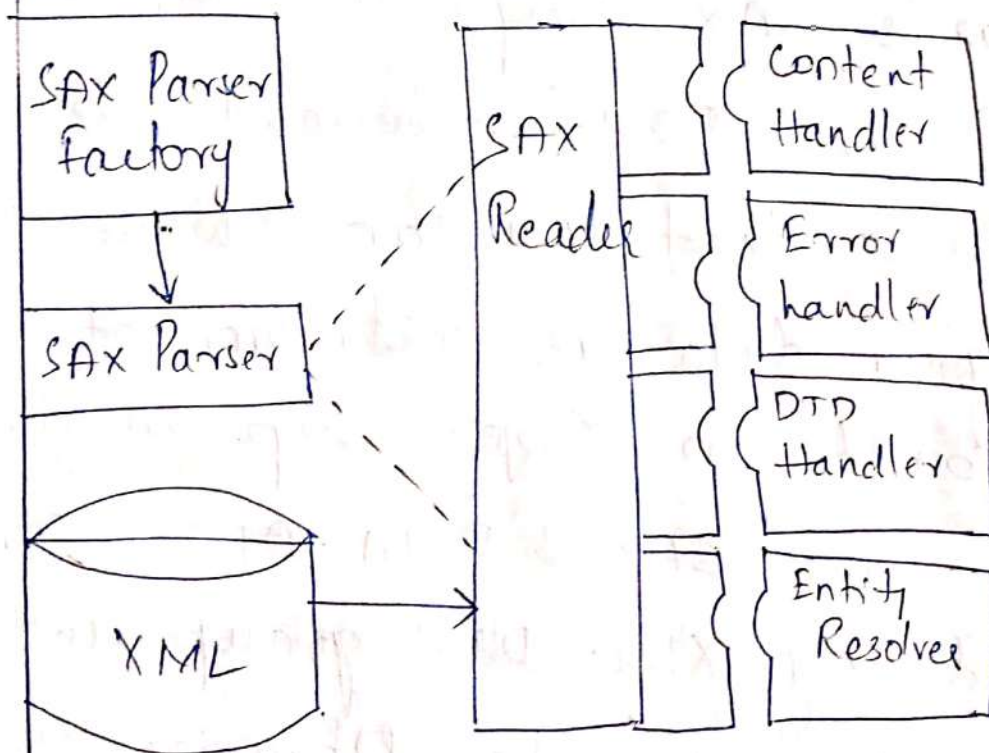
The SAX & DOM API's are defined by XML-DEV group and by W3C respectively. The libraries that define those API's are as follows:

a) `javax.xml.parsers`: which provides a common interface for SAX & DOM

b) `org.w3c.dom`: defines DOM as well as classes for all components of DOM.

c) `org.xml.sax`: Defines basic SAX API

↳ The Simple API for XML (SAX) API's:-



SAX API's

To start the process, an instance of SAXParserFactory class is used to generate an instance of parser.

The parser wraps a SAXReaderObject. When parser's parse() method is invoked, the reader invokes one of several callback methods implemented in applications. Those methods are defined in interfaces.

ContentHandler, ErrorHandler, DTDHandler, and EntityResolver.

SAXParserFactory :-

A SAXParserFactory Object creates an instance of parser determined by system property, `javax.xml.parsers.SAXParserFactory`.

SAXParser :-

This interface defines several kinds of parse() methods. In general if you pass an XML data source and a DefaultHandler Object to parser, which processes XML and invokes appropriate methods in handler Object.

SAXReader :-

It carries out conversion with SAXEvent handlers you define.

DefaultHandler :-

implements ContentHandler, ErrorHandler, DTDHandler and EntityResolver interfaces.

ContentHandler :-

Method such as startDocument(), endDocument(), startElement() and endElement() are invoked when an XML tag is recognized.

ErrorHandler :-

Methods errors(), fatalError() and warning() are invoked in response to various parsing errors.

DTDHandler :-

Used when processing a DTD to recognize and act on declarations for an unpaired entity.

EntityResolver :-

The resolveEntity() method is invoked when parser must identify data identified by a URL.

Package	Description.
org.xml.sax	Defines SAX interfaces
org.xml.sax.ext	defines SAX extensions that are used for doing more sophisticated SAX processing.
org.xml.sax.helpers	Contains helper classes that makes easier to SAX
javax.xml.parsers	Defines SAXParserFactory class, which returns SAX parser.

SAX Packages

→ Document Object Model (DOM) Parser:-

DOM is an interface that exposes an XML document as a tree structure comprised of nodes. The DOM allows you to programmatically navigate tree and add, change, delete any of its elements.

Benefits of DOM:-

- 1) Move one part of document tree to another without destroying content.
- 2) Create elements and attach them to any point in document tree.
- 3) Organize and manipulate new/existing tree branches in a document.

Package	Description.
org.w3c.dom	Defines DOM pgming interfaces in XML doc's as specified by w3c.
javax.xml.parsers	Defines DocumentBuilderFactory class and DocumentBuilder class, which returns an Object that implements w3c document interface.

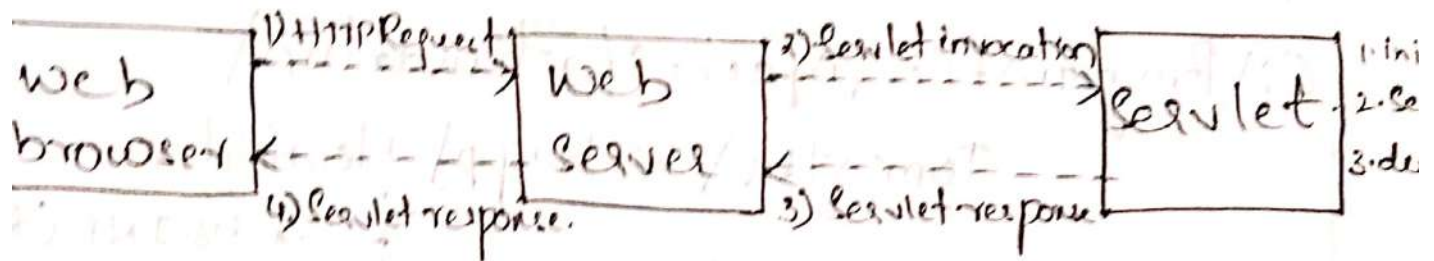
↳ Differences b/w SAX & DOM.

3-13

SAX	DOM
1) import javax.xml.parsers.*; " org.xml.sax.*; " org.xml.sax.helpers.*; 2) Parses node by node. 3) Doesn't store XML in memory. 4) We can't insert or delete a node. 5) Top to bottom traversing 6) SAX is an event based parser 7) SAX doesn't preserve comments. 8) SAX parser serves the client appn always only with pieces of document.	1) import javax.xml.parsers.*; " org.w3c.dom.* 2) stores entire XML document into memory before processing. 3) occupies more memory. 4) we can insert or delete nodes. 5) Traverse in any direction 6) DOM is a tree model parser. 7) DOM preserves comments 8) DOM parser always serves the client appn with entire document.

INTRODUCTION TO SERVLETS

A Servlet can be thought of as a server side applet. Servlets are loaded and executed by webserver. Servlet accepts a request from a client, performs some task and returns results to client (browser).



To develop and run servlets we need follow

- a) Java Development kit (JDK) installation.
- b) Java Servlet Development kit (JSDK) installation
- c) A web server
- d) A client application (generally a web browser)

→ Basic Servlet Structure:-

```

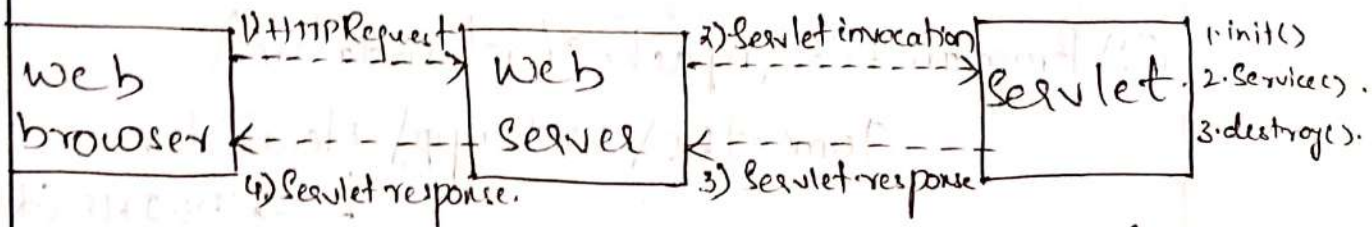
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ServletTemplate extends
    HttpServlet {

```


→ INTRODUCTION TO SERVLETS

A Servlet can be thought of as a server side applet. Servlets are loaded and executed by web server. Servlet accepts a request from a client, performs some task and returns results to client (browser).



To develop and run servlets we need following:

- Java Development kit (JDK) installation.
- Java Servlet Development kit (JSdk) installation
- A web server
- A client application (generally a web browser).

→ Basic Servlet Structure:-

```

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ServletTemplate extends
    HttpServlet {
    
```



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

```
{  
    PrintWriter out = res.getWriter();  
}
```

↳ Folder Structure :-

C:\Program Files\Apache Software Foundation\

Tomcat5\webapp\Newfolder

WEB-INF (folder)

classes (.class files)
(folder)

lib
(folder)

web.xml

all Java files

html files
etc.

→ Write a program to print "Hello world" using Servlets

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

public class hello extends HttpServlet {

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

        PrintWriter pw = res.getWriter();
        pw.println("Hello world");

    }

}
```

→ LIFECYCLE OF A SERVLET

Three methods are central to life cycle of a Servlet:

init():- initialization parameters to Servlet.

service():- process the request HttpServletRequest

destroy():- destroys any resources such as file handles that are allocated for Servlet.

→ SERVLET API:-

2 packages contain classes and interfaces that are required to build servlets.

These are: `javax.servlet` and `javax.servlet.http`.

→ THE javax.servlet PACKAGE

This package contains a number of interfaces and classes that establish the framework in which servlets operate. The following table summarizes core interfaces that are provided in this package. The most significant of these is `Servlet`.

<code>Servlet</code>	- Declares lifecycle method for a servlet
<code>ServletConfig</code>	- Allows servlets to get initialization parameters
<code>ServletContext</code>	- Enables servlets to log events & access info. abt their environment.
<code>ServletRequest</code>	- Used to read data from a client request
<code>ServletResponse</code>	- Used to write data to a client response.

The following table summarizes core classes that are provided in javax.servlet package.

GenericServlet -	Implements Servlet and ServletConfig interfaces.
ServletInputStream -	Provides an inp stream for reading requests from a client.
ServletOutputStream -	Provides an o/p stream for writing responses to a client.
ServletException -	Indicates a Servlet error occurred.
UnavailableException -	Indicates a Servlet unavailable

↳ Servlet Interface :-

All Servlet must implement Servlet Interface.

- a) void destroy() → called when Servlet is unloaded.
- b) ServletConfig getServletConfig() → Returns a ServletConfig object that contains any initialization parameters.

c) String getServletInfo() → Returns a string described the servlet.

d) void init(ServletConfig sc) → called when servlet throws ServletException, is initialized. Initialization parameters for the servlet can be obtained from sc.

e) void service(ServletRequest req, ServletResponse res) → throws ServletException, IOException.

Called to process a request from a client. The request from client can be read from req. The response to client can be written to res. An exception is generated if a servlet or IO problem occurs.

Q1)

↳ ServletConfig Interface :-

It is implemented by Servlet container. It allows a servlet to obtain config. data when it is loaded.

a) ServletContext getServletContext()

- Returns context for this Servlet.

- b) `String getInitParameter(String param)`
 - Returns value of initialization parameter named `param`.
- c) `Enumeration getInitParameterNames()`
 - Returns an enumeration of all initialization parameter names.
- d) `String getServletName()`
 - Returns name of invoking servlet.

↳ ServletContext Interface :-

It is implemented by Servlet container. It enables Servlets to obtain info. about their environment.

- a) `Object getAttribute(String attr)`
 - Returns value of server attribute named `attr`.
- b) `String getMimeType(String file)`
 - Returns MIME type of file.
- c) `String getRealPath(String vpath)`
 - Returns real path that corresponds to virtual `vpath`.

String getServletInfo()

- Returns info abt server.

void log(String s)

- Writes s to Servlet log.

void log(String s, Throwable e)

- Writes s and stack trace for e to Servlet log.

void setAttribute(String attr, Object val)

- sets attribute specified by attr to value passed in val.

ServletRequest Interface:-

Implemented by Servlet Container.

It enables a Servlet to obtain info. abt a client request.

Object getAttribute(String attr)

- Returns value of attribute named attr.

String getCharacterEncoding()

- Returns character encoding request.

int getContentLength()

- Returns size of request. The value -1 is returned if size is unavailable.

- d) `String getContentType()`
 - Returns type of request. A null value is returned if type cannot be determined.
- e) `ServletInputStream getInputStream()` throws `IOException`
 - Returns a `ServletInputStream` that can be used to read binary data from the request.
- f) `String getParameter(String pname)`
 - Returns value of parameter named `pname`.
- g) `Enumeration getParameterNames()`
 - Returns an enumeration of parameter names for this request.
- h) `String[] getParameterValues()`
 - Returns an array of strings.
- i) `String getProtocol()`
 - Returns a description of protocol used.
- j) `BufferedReader getReader()`
 - Returns a `BufferedReader` object that can be used to read text from request.

- k) `String getRealPath (String vpath)`
 - returns a real path corresponding to virtual path `vpath`.
- l) `String getRemoteAddr()`
 - returns String equivalent of IP address of client.
- m) `String getRemoteHost()`
 - returns name of client.
- n) `String getServerName()`
 - returns name of server.
- o) `int getServerPort()`
 - returns port number on which request was received.

① Servlet interface:-

5-6.

interface Servlet {

void destroy();

Servlet Config getServletConfig();

String getServletInfo();

void init (Servlet Config config)

throws ServletException,

UnavailableException;

void service (ServletRequest req,

ServletResponse res) throws

Servlet Exception, IOException;

}

② Servlet Config interface:-

interface Servlet Config {

Servlet Content getServletContent();

String getInitParameter (String param);

Enumeration getInitParameterNames();

}

③ ServletContext Interface:-

```
interface ServletContext {  
    Object getAttribute (String attr);  
    String getMimeType (String file);  
    String getRealPath (String vpath);  
    String getServletInfo ();  
    void log (String s);  
    void log (String s, Throwable e);  
    void setAttribute (String attr,  
        Object val);  
}
```

④ ServletRequest Interface:-

```
Interface ServletRequest {  
    Object getAttribute (String attr);  
    String getCharacterEncoding ();  
    int getContentLength ();  
    String getContentType ();  
    ServletInputStream getInputStream ()  
        throws IOException;  
    String getParameter (String pname);  
    Enumeration getParameterNames ();
```



```

String() getParameterValues();
String getProtocol();
BufferedReader getReader();
String getRealPath (String ~path);
String getRemoteAddr();
String getRemoteHost();
String getServerName();
int getServerPort();
}

```

⑤

↳ ServletResponse Interface :-

This interface enables a Servlet to form responses for client. This interface is implemented by Servlet.

Interface ServletResponse {

String getCharacterEncoding();

ServletOutputStream getOutputStream()

throws IOException;

PrintWriter getWriter() throws

IOException;


```
void setContentLength (int size);  
void setContentType (String type);  
String }
```

a) String getCharacterEncoding
- returns character encoding for response.

b) ServletOutputStream getOutputStream()
- returns a ServletOutputStream object that can be used to write binary data to response.

c) PrintWriter getWriter()
- returns a PrintWriter object that can be used to write character data to response.

d) void setContentLength (int size)
- used to set content length of response to size.

e) void setContentType (String type)
- used to set contentType of response to type.

↳ GenericServlet Class:

The GenericServlet class implements Servlet & ServletConfig interfaces. This method provides basic implementation for servlet lifecycle methods: init(), service & destroy().

class GenericServlet implements Servlet, ServletConfig, Serializable

```
{
    GenericServlet();
    void destroy();
    String getInitParameter(String name);
    Enumeration getInitParameterNames();
    ServletConfig getServletConfig();
    ServletContext getServletContext();
    String getServletInfo();
    void init(ServletConfig config) throws ServletException;
    void log(String str);
    void service(ServletRequest req, ServletResponse res) throws ServletException, IOException;
}
```


- a) `Servlet()`
 - is default constructor.
- b) `void destroy()`
 - is called before unloading a Servlet.
- c) `String getInitParameter(String name)`
 - returns all value of named initialization parameter.
- d) `Enumeration getInitParameterNames()`
 - returns all Servlet initialization parameter names as an enumeration of String objects. If no parameter exist it returns an empty enumeration.
- e) `ServletConfig getServletConfig()`
 - returns Servlet's `ServletConfig` object.
- f) `ServletContent getServletContent()`
 - returns Servlet's `ServletContent` object.
- g) `String ServletInfo()`
 - returns a String that describes the Servlet.

h) void init(ServletConfig config)
- is called by server after the Servlet is first loaded into server's memory space & before service method is called.

i) void log(String str)
- writes str to log file.

j) void service(ServletRequest req, ServletResponse res)

②

↳ ServletInputStream class :-

This class extends InputStream.
It provides an input stream that a Servlet developer can use to read data from a client request.

```
class ServletInputStream extends InputStream  
{  
    protected ServletInputStream();  
    public int readLine(byte b[], int offset,  
                        int len) throws IOException.  
}
```

a) protected ServletInputStream()

6 - is a default constructor.

b) public int readLine(byte b[], int offset, int len)

- reads bytes from input stream into byte array b, starting at an offset in array given by offset.

②

↳ ServletOutputStream class:-

This class extends OutputStream. It is implemented by Servlet Container and provides an output stream that a Servlet developer can use to write data to a client response.

Class ServletOutputStream extends OutputStream {

protected ServletOutputStream();

public void print(boolean b) throws IOException;

 " " " char c "

 " " " double d "

 " " "

S-10

```

public void print (float f) throws IOException
    "      "      "      int i
    "      "      "      String s
    "      "      "      long l
    "      "      "      println ()
    "      "      "      (boolean b)
    "      "      "      char c
    "      "      "      double d
    "      "      "      float f
    "      "      "      int i
    "      "      "      long l
    "      "      "      String s
    }

```

①
 ↳ ServletException class:-
 This class indicates that a
 servlet problem has occurred.

```

ServletException()
ServletException (String str).

```


→ READING SERVLET PARAMETERS :-

login.html

<html>

<body>

<form name=form1 action="/param"
method=post>

Enter UserName

<input type=text name=name>

Enter Password

<input type=password name=pass>

</body>

</html>

param.java

import java.io.*;

import java.util.*;

import javax.servlet.*;

import javax.servlet.http.*;

public class param extends ~~Generic~~ Servlet

{

```

public void service(ServletRequest req,
    ServletResponse res) throws
    ServletException, IOException
{
    res.setContentType("text/html");
    PrintWriter pwr = res.getWriter();
    String name = req.getParameter("name");
    String pass = req.getParameter("pass");
    pwr.println("User Name is " + name + "<br>");
    pwr.println("Password is " + pass + "<br>");
}
}

```

→ Javan. Servlet. HTTP Package :-

Interface	Description.
HttpServletRequest	Enables Servlets to read data from an HTTP request
HttpServletResponse	Enables Servlets to write data to a HTTP response
HttpSession	Allows session data to be read & written.
HttpSessionBindingListener	informs an object that it is bound to or unbound from a session.

class	Description
Cookie	allows state info. to be stored on a client machine.
HttpServlet	Provides methods to handle Http request & responses
HttpSessionEvent	Encapsulates a session-changed event.
HttpSessionBindingEvent	Indicates when a listener is bound to or unbound from a session value, or that a session attribute changed.

①
→

Interface HttpServletRequest:-

This interface enables servlet to obtain info. abt a client request. It is implemented by Server.

```
public interface HttpServletRequest
    extends ServletRequest
{
```

```
    String getAuthType();
    Cookie[] getCookies();
    String getMethod();
    String getPathInfo();
```

```

String getQueryString();
String getRemoteUser();
String getRequestedSessionId();
String getRequestURL();
String getServletPath();
HttpSession getSession(boolean new);
}

```

- a) String getAuthType()
 - returns authentication scheme
- b) Cookie[] getCookies()
 - returns an array of cookie objects.
- c) String getMethod()
 - returns http method for client request (get/post/head etc)
- d) String getPathInfo()
 - returns extra path info associated with request.

- e) `String getQueryString()`
 - Returns query string in URL
- f) `String getRemoteUser()`
 - Returns name of user who issued this request.
- g) `String getRequestedSessionId()`
 - returns session id specified by client
- h) `String getRequestURI()`
 - Returns URI of request.
- i) `String getServletPath()`
 - Returns path of URI that refers to servlet.
- j) `HttpSession getSession (boolean new)`
 - Returns current session associated with user making request.

interface HttpServletResponse:-

This interface enables a servlet to formulate HTTP response to a client.

interface HttpServletResponse extends
ServletResponse

```

{
    void addCookie(Cookie cookie);
    boolean encodeURL(String url);
    void sendError(int code) throws IOException;
    void sendError(int code, String s) throws IOException;
    void sendRedirect(String url)
}

```

a) void addCookie(Cookie cookie)
- adds cookie to HTTP response.

b) boolean encodeURL(String url)
- determines if session ID must be encoded in URL identified as url. If so, returns modified version of url.

c) void sendError(int code)

- sends error code to client

d) void sendError(int code, String str)

- sends errorcode code, error msg str to client

e) void sendRedirect(String url)

- redirects response to specified location automatically setting the status code & location header.



③ HttpSession Interface:-

This interface enables a Servlet to read & write status info that is associated with a HttpSession.

Interface HttpSession {

Object getAttribute(String attr);

Enumeration getAttributeNames();

long getCreationTime();

String getId();

```

long   getLastAccessedTime();
boolean isNew();
void   removeAttribute (String str);
void   setAttribute (String attr,
                    Object val);
}

```

@Object getAttribute (String str):
 - Returns value associated with name passed in str.

(b) Enumeration getAttributeNames()

- Returns an enumeration of attribute names associated with session.

(c) long getCreationTime():

- Returns time (in ms since midnight, Jan 1 1970) when session was created.

(d) String getId()

- Returns session Id.

(e) long getLastAccessedTime()

- Returns time when client last made a request for this session.

(f) boolean isNew()

- Returns true if server created the session and it has not yet been accessed by client.

(g) void removeAttribute(String attr)

- Removes attribute specified by attr from session

(h) void setAttribute(String attr, Object val)
- Associates value passed in val with attribute name passed in attr.

ex: (Session tracking)

```
import java.io.*;
```

```
import java.util.*;
```

```
import javax.servlet.*;
```

```
import javax.servlet.http.*;
```

```
public class Session extends HttpServlet
```

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

```
{  
    HttpSession hs = req.getSession(true);  
    res.setContentType("text/html");
```

```
PrintWriter pw = res.getWriter();  
pw.println("<br>");  
Date date = (Date) hs.getAttribute("date");  
if (date != null)  
{  
    pw.println("Last Access: " + date + "<br>");  
}  
date = new Date();  
hs.setAttribute("date", date);  
pw.println("Current date: " + date);  
}  
}
```

Q
→

HttpSessionBindingListener Interface:-

This interface is implemented by objects that need to be notified when they are bound to or unbound from a HttpSession.

interface HttpSessionBindingListener extends EventListener

```
{  
    void valueBound(HttpSessionBindingEvent event);  
    void valueUnbound(" " " ");  
}
```


(a) void valueBound (HttpSession Binding Event event)

- is called when listener is bound to a session.

(b) void valueUnBound (HttpSession Binding Event event)

- when listener is unbound from a session.

①



Cookie Class:

A cookie is stored on client and contains status info. For ex: a cookie can save user's name address and other info. so that user does not need to enter this data each time.

A Servlet can write a cookie to client machine using addCookie() method of HttpServletResponse interface.

The data for cookie is then included in header of HttpServletResponse that is sent to client browser.

The names & values of cookies are stored on user's machine. Some of the info. that is saved includes:

- a) cookies name
- b) cookies value
- c) expiration date
- d) domain & path.

The expiration date determines when cookie is deleted from client machine.

```
class Cookie {  
    Cookie(String name, String val);  
    Object clone();  
    String getComment();  
    String getDomain();  
    int getMaxAge();  
    String getName();  
    String getPath();  
    boolean getSecure();  
    String getValue();  
}
```



```

int getVersion();
void setComment(String str);
void setDomain(String domain);
void setMaxAge(int secs);
void setPath(String path);
void setSecure(boolean secure);
void setValue(String val);
void setVersion(int ver);
}

```

a) Object Clone()

↳ - Returns a copy of cookie object.

b) String getComment()

- Returns a comment associated with cookie.

c) String getDomain()

- Returns domain associated with cookie.

d) int getMaxAge()

- Returns age in seconds allowed for cookie.

(e) String getName()

- Returns name of cookie.

(f) String getPath()

- Returns path limitation for this cookie.

(g) boolean getSecure()

- Returns a true value of cookie requires a secure connection, otherwise false.

(h) String getValue()

- Returns actual value of this cookie in string format.

(i) int getVersion()

- Returns version no. of this cookie.

(j) void setComment(String str)

- Sets the comment field of cookie.

(k) void setDomain(String domain)

- Specifies domain restriction pattern.

(2) void setMaxAge(int sec)
- specifies max. age of cookie in seconds before it expires.

(3) void setPath(String path)
- specifies path for cookie.

(4) void setSecure(boolean secure)
- If secure = true, cookie should

(5) void setValue(String val)
- assigns a new value to cookie

(6) void setVersion(int ver)
- assigns ver as new version no. of cookie.

(2)

→ HttpServlet Class:-

It is commonly used when developing servlets that receives and processes HTTP Request.

Class HttpServlet extends GenericServlet
{
public HttpServlet();

```
public void service(ServletRequest req,  
                    ServletResponse res) throws
```

```
ServletException, IOException;
```

```
public void service(HttpServletRequest req,  
                    HttpServletResponse res) throws  
ServletException, IOException;
```

```
protected void doDelete(HttpServletRequest req,  
                        HttpServletResponse res) throws ServletException,  
                        IOException;
```

```
protected void doGet(HttpServletRequest req,  
                    HttpServletResponse res) throws ServletException,  
                    IOException;
```

```
protected long getLastModified (" " );  
}
```

HTTPSessionEvent class:-

This encapsulates session events. It extends EventObject and is generated when a change occurs to session.

```
class HttpSessionEvent {  
    HttpSessionEvent(HttpSession session);  
    HttpSession getSession();  
}
```


① → HttpSessionBindingEvent class:-

Extends HttpSessionEvent. It is generated when a listener is bound to or unbound from a value in HttpSession Object.

```
class HttpSessionBindingEvent {  
    HttpSessionBindingEvent(HttpSession ses,  
                             String name);  
    "  
    "  
    "  
    Object val);  
    String getNames();  
    HttpSession getSession();  
    Object getValues();  
}
```

→ Examples for Cookies:-

Cookie.html

<html>

<body>

<form name=form1 method=post

action="/addCookieServlet">

Enter value for first cookie

<input type=text name=cookie1>

Enter value for second cookie

<input type=text name=cookie2>

<input type=submit>

</body>

</html>

addcookieservlet.java :-

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

public class addcookieservlet extends
    HttpServlet
{
    public void doPost(HttpServletRequest
        req, HttpServletResponse res) throws
        ServletException, IOException
    {
        String cookie1 = req.getParameter("cookie1");
        String cookie2 = req.getParameter("cookie2");
        Cookie c1 = new Cookie("Cookie1new",
                                cookie1);
        Cookie c2 = new Cookie("Cookie2new",
                                cookie2);
        res.addCookie(c1);
        res.addCookie(c2);
        res.setContentType("text/html");
        PrintWriter pw = res.getWriter();
        pw.println("Cookie1 new is set to "
                    + cookie1 + "<br>");
    }
}
```

```
pw.println("cookies new" + cookies);  
}  
}
```

getCookies.java:-

```
import java.io.*;  
import javax.servlet.*;  
import javax.servlet.http.*;  
public class get_cookies extends HttpServlet  
{  
    public void doGet(HttpServletRequest req,  
        HttpServletResponse res) throws ServletException,  
        IOException  
    {
```

```
        Cookie[] cookies = req.getCookies();  
        res.setContentType("text/html");  
        PrintWriter pw = res.getWriter();  
        pw.println("<b>");  
        for (int i = 0; i < cookies.length; i++) {  
            String name = cookies[i].getName();  
            String value = cookies[i].getValue();  
            pw.println("name = " + name + "  
            },  
            value = " + value);  
        }  
    }
```


→ HANDLING HTTP REQUEST AND RESPONSE

↳ Handling HTTP GET Request:

<html>

<body>

<form name="form1" action="/colorgetServlet">

 color:

<select name="color" size="1">

<option value="Red"> Red </option>

" " green green "

" " Blue Blue "

</select>

<input type="submit">

</form>

</body>

</html>

~~Handwritten~~
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ColorGetServlet extends HttpServlet
{

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

String color = req.getParameter("color");
res.setContentType("text/html");
PrintWriter pw = res.getWriter();
pw.println("<h2>The selected
color is : ");
pw.println(color);
pw.close();

}

→ JSP INTRODUCTION:-

Jsp (Java Server Pages) is a technology invented by sun Microsystems which allows the easy creation & maintenance of server side HTML pages, which can be used as both kind of Dynamic HTML and CGI replacement.

Jsp are built on top of Java Servlets and designed to increase the efficiency in which programmers and even nonprogrammers can create web content.

Jsp is a technology for developing webpages that include dynamic content.

Unlike a plain HTML page, which contains static content that always remains the same, a Jsp page can change its content based on any no. of variable items, including identity of users.

A Jsp page contains standard markup languages, elements such as HTML

page just like a regular webpage. A Jsp page also contains special Jsp elements that allow the server to insert dynamic content in the page. Jsp elements can be used for a variety of purposes, such as retrieving info. from a database or registering user preferences.

→ THE PROBLEM COMPARED WITH SERVLETS

A Jsp page is ultimately compiled to a servlet. Thus, a Jsp page is nothing but a servlet in certain sense. However, Jsp's can be more easily deployed, since they can live in any directory; also it is far easier to embed long HTML strings in a Jsp than a servlet.

→ JSP ADVANTAGES :-

- 1) JSP supports both scripting & element based dynamic content and allows developers to create custom tag libraries to satisfy appln specific needs.
- 2) JSP pages are compiled for efficient server processing.
- 3) JSP Pages can be used in combination with servlets that handle business logic.

→ JSP PROCESSING :-

Just as a webserver need a servlet container to provide an interface to servlets, the server needs a JspContainer to process JSP pages. Jsp container is responsible for intercepting request for JSP pages. To process all JSP elements in page, the container first turns the JSP page into a servlet. The container then compiles servlet class.

Converting Jsp page to a Servlet and compiling Servlet form the translation phase. The Jsp container is also responsible for invoking Jsp page implementation class to process each request and generate the response. This called request processing phase.

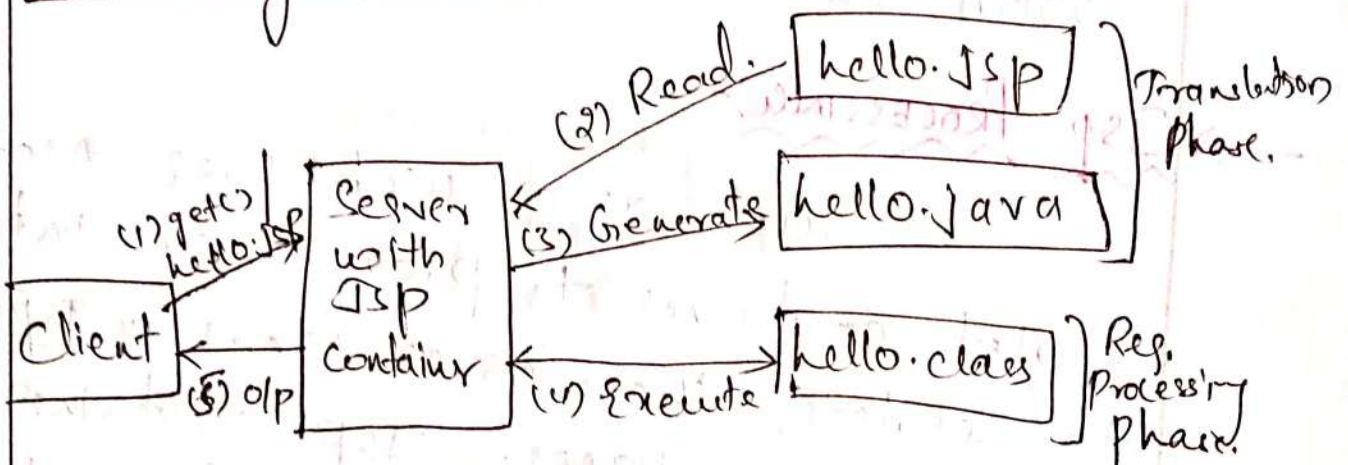


Fig: Jsp page translation & processing phases.

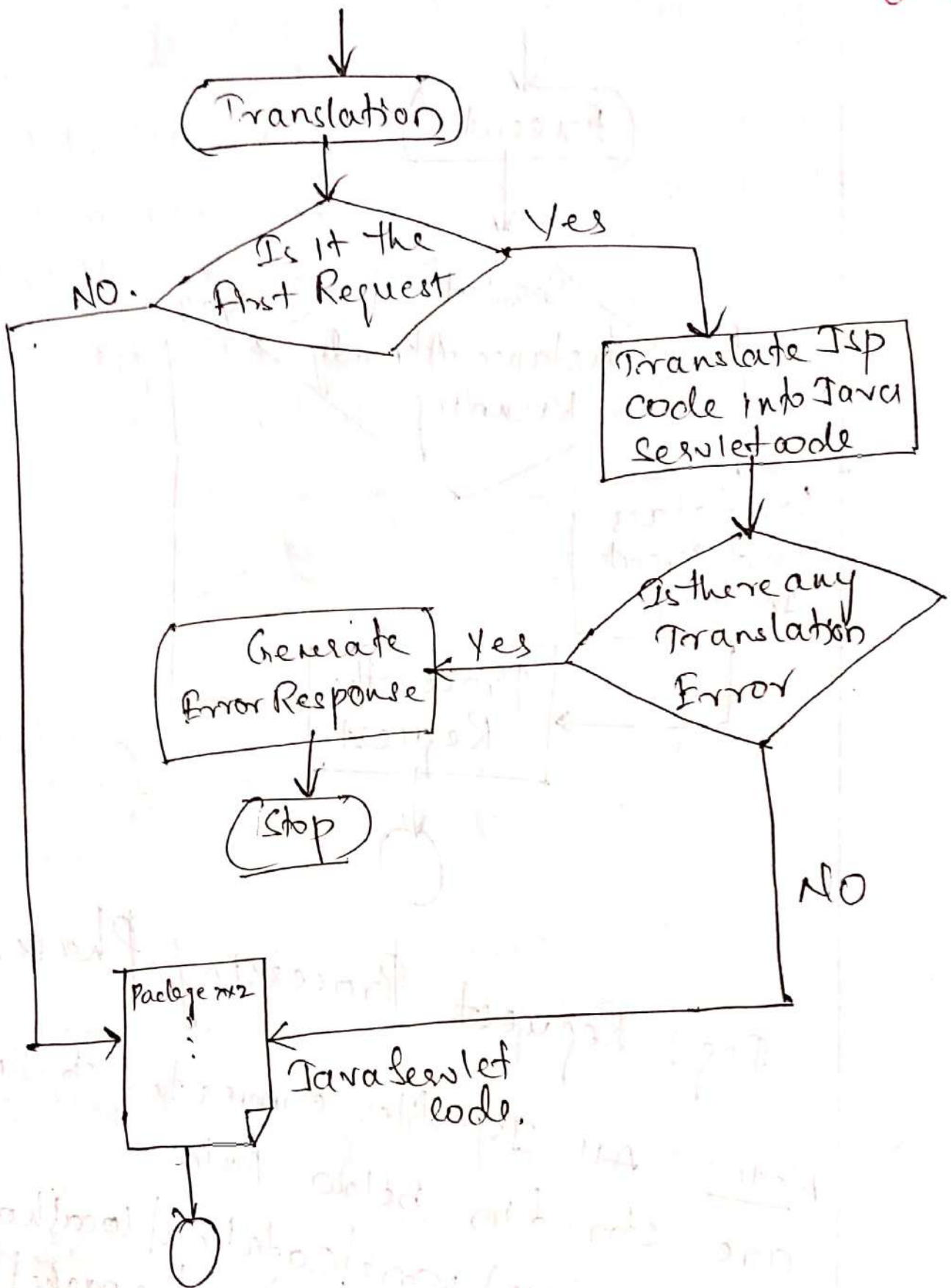


Fig: Translation phase.

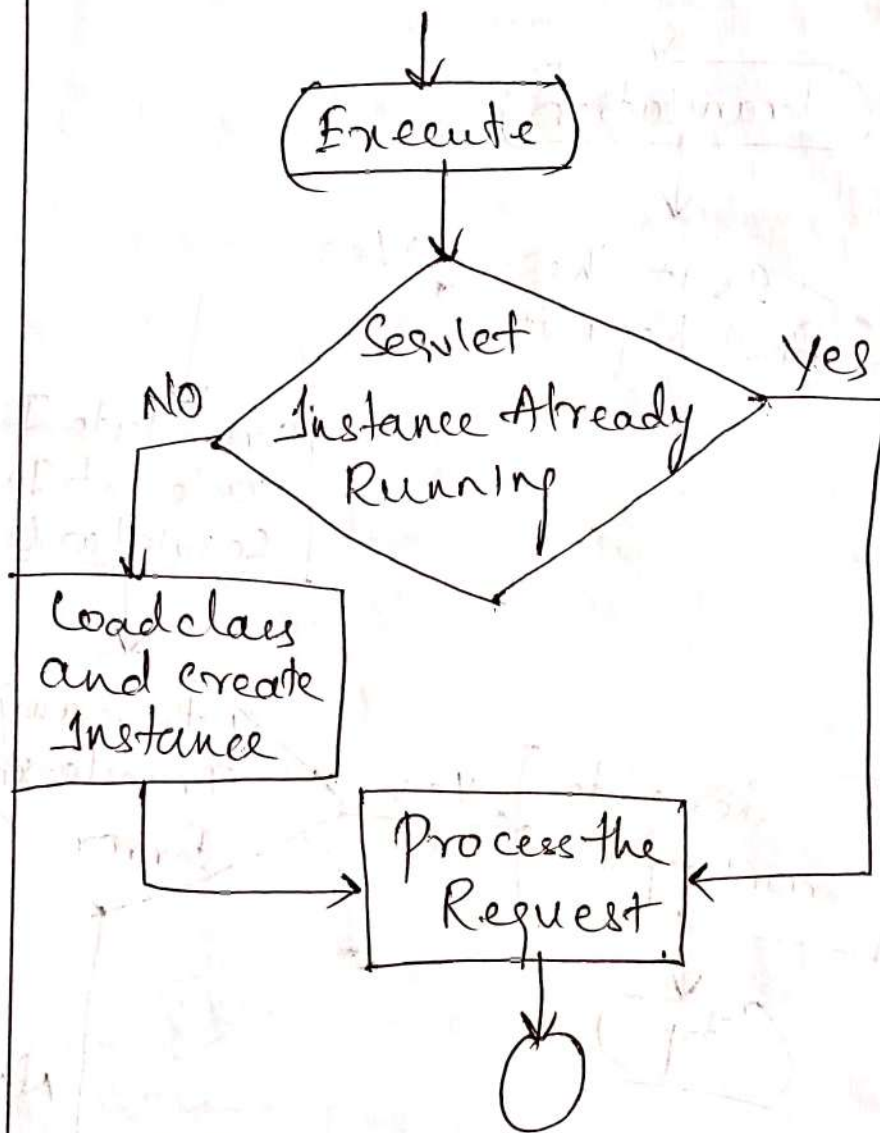


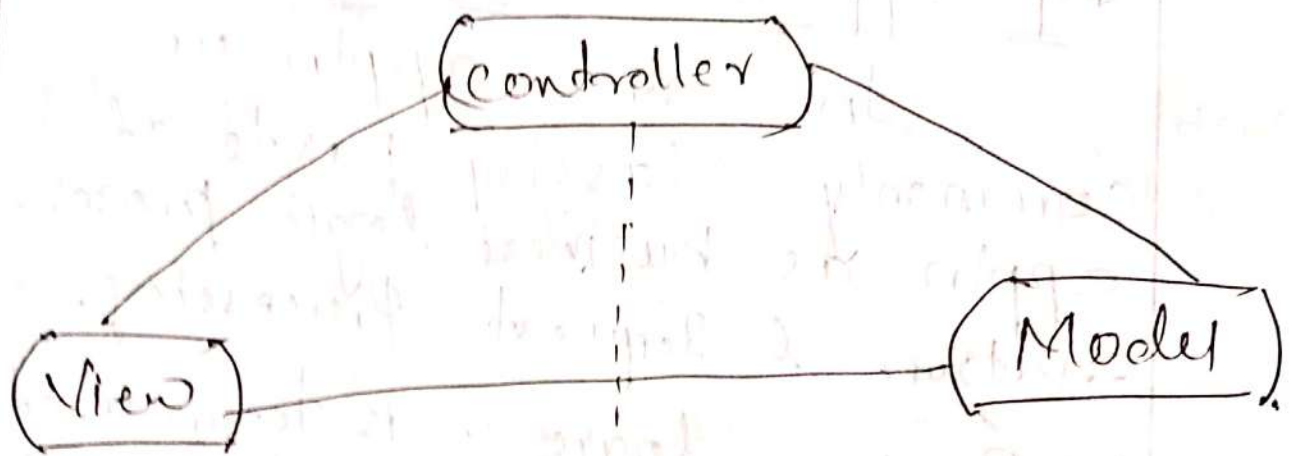
Fig: Request Processing Phase.

Note: All Jsp files converted to Java are stored in below path:

c:\pf\ASF\JS\work\catalina\localhost\jspbean\org\apache\Jsp.

→ Jsp Application Design WITH MVC:- 6-5

Definition :- MVC is a model used in swing components. Model represents data of component. View represents its appearance and controller specifies how it is controlled.



View represents visual appearance of component based on model data.

Model represents data the state of component.

When user interacts with component first of all controller will understand this & modifies model data accordingly.

The Goal of MVC arch. is to separate appln Object (Model), its representation to use (View) and way it is controlled by user (Controller).

Is p Application with MVC:-

In server appln, we commonly classify parts of appln as business logic, presentation & request processing.

- (a) Business logic :- is term used for manipulation of an appln data such as customer Product and Order Info.
- (b) Presentation refers to how appln data is displayed to user.
- (c) Request Processing is what ties the business logic & presentation together.

In MVC terms, the Model corresponds to business logic & data, the view is presentation and controller to request processing.

→ ANATOMY OF A JSP PAGE :-

Jsp page is almost like a regular web page with inclusion of dynamic behaviour through Jsp elements. Hence, we can say that a Jsp page has two components,

1) Jsp elements :- These are things that the Jsp container understands & translates. Hence, we can say that Jsp elements instructs Jsp container, what code to generate and how it should operate.

2) Template Data :- Everything else in page that the Jsp container does not understand is called template data. Actually these are static

portion of Jsp page which are passed through Jsp container unprocessed, to browser.

Ex: `<%@page import="java.util.Date"%>`] JSP element
`<html>`]
`<head>` `</head>` Template data.
`<body>`]
`<h3>` current time is `</h3>`]
`<% = new Date() %>`] JSP element
`</body>`] Template data.
`</html>`]

Jsp Elements:-

we have four types of Jsp elements.

They are,

- 1) Directive elements
- 2) Scripting elements
- 3) Action elements ~~Scripting elements~~

6-2

8.8 possible Directives let us to

- 1 Overall structure of servlet.
- 2 Top container

Directives give messages to Jsp container containing info. which are used during translation, to generate the corresponding Servlet- and to control some characteristics of Jsp page. Directives have following general form,

$\langle \% @directive [attribute_1 = 'value_1' \dots] \% \rangle$

$\downarrow$ $\downarrow$

Name of
directive optional.

There are three standard directives. They are:

a) Page b) include c) taglib.

- a) Page directive:- It is used to specify current page setting as a whole,
syntax: `%\@ page [attribut 1 = value 1 - -] %\>`

Page directive lets you define one or more following case-sensitive attributes:

- (i) import (ii) contentType (iii) pageEncoding
- (iv) session (v) isELIgnored (vi) buffer
- (vii) auto flush (viii) info (ix) errorPage
- (x) isErrorPage (xi) isThreadSafe (xii) language
- (xiii) extends.

Ex:
import:- lets you specify packages that should be imported by Servlet into which Jsp Page gets translated.

Ex: `<%@ page import="java.util.*"%>`
`<html>`

`<body>`

`<% System.out.println("Evaluating date");`

`Date date = new Date();`

`%>`

`Hello! Time is now <%= date %>`

`</body>`

`</html>`

↳ Jsp expression.

(ii) contentType:- gives MIME (Multipurpose Internet mail Extension) type of response document.

Common MIME types:

appln/msword → Microsoft word doc.
appln/pdf → Acrobat file.
text/html → html document etc,

Ex: `<%@page contentType="application/pdf"%>`

b) include directive:- It is used to include content from other resources during translation phase. Here, the page is inserted when jsp file is converted to servlet.

`<%@include file="url"%>`

Ex: `<html>`
`<body>`

Going to include hello.jsp. `<br>`

`<%@include file="hello.jsp"%>`

`</body>`

`</html>`

- c) taglib directive:-

It is used to extend JSP functionality by making use of custom actions. Taglib directive enables a programmer to make use of custom actions defined in tag library.

Syntax:

```
<%@ taglib (uri = "taglibURI" / tagdir = "tagdir" )  
           prefix = "tagPrefix" %>
```

Attributes:-

a) uri — This attribute uniquely associates tag library with named tag prefix. ^{used to} uri is uniquely identify library, it will not be directly accessed.

b) tagdir — specifies container to look for tag library in /WEB-INF/tags directory or under subdirectories

c) prefix — defines an identifier, which should be used to access the custom tags in tag library identified by uri or tagdir.

② Scripting elements:-

Jsp scripting elements let the programmer to insert java code into the Servlet that will be generated from current Jsp page. There are three types of scripting elements:

- (a) Expression (b) Scriptlets (c) Declaration

(a) Expression:-

A Jsp expression is used to insert values directly into the output. It has following form: `<%= Java Expression %>`

The expression is evaluated, converted to a string and inserted in the page. This evaluation is performed at runtime.

predetined Variables:-

To simplify expressions, you can use a number of predetined variables.

The most important ones are these:

request - the `HttpServletRequest`

response - the `HttpServletResponse`.

Session - HttpSession associated with request
out - the writer used to send output
to client.

application - Servlet Context. This is a data
structure shared by all Servlets
and Jsp pages.

Ex: `<html>`
 `<body>`
 Hello! The time is now `<%= new`
 `Java.util.Date()%>`
 `</body>`
`</html>`

b) Jsp scriptlets:

Jsp scriptlets let you insert
arbitrary code into Servlet method
that will be built to generate page.
Scriptlets have following form:

`<%= Java Code %>`


```

Ex: <html>
    <body>
        <% out.println ("Evaluating date now");
            Java.util.Date date = new
                Java.util.Date();
        %>
        Hello! The time is now.
        <% out.println (date);
            out.println("<br> Your machine
                address is");
            out.println (request.getRemoteHost());
        %>
    </body>
</html>

```

(c) Jsp Declarations:-

Jsp turns into a class. All the scriptlets you write are placed inside a single method of this class. You can also add variable and method declarations to this class. You can use these variables and methods from scriptlets & expressions.

A declaration has following form:

<%! ... %>

Ex:

<html>

<body>

<%! java.util.Date date = new
java.util.Date();

Date getDate()

{
s.o.pln("In getDate() method");
return date;
}

%>

Hello! The time is now <%= getDate() %>

</body>

</html>

③ Action Elements:-

specific functionality is encapsulated by action elements in predefined tags, to be used by programmers in a JSP page. At

translation time these action elements are replaced by Java code which corresponds to specific functionality.

a) Jsp: include action:-

`<jsp:include page="url" flush="true"/>`
Includes a file at the page is requested.

Ex: `<html>`

`<body>`

Here is a summary for include action

`<jsp:include page="abc.html" flush="true"/>`

`</body>`

`</html>`

b) Jsp: useBean action:-

This action lets you load in a JavaBean to be used in Jsp page.

`<jsp:useBean id="name" class="package.class"/>`

id — gives a name to variable that will reference the bean.

class — designates full package name of bean.

scope - Indicates context in which bean should be made available

(a) page - Indicates bean is only available on current page.

(b) request - Indicates bean is available only for current client request.

(c) session - Indicates that object is available to all pages during life of current HttpSession.

(d) application - Indicates that it is available to all pages that share same ServletContext.

type - Specifies type of variable that will refer to object.

beanName - gives name of bean as you would supply it to instantiate method of Beans.

Ex: beanTest.jsp

<html>

<body>


```
<jsp:useBean id="dest" class="web.beans.SimpleBean"/>
```

```
<jsp:setProperty name="dest" property="message"
value="Hello world" />
```

```
<H1> Message :
```

```
<I>
```

```
<jsp:getProperty name="dest" property="message" />
```

```
</I> </H1>
```

```
</body> </html>
```

SimpleBean.java

```
package web.beans;
```

```
public class SimpleBean {
```

```
    private String message = "No message specified";
```

```
    public String getMessage() {
```

```
        return message;
```

```
    }
```

```
    public void setMessage(String message)
```

```
    {
```

```
        this.message = message;
```

```
    }
```

```
}
```

c) Jsp:setProperty Action

This is used to give values to properties of beans that have been declared earlier. You can do this in two ways:

```
<jsp:useBean id="myName" />
```

```
<jsp:setProperty name="myName"  
property="someProperty" />
```

```
<jsp:useBean id="myName">
```

```
<jsp:setProperty name="myName"  
property="someProperty" />
```

```
</jsp:useBean>
```

Attributes:

name - required attribute designates the bean whose property will be set.
property - required attribute indicates the property you want to set. However, there is one special case: a value of '*' means that all required bean property names will be passed to appropriate setter methods.

value: It is an optional attribute specifies the value of property.

param - Optional attribute designates the request parameter from which property should be derived. Value & param ~~tag~~ can't be used together.

d) Jsp: getProperty Action:-

Retrieves value of a bean property, converts it to a string and inserts it into output. The two required attributes are name and property.

```
<jsp:getProperty name=".." property=".." />
```

Ex: HelloWorld.jsp

```
<%@page import="web.beans.NameHandler"%>
<jsp:useBean id="myBean" scope="page"
    class="web.beans.NameHandler"/>
<jsp:setProperty name="myBean" property="*" />
<html>
  <body>
```

<h1> My name is aaa, what ur's </h1>

<form method= get>

<input type= "text" name= "username">

<input type= "submit">

<input type= "reset">

</form>

<%

if (request.getParameter("username")
!= null)

{

%%>

<% @ include file= "response.jsp" %>

<%

}

%>

</body>

</html>

response.jsp

```
<html>
<h1> Hello <jsp:getProperty name="my Bean"
property="username"/>

</h1>
</html>
```

namehandler.java:

```
package web.beans;
public class namehandler
{
    private String username;
    public namehandler()
    {
        username = null;
    }
    public void setUsername(String name)
    {
        username = name;
    }
    public String getUsername()
    {
        return username;
    }
}
```

compilation : javac -d. namehandler.java.
for pkg

e) Jsp: forward Action:-

This action lets you forward the request to another page. It has single attribute page.

`<jsp:forward page="relativeUrl"/>`

④ ~~Expression Language Statement~~:-

↳ Comments in Jsp:-

i) HTML comment

`<!-- comment included -->`

ii) Jsp comment

`<% -- comment included -- %>`

- Database Programming using
- studying javax.sql.* package
- Accessing a database from a Jsp page.
- Application-specific Database Action
- Deploying JAVA Beans in a Jsp Page.
- Introduction to struts framework.

DSN Connection (dataSourceName)

- ① goto control panel → Administrative tools
↳ Data Sources (ODBC)
- ② Add ← select any driver based on database used
- ③ specify DataSourceName in the textfield specified and
select/create/repair the data by
choosing the path where the ~~exe~~
database is present.

Database are tools used to collect and store large volumes of data. Data is collected & stored in form of tables consisting of rows & columns.

→ JAVA DATABASE CONNECTIVITY:- (JDBC)

JDBC is also abstraction, that interfaces your Java appln with different kinds of databases. It sits b/w your application & database you want to access.

JDBC interface is a pure JAVA API used to execute SQL statements. JDBC provides a set of classes and interfaces that can be used to develop or to write database applns. Basic JDBC interaction in its simplest form, can be broken down into four steps:

1) Load JDBC driver/get a DataSource Object using JNDI.

class.forName ("name of driver");

2) Create a database connection
Connection conn = DriverManager.getConnection (url, uid, pwd);

(3) Create or Execute a SQL statement.

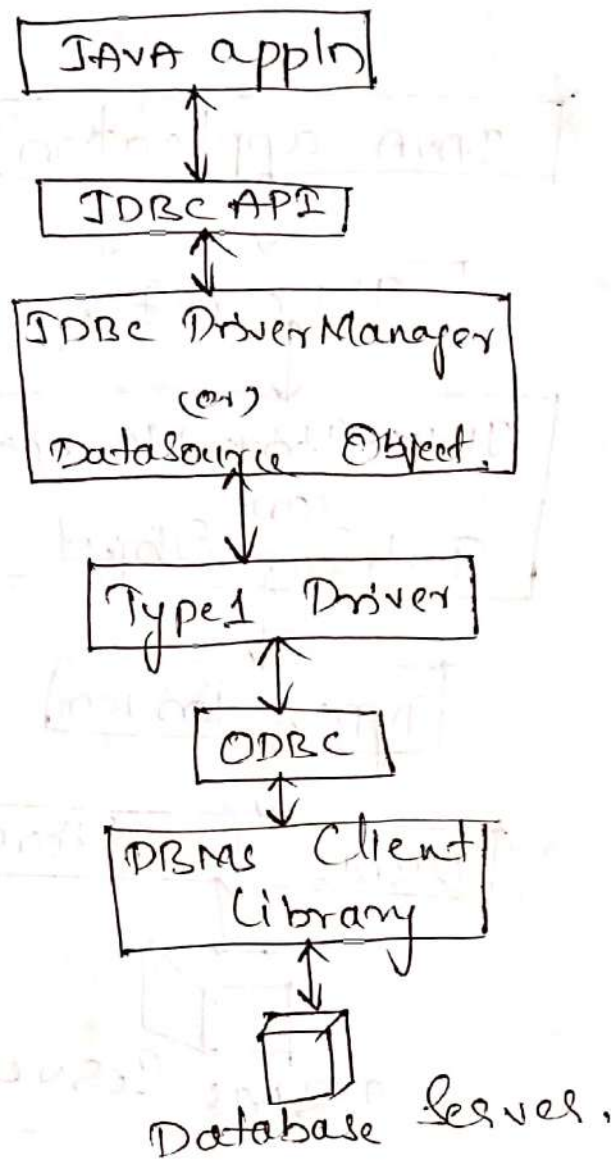
(4) Process the result.

(5) Close connection to database.

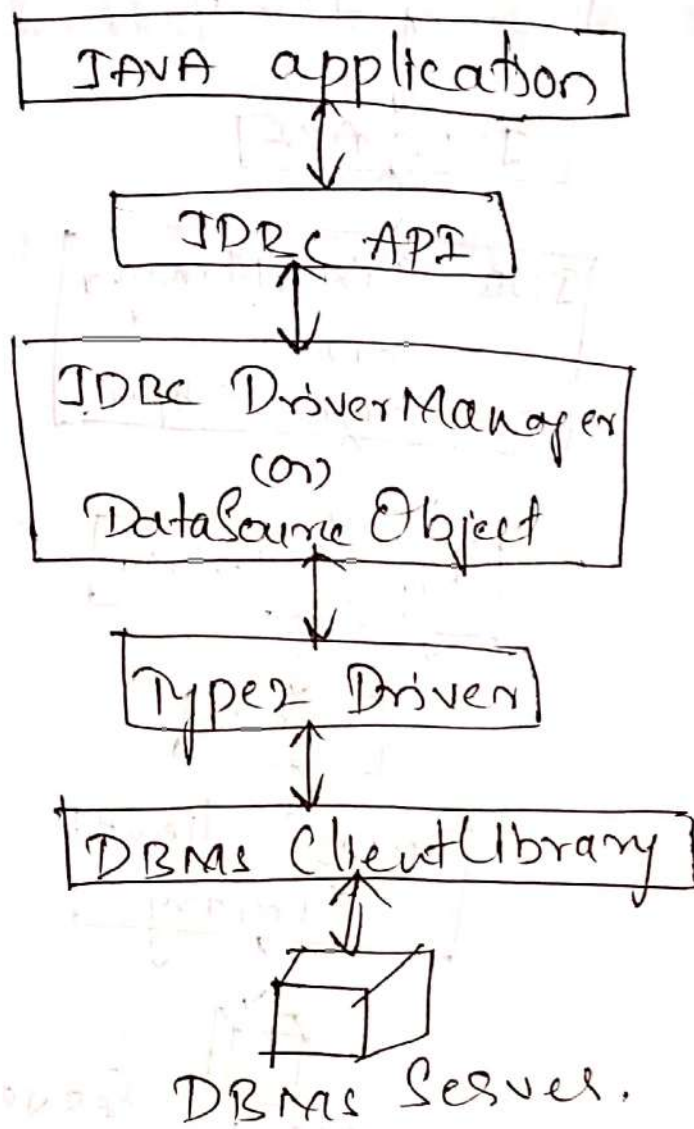
→ JDBC DRIVERS TYPE:-

① Type 1 (JDBC-ODBC Bridge) Driver :-

This allows an app to access database through an intermediate ODBC driver. Here ODBC acts as mediating layer b/w JDBC driver and vendor's client libraries.



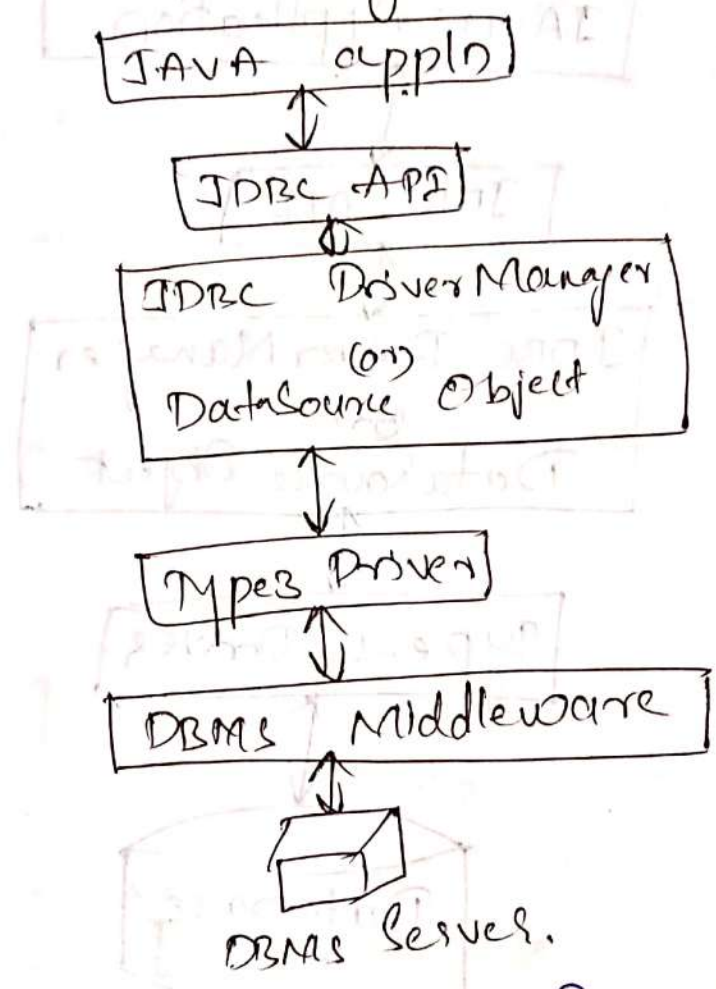
- ② Type2 (Partial JDBC / Native API) Driver :-
 Type2 driver converts JDBC calls into client API for DBMS. The driver communicates directly with database server therefore requires that some database client also be loaded on each client machine again limiting its usefulness for internet.



Type 3 (Pure Java JDBC driver for database middleware) Driver

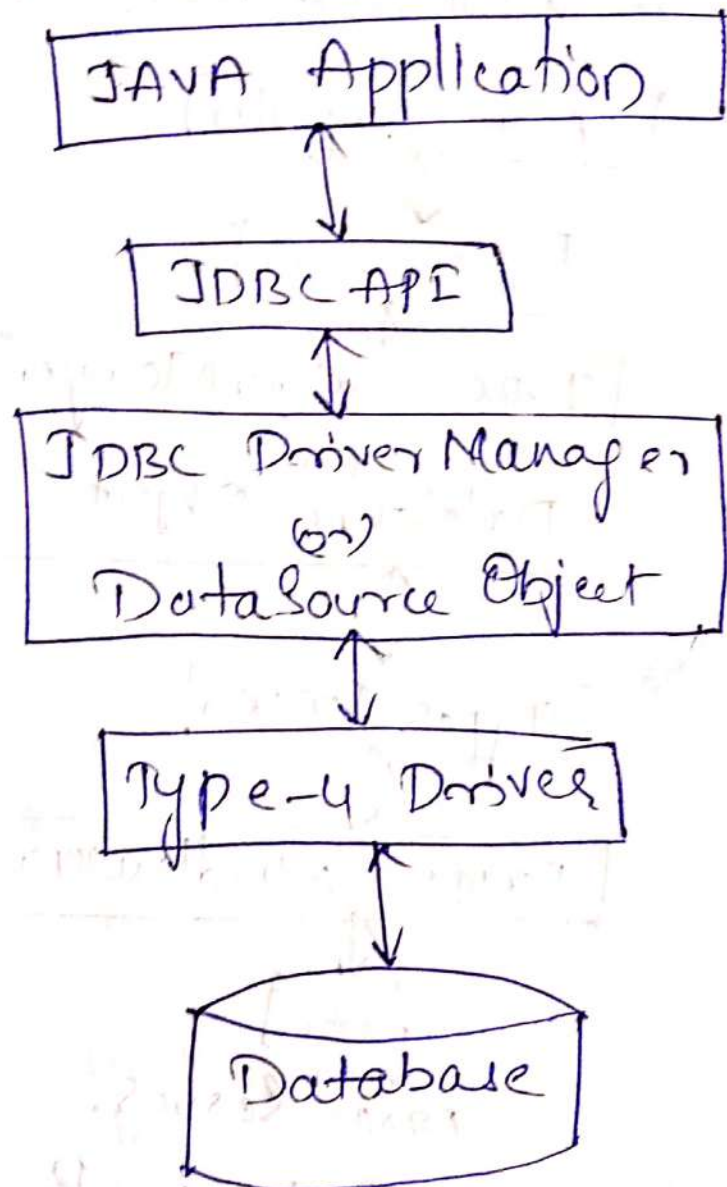
Type 3 driver is completely implemented in Java, hence it is a pure Java JDBC driver. It translates JDBC calls into middleware vendors protocol

which is subsequently translated to a DBMS protocol by a middleware server. These types of drivers are best suited for environments that need to provide connectivity to a variety of DBMS servers & heterogeneous databases.



- ④ Type-4 (Native-Protocol Pure Java Driver):-
 The type-4 drivers are pure Java Drivers that communicate directly with vendor's database. They do this by directly converting

JDBC commands into database engine's native protocol. It has no additional translator or middleware layers, which improves performance tremendously.



→ Steps to Connect to Database in Java:

Steps:

① Register driver class

`forName()` method of `Class` class is used to register driver class. This method is used to dynamically load driver class.

p.s. `forName(String classname)` throws `ClassNotFoundException`.

② Ex: `Class.forName("Oracle.jdbc.driver.
OracleDriver");`
(for oracle driver)

② Creating Connection:

The `getConnection()` method of `DriverManager` class is used to establish connection with database.

Syntax:

- 1) `ps Connection getConnection(String url)`
throws `SQLException`
- 2) `ps Connection getConnection(String url, String name, String pwd)` throws `SQLException`.

Ex:

```
Connection con = DriverManager.getConnection
```

```
("jdbc:oracle:thin:@localhost:  
1521:xe", "Dname", "pwd");
```

③ Create the Statement Object.

createStatement() method of connection Interface is used to create statement.

P. Statement createStatement();

Ex:

```
Statement st = con.createStatement();
```

④ Execute the query:-

executeQuery() method of Statement Interface is used to execute queries to database.

```
public ResultSet executeQuery(String sql)  
throws SQLException.
```

Ex:

```
ResultSet rs = st.executeQuery("select *  
from emp");
```

```
while (rs.next()) {
```

```
    s.o.pln(rs.getInt(1) + " " + rs.getString(2))
```

⑤ Close connection object.

By closing connection object statement and ResultSet will be closed automatically.

✶ close() throws SQLException

Ex: con.close();

→ Java.sql package:-

Interfaces:-

① Connection

② Statement

③ PreparedStatement

④ CallableStatement

⑤ Driver

⑥ ResultSet

① Connection Interface :-

This interface is used to establish a connection to database you want to access. ~~the~~ Methods are:

Interface Connection {

public void close();

public Statement createStatement();

public CallableStatement

prepareCall (String s);

public PreparedStatement

prepareStatement (String sql);

public void commit();

" rollback();

}

② Statement Interface:-

It is used to send SQL queries to database & retrieve a set of data. SQL Statement can be queries, updates, insertion, deletions etc.

Interface Statement {

public void close();

public boolean execute(String sql);

public abstract int executeUpdate(String sql);

// - will execute an SQL stmt and
// return no. of rows

public ResultSet executeQuery(String sql);

}

③ PreparedStatement Interface:-

It extends Statement Interface.

This interface is used to pass parameters into a SQL statement. Since it is extending stmt interface, all methods

available in that interface are available in this interface also.

(ii) CallableStatement interface:-

This interface extends PreparedStatement. This interface can be used to execute stored SQL Procedures in a database.

(i) Driver Interface:-

This provides methods for establishing connection with specific databases. Such methods use a URL type string as parameter to locate and access database. The URL string should have following characteristics:

- a) it should specify type of database being accessed.
- b) it should contain database access info. like hostname, database name, portno, username, password etc.

JDBC URLs have following syntax:

Jdbc: <subprotocol> : <subname>

<subprotocol> → type of database driver

<subname> → provides also encoded name of database being accessed.

⑥ ResultSet interface :-

When SQL stmt is executed, one or more tables of data may be generated. The methods in ResultSet interface can be used to retrieve these data.

```
interface ResultSet {
```

```
    public boolean next();
```

```
    public String getString(int index);  
    public String getString(String colname);
```


Classes

① DriverManager

② Date

③ DriverPropertyInfo.

④ Types.

① DriverManager Interface Class:-

DriverManager class controls the loading of driver specific classes. Drivers are implemented as a set of class files.

Class DriverManager {

- public static Connection

getConnection (String url)

throws SQLException;

- public static Connection

getConnection (String url,
String uname, String pwd) throws
SQLException;

8-1
- public static void registerDriver(
Driver driver).

② Date class:-

This class is used for displaying date.

③ Driver Property Info Class:-
Gives the properties of the registered driver which we are using.

④ Types class:-
This class defines a set of integer constants that identify SQL datatypes. These constants are then used to map SQL datatypes into corresponding Java datatypes.

→ Ans Creation of table using
Statement interface with mdb
database:

```
import java.io.*;
import java.sql.*;
public class insertcreatedb {
    {
        psvm (String a[]) throws SQLException, IOException
        {
            try {
                int i;
                Class.forName("sun.jdbc.odbc.
                                JdbcOdbcDriver");
                Connection con = DriverManager
                    .getConnection("jdbc:
                                    odbc: first");
                S.o.p l ("connection is
                            successful");
                Statement st = con.create
                    Statement();
```

```
String table = "Create table Employee11  
(Emp_code integer, Emp_name  
varchar(10));
```

```
st.executeUpdate(table);
```

```
s.o.pln("Table created");
```

```
st.close();
```

```
con.close();
```

```
}
```

```
catch (ClassNotFoundException e) {
```

```
s.o.pln(e);
```

```
}
```

```
}
```

```
}
```

→ Program for inserting details using
Statement Interface in mdb:-

```
import java.io.*;
```

```
import java.sql.*;
```

```
public class Insertdb {
```

```
    public static void main (String a[]) throws
```

```
{
```

```
        SQLException, IOException
```



```

try {
    int i;
    Class.forName("sun.jdbc.odbc.
        JdbcOdbcDriver");
    Connection con =
        DriverManager.getConnection("
            jdbc:odbc:Test");
    S.o.pln("connection is
        successful");
    String select s = "insert into
        Test (ID, NAME, ADDRESS)
            Values (1, 'aaa', 'aaa')";
    Statement st = con.createStatement();
    st st.executeUpdate(s);
    S.o.pln("inserted");
    st.close();
    con.close();
}
catch (ClassNotFoundException e)
{
    S.o.pln(e);
}
}
}

```

→ Program for deletion of records from database (mdb) using Statement Interface

```
import java.io.*;
```

```
import java.sql.*;
```

```
public class deletedb {
```

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

```
    {
```

```
        try {
```

```
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

```
            Connection conn = DriverManager.getConnection(
                "jdbc:odbc:First");
```

```
            System.out.println("Connection Successful");
```

```
            String s = "delete from First
                where NAME='i i'";
```

```
            Statement st = conn.createStatement();
```

```
            st.executeUpdate(s);
```

```
            System.out.println("deleted Successfully");
```

```
            st.close();
```

```
            conn.close();
```

```
        } catch (ClassNotFoundException e) {
            System.out.println(e); } } }
```


→ Program for selecting records from database (mdb) using Statement Interface.

selectdb

8-11

```
///Print to be taken//
import java.io.*;
import java.sql.*;

public class selectdb
{
    public static void main(String[] args) throws SQLException, IOException
    {
        try
        {
            int i;
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection conn =
            DriverManager.getConnection("jdbc:odbc:first");
            System.out.println("connection is successful!!!");
            String selectSQL = "select ID, NAME, ADDRESS from first";
            Statement stmt = conn.createStatement();
            ResultSet rset = stmt.executeQuery(selectSQL);
            while(rset.next())
            {
                System.out.println("ID: " +
                rset.getString(1) + " NAME: " + rset.getString(2) + " ADDRESS: " +
                rset.getString(3));
            }
            stmt.close();
            rset.close();
            conn.close();
        }
        catch(ClassNotFoundException e) {
            System.out.println(e);
        }
    }
}
```

→ Program for updating records from database (mdb) using Statement Interface.

updatedb

8-12

```
//print to be taken//
import java.io.*;
import java.sql.*;

public class updatedb
{
    public static void main(String[] args) throws SQLException, IOException
    {
        try
        {
            int i;
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection conn =
DriverManager.getConnection("jdbc:odbc:first");
            System.out.println("connection is successful!!!");
            String selectSQL = "update first set NAME='iii' where
NAME='aaa'";
            Statement stmt = conn.createStatement();
            i= stmt.executeUpdate(selectSQL);

            System.out.println("updated succesfully");
            stmt.close();
            conn.close();
        }
        catch(ClassNotFoundException e) {
            System.out.println(e);
        }
    }
}
```


→ Pgm for inserting details of a user to database using servlets

registerdb.html

8-13

```
<html>
<head> Register
<title> login </title>
</head>
<body bgcolor="pink">
<form name=form1 action="./regdb">
<br><br><br><br><br><br>
<table border=0 align=center>
<tr><td colspan=2><center>User Register</center></td></tr>
<tr><td>UserName</td><td><input type=text name="name"></td></tr>
<tr><td>Password</td><td><input type=password name="pass"></td></tr>
<tr><td>CPassword</td><td><input type=password name="cpass"></td></tr>
<tr><td>Address</td><td><TextArea name="addr"></td></tr>
<tr><td colspan=2><input type=submit value=login><input type=reset></td></tr>
</table>
</form>
</body></html>
```

web.xml

<Web-app>

<Servlet>

<servlet-name> registerdb </servlet-name>

<servlet-class> " " </servlet-class>

</Servlet>

<servlet-mapping>

<servlet-name> registerdb </servlet-name>

<url-pattern> /registerdb </url-pattern>

</servlet-mapping>

</web-app>

```

import java.sql.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class regdb extends HttpServlet
{
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOException
    {
        PrintWriter pw=res.getWriter();
        res.setContentType("text/html");
        String a,b,c,d;
        Connection con;
        PreparedStatement ps;
        ResultSet rs;
        try
        {
            pw.println("in try");
            a=req.getParameter("name");
            b=req.getParameter("pass");
            c=req.getParameter("cpass");
            d=req.getParameter("addr");
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            con=DriverManager.getConnection("Jdbc:Odbc:servletdb");
            ps=con.prepareStatement("insert into
reg(uname,pass,cpass,addr) values(?,?,?,?)");
            ps.setString(1,a);
            ps.setString(2,b);
            ps.setString(3,c);
            ps.setString(4,d);
            ps.executeUpdate();
            pw.println("inserted");
            //res.sendRedirect("C:/Program Files/Apache Software
Foundation/Tomcat 5.5/webapps/priyanka/login.html");
            //RequestDispatcher rd
            =getServletContext().getRequestDispatcher("C:/Program Files/Apache Software
Foundation/Tomcat 5.5/webapps/priyanka/login.html");
            //rd.forward(request,response);
        }
        catch (SQLException se)
        {
        }
        catch(ClassNotFoundException ce){}
    }
}

```


→ Program for login using database.

logindb.html

```
<html>
<head>
<title>login</title>
</head>
<body bgcolor="pink">
<form name=form1 action="./logindb">
<br><br><br><br><br><br>
<table border=0 align=center>
<tr><td colspan=2><center>User login</center></td></tr>
<tr><td>UserName<td><input type=text name="name"></td></tr>
<tr><td>Password<td><input type=password name="pass"></td></tr>
<tr><td colspan=2><input type=submit value=login><input type=reset></td></tr>
</table>
</form>
</body></html>
```

web.xml

```
<web-app>
  <servlet>
    <servlet-name>logindb </
    <servlet-class>    "    </
  </
  <servlet-mapping>
    <servlet-name>logindb </
    <url-pattern>    "    </
  </
</web-app>
```

8-16

logindb.java

```

///Print to be taken//
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
public class logindb extends HttpServlet
{
    public void doGet(HttpServletRequest req,HttpServletResponse res) throws
IOException,ServletException
    {
        PreparedStatement ps;
        Connection conn;
        ResultSet rset;
        res.setContentType("text/html");
        PrintWriter pw=res.getWriter();
        String u=req.getParameter("uname");
        String p=req.getParameter("pass");
        pw.println(u);
        try
        {
            pw.println("in try");
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            pw.println("sun.jdbc.odbc.JdbcOdbcDriver");
            conn = DriverManager.getConnection("jdbc:odbc:MS Access
Database");
            pw.println("connection is successful!!!");
            ps=conn.prepareStatement("select * from login where uname=?
and pass=?");
            ps.setString(1,u);
            ps.setString(2,p);
            rset = ps.executeQuery();
            if(rset.next())
            {
                pw.println("welcome"+u);
            }
            else
            {
                pw.println("incorrect uname/password");
            }
        }
        catch(ClassNotFoundException e) {
            system.out.println(e);
        }
        catch(SQLException se) {
            system.out.println(se);
        }
    }
}

```