What is XML?

- XML stands for e**X**tensible **M**arkup **L**anguage.
- XML is designed to transport and store data.
- HTML was designed to display data.
- XML is a markup language much like HTML
- XML was designed to carry data, not to display data
- XML tags are not predefined. You must define your own tags
- XML is designed to be self-descriptive
- XML is Not a Replacement for HTML
- XML is the most common tool for data transmissions between all sorts of applications.
- XML is a W3C Recommendation

The Difference Between XML and HTML

XML and HTML were designed with different goals:

- XML was designed to transport and store data, with focus on what data is
- XML was created to structure, store, and transport information.
- HTML was designed to display data, with focus on how data looks

Role of XML

XML Simplifies Data Sharing

In the real world, computer systems and databases contain data in incompatible formats.

XML data is stored in plain text format. This provides a software- and hardware-independent way of storing data.

This makes it much easier to create data that can be shared by different applications.

XML Simplifies Data Transport

One of the most time-consuming challenges for developers is to exchange data between incompatible systems over the Internet.

XML Separates Data from HTML

If you need to display dynamic data in your HTML document, it will take a lot of work to edit the HTML each time the data changes.

With XML, data can be stored in separate XML files. This way you can concentrate on using HTML/CSS for display and layout, and be sure that changes in the underlying data will not require any changes to the HTML

XML DOCUMENT:

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
    <to> Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

XML document contains following parts:

- **Prolog**
- **Body**

Prolog:

- Xml declaration
- Optional processing instrucion
- Comments
- Document type declartion

Xml Declaration

<?xml version="1.0"? >

- Specifies its an xml document
- Version is 1.0(mandatory attribute)
- Encoding and standalone (optional attributes).

<?xml version="1.0" encoding="utf-8" standalone="no"? >

- Utf-8:unicode transformation format which is the same character set as ASCII
- Standalone specifies is the XML document depend on other document or not(depedent:yes, not depedent:no)

Processing instruction

Processing instruction starts with <? And ends with ?>.They allow document to contain special instructions that are used to pass parameters to applications

Comments

<!-- comment text -- >

- Donot use - with in comments
- Never place a comment with in tag
- Never place a comment before xml declarion

BODY:

```
<?xml version="1.0" encoding="UTF-8"?>
< note>
        < to>Tove</to>
```

< from>Jani</from>
< heading>Reminder</heading>
< body>Don't forget me this weekend!</body>
< /note>

- Only one root element in the document

<?xml version="1.0" encoding="UTF-8"?>
< to>Tove</to>
< from>Jani</from>

(not correct xml having two root elements)

XML Element

<tagename attribute="value"> Content </tagname>

Naming Rules:

Rules should follow while selecting element name

- Names can only contains letters,digits and some special characers.
- Name canot start with a number or puncuation mark
- Names must not contain the string "xml" "XML" "Xml"
- Names canot contain white spaces

Empty element

<line></line> or <line/>

WELL FORMED XML

- Document must have exactly one root element
- All tags must be closed

<name>Ram (not correct)

<name> Ram </name> (correct)

- All tags must be nested properly

Incorrect nesting

Correct nesting

- Xml tags are case sensitive

<Message>This is correct</Message>

<Message>This is incorrect</message>

<MESSAGE>This is incorrect</message >

- Attributes must be Quoted

<speed unit="rpm">65777</speed> (correct)

<speed unit=rpm>65777</speed> (not correct)

- Certain characters are reserved for processing

<condition>if salary <1000 then</condition> (not correct)

<condition>if salary < 1000 then</condition>

Predefined Entities:

Entity name	Character	Description
<	<	less than
>	>	greater than
&	&	ampersand
'	'	apostrophe
"	"	quotation mark

Name space

Xml was developed to be used by many applications. If many applications want to communicate using xml documents, a potential problem may occur.

Client1.xml

<table>

<item1> Apple </item1>

```
    <item2>  Ice     </item2>
</table>
```

Client2. xml

```
<table>
     <item3> Tea </item3>
    <item4>   Milk   </item4>
</table>
```

Merge. xml

```
<content >
    <table>
       <item1> Apple </item1>
       <item2>  Ice     </item2>
    </table>
   <table>
      <item3> Tea </item3>
      <item4>   Milk   </item4>
  </table>
</content >
```

Improve Merge. xml

```
<technology>
 <client1>
     <table>
       <item1> Apple </item1>
       <item2>  Ice     </item2>
    </table>
```

```
</client1>
<client2 >
   <table>
      <item3> Tea </item3>
      <item4>  Milk  </item4>
   </table>
</client2 >
</technology>
```

Using Prefix:

Syntax: <prefix:tagname> tag content </prefix:tagname>

```
<content >
     <c1:table>
        < c1:item1> Apple </ c1:item1>
        < c1:item2> Ice    </ c1:item2>
     </ c1:table>
     <c2:table>
        < c2:item3> Tea </ c2:item3>
        < c2:item4>  Milk  </ c2:item4>
     </ c2:table>
<content>
```

Namespace Binding:

Syntax:

Xmlns:prefix="URL"

```
<c:table xmlns:c="http://client.html/client1">
        < c:item1> Apple </ c:item1>
        < c:item2> Ice    </ c:item2>
     </ c:table>
```

```
<s:table xmlns:s="http://server.html/client2">
    < s:item3> Tea </ s:item3>
    < s:item4>  Milk  </ s:item4>
</ s:table>
```

Namespace rules:

Rule1:

```
<content xmlns:c="http:google.com/content" xmlns:k="http:google.com/content" >
    <c:item1 >computer</c:tem1>
    <k:item2 >keyboard </k:tem2>
</content>
```

Rule2:

```
<content>
    <c:table xmlns:c="http://client.html/client">
        < c:item1> Apple </ c:item1>
        < c:item2> Ice  </ c:item2>
    </ c:table>
    <c:table xmlns:c="http://server.html/server">
        < c:item3> Tea </ c:item3>
        < c:item4>  Milk  </ c:item4>
```

</ c:table>

</content>

Default Namespace:

<content xmlns="http://w3schools.html">

< item1 >computer</ tem1>

< item2 >keyboard </ tem2>

</content>

XML DTD:

- A Document Type Definition (DTD) defines the legal building blocks of an XML document.
- A DTD defines the document structure with a list of legal elements and attributes.
- A DTD can be declared inside an XML document, or as an external reference.
- DTD is two types
  internal DTD
  External DTD

External DTD Declaration

If the DTD is declared in an external file, it should be wrapped in a DOCTYPE definition with the following syntax:

<!DOCTYPE root-element SYSTEM "filename">

Xml document

```
<?xml version="1.0" standalone="yes"?>
< !DOCTYPE note SYSTEM "note.dtd">
<note>
    < to>Tove</to>
    < from>Jani</from>
    < heading>Reminder</heading>
    < body>Don't forget me this weekend!</body>
< /note>
```

And this is the file "note.dtd" which contains the DTD:

<!ELEMENT note (to,from,heading,body)>
< !ELEMENT to (#PCDATA)>
< !ELEMENT from (#PCDATA)>
< !ELEMENT heading (#PCDATA)>
< !ELEMENT body (#PCDATA)>

Why Use a DTD?

With a DTD, independent groups of people can agree to use a standard DTD for interchanging data.

The Building Blocks of XML Documents

Seen from a DTD point of view, all XML documents (and HTML documents) are made up by the following building blocks:

- Elements
- Attributes
- Entities
- PCDATA
- CDATA

PCDATA

PCDATA means parsed character data.

Think of character data as the text found between the start tag and the end tag of an XML element.

PCDATA is text that WILL be parsed by a parser. The text will be examined by the parser for entities and markup.

Tags inside the text will be treated as markup and entities will be expanded.

However, parsed character data should not contain any &, <, or > characters; these need to be represented by the & < and > entities, respectively.

CDATA

CDATA means character data.

CDATA is text that will NOT be parsed by a parser. Tags inside the text will NOT be treated as markup and entities will not be expanded.

Declaring Elements

In a DTD, XML elements are declared with an element declaration with the following syntax:

<!ELEMENT element-name (element-content)>

<!ELEMENT element-name (#PCDATA)>

Example:

< !ELEMENT from (#PCDATA)>

<!ELEMENT element-name (#CDATA)>

Example:

< !ELEMENT from (#CDATA)>

Elements declared with the category keyword ANY, can contain any combination of parsable data:

<!ELEMENT element-name ANY>

Example:

< !ELEMENT note ANY>

Elements with Children (sequences)

Elements with one or more children are declared with the name of the children elements inside parentheses:

<!ELEMENT element-name (child1)>
or
< !ELEMENT element-name (child1,child2,...)>

Example:

< !ELEMENT note (to,from,heading,body)>

When children are declared in a sequence separated by commas, the children must appear in the same sequence in the document. In a full declaration, the children must also be declared, and the children can also have children. The full declaration of the "note" element is:

<!ELEMENT note (to,from,heading,body)>
< !ELEMENT to (#PCDATA)>
< !ELEMENT from (#PCDATA)>
< !ELEMENT heading (#PCDATA)>
< !ELEMENT body (#PCDATA)>

Syntax	Meaning	Explanation
<!ELEMENT element-name (child-name)> Example: < !ELEMENT note (message)>	Declaring Only One Occurrence of an Element	The example above declares that the child element "message" must occur once, and only once inside the "note" element.
<!ELEMENT element-name (child-name+)> Example: < !ELEMENT note (message+)>	Declaring Minimum One Occurrence of an Element	The + sign in the example above declares that the child element "message" must occur one or more times inside the "note" element.
<!ELEMENT element-name (child-name*)> Example: < !ELEMENT note (message*)>	Declaring Zero or More Occurrences of an Element	The * sign in the example above declares that the child element "message" can occur zero or more times inside the "note" element.
<!ELEMENT element-name (child-name?)> Example: < !ELEMENT note (message?)>	Declaring Zero or One Occurrences of an Element	The ? sign in the example above declares that the child element "message" can occur zero or one time inside the "note" element.
<!ELEMENT note (to,from,header,(message|body))>	Declaring either/or Content	The example above declares that the "note" element must contain a

		"to" element, a "from" element, a "header" element, and either a "message" or a "body" element.

Declaring Attributes:

An attribute declaration has the following syntax:

<!ATTLIST element-name attribute-name attribute-type attribute-value>

DTD example:

< !ATTLIST payment mode CDATA "check">

XML example:

< payment mode="check" />

The **attribute-type** can be one of the following:

Type	Description
CDATA	The value is character data
(*en1*|*en2*|..)	The value must be one from an enumerated list
ID	The value is a unique id
IDREF	The value is the id of another element
IDREFS	The value is a list of other ids
NMTOKEN	The value is a valid XML name
NMTOKENS	The value is a list of valid XML names

ENTITY	The value is an entity
ENTITIES	The value is a list of entities
NOTATION	The value is a name of a notation
xml:	The value is a predefined xml value

The **attribute-value** can be one of the following:

Value	Explanation
Value	The default value of the attribute
#REQUIRED	The attribute is required
#IMPLIED	The attribute is optional
#FIXED *value*	The attribute value is fixed

#REQUIRED

Syntax

<!ATTLIST element-name attribute-name attribute-type #REQUIRED>

Example

DTD:

< !ATTLIST person number CDATA #REQUIRED>

Valid XML:

< person number="5677" />

Invalid XML:

< person />

#IMPLIED

Syntax

<!ATTLIST element-name attribute-name attribute-type #IMPLIED>

Example

DTD:

< !ATTLIST contact fax CDATA #IMPLIED>

Valid XML:

< contact fax="555-667788" />

Valid XML:

< contact />

#FIXED

Syntax
<!ATTLIST element-name attribute-name attribute-type #FIXED "value">

Example
DTD:
< !ATTLIST sender company CDATA #FIXED "Microsoft">

Valid XML:
< sender company="Microsoft" />

Invalid XML:
< sender company="W3Schools" />

Xml document

```
<?xml version="1.0" standalone="yes"?>
< !DOCTYPE note SYSTEM "note.dtd">
<note>
    < to  verfied="yes">Tove</to>
    < from>Jani</from>
    < heading>Reminder</heading>
    < body  attach="yes">Don't forget me this weekend!</body>
< /note>
```

And this is the file "note.dtd" which contains the DTD:

```
<!ELEMENT note (to,from,heading,body)>
< !ELEMENT to (#PCDATA)>
< !ELEMENT from (#PCDATA)>
< !ELEMENT heading (#PCDATA)>
< !ELEMENT body (#PCDATA)>
<! ATTLIST  to  verfied  CDATA  "yes">
<! ATTLIST  body  attch  CDATA  "yes">
```

Entities are variables used to define shortcuts to standard text or special characters.

- Entity references are references to entities

Syntax
<!ENTITY entity-name "entity-value">

Example
DTD Example:

< !ENTITY writer "Donald Duck.">
< !ENTITY copyright "Copyright.">

XML example:

< author>&writer;&copyright;</author>

Note: An entity has three parts: an ampersand (&), an entity name, and a semicolon (;).

Internal DTD

```
<?xml version="1.0"?>
< !DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
< note>
    <to>Tove</to>
     <from>Jani</from>
     <heading>Reminder</heading>
    <body>Don't forget me this weekend</body>
< /note>
```

XML SCHEMA:

- An XML Schema describes the structure of an XML document.
- XML Schema is an XML-based alternative to DTD.
- The XML Schema language is also referred to as XML Schema Definition (XSD).

What is an XML Schema?:

The purpose of an XML Schema is to define the legal building blocks of an XML document, just like a DTD.

An XML Schema:

- defines elements that can appear in a document
- defines attributes that can appear in a document
- defines which elements are child elements
- defines the order of child elements

- defines the number of child elements
- defines whether an element is empty or can include text
- defines data types for elements and attributes
- defines default and fixed values for elements and attributes

Limtations of DTD:

- There is no built in datatypes in DTD
- No Namespaces are supported
- Defaults for elements canot be specified
- DTD are written in a strange (non-XML)format and are difficult to validate.

Strengths of XML Schema:

- XML Schemas provide much greater specificity than DTD
- XML Schema support large number of built in datatypes
- Support for Namespacs
- They are extensible to future additions
- Easy to validate correctness of data
- Syntax is much like XML easy to learn

Schema Structure:

- The **schema** element
- Element Definitions
- Attribute Definitions
- Annotations
- Type Definitions

schema element:

The <schema> element is the root element of every XML Schema.

```
<?xml version="1.0"?>

< xs:schema xmlns:xs="www.w3schools.com">
...
```

...

< /xs:schema>

Example of XML SCHEMA:

note.xsd

```
<?xml version="1.0"?>
< xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="note">
        <xs:complexType>
          <xs:sequence>
              <xs:element name="to" type="xs:string"/>
             <xs:element name="from" type="xs:string"/>
             <xs:element name="heading" type="xs:string"/>
            <xs:element name="body" type="xs:string"/>
          </xs:sequence>
   </xs:complexType>
< /xs:element>

< /xs:schema>
```

XML file:

```
<?xml version="1.0"?>

<note xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="note.xsd">

     <to>Rameesh</to>

     <from>GIRI</from>

    <heading>Greetings</heading>

    <body>How are you</body>

</note>
```

Element Declaration In Xml Schema(simple elements):

<xs:element name=”element-name” type=”data-type”>

Schema:

```
<xs:element name="to" type="xs:string"/>
<xs:element name="from" type="xs:string"/>
<xs:element name="heading" type="xs:string"/>
<xs:element name="body" type="xs:string"/>
```

Corresponding Xml

```
<to>Rameesh</to>
<from>GIRI</from>
<heading>Greetings</heading>
<body>How are you</body>
```

XML Schema has a lot of built-in data types. The most common types are:

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

Element Declaration In Xml Schema(complex elements):

```
<xs:complexType>
        <xs:sequence>
          .....

        </xs:sequence>
  </xs:complexType>

<xs:element name="note">
        <xs:complexType>
         <xs:sequence>
            <xs:element name="to" type="xs:string"/>
           <xs:element name="from" type="xs:string"/>
           <xs:element name="heading" type="xs:string"/>
```

```
            <xs:element name="body" type="xs:string"/>
          </xs:sequence>
    </xs:complexType>
< /xs:element>
```

Default & Fixed indicators:

```
<xs:element name="to" type="xs:string"  default=”ramesh”/>
```

Valid XML:

```
<to>ramesh</to>
```

```
<xs:element name="to" type="xs:string"  fixed=”ramesh”/>
```

Valid XML:

```
<to>ramesh</to>
```

Not Valid XML:

```
<to>Hari</to>
```

Occurence Indicators:

```
<xs:element name="body" type="xs:string"   minOccurs=”0” maxOccurs=”6”/>
```

note.xsd

```
<?xml version="1.0"?>
< xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="note">
        <xs:complexType>
          <xs:sequence>
             <xs:element name="to" type="xs:string" default=”ramesh”/>
            <xs:element name="from" type="xs:string"  fixed=”hari”/>
            <xs:element name="heading" type="xs:string"/>
           <xs:element name="body" type="xs:string" minOccurs=”0” maxOccurs=”2” />
         </xs:sequence>
   </xs:complexType>
< /xs:element>
```

< /xs:schema>

XML file:

<?xml version="1.0"?>

<note xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="note.xsd">

<to>ramesh</to>

<from>hari</from>

<heading>Greetings</heading>

<body>How are you</body>

<body>This is very good </body>

</note>

Defining Attributes:

<xs:attribute name="attribute-name" type="attribute-type"/>

<xs:attribute name="id" type="xs:string"/>

XML SCHEMA:Emp.xsd

<?xml version="1.0"?>

<xs:schema xmlns:xs="http://google.com">

<xs:element name="employee" >

<xs:complexType>

<xs:sequence>

<xs:element name="firstname" type="xs:string"/>

<xs:element name="lastname" type="xs:string"/>

</xs:sequence>

<xs:attribute name="id" type="xs:string"/>

```
        </xs:complexType>
    </xs:element>
</xs:schema>
```

XML FILE

```
<?xml version="1.0"?>
<employ xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Emp.xsd" id="emp1">
    <firstname> hari</firstname>
    <lastname> ram</lastname>
</employ>
```

Attribute element Properties:

DTD Syntax: <!ATTLIST element-name attribute-name attribute-type #REQUIRED>

```
<xs:attribute name="id" type="xs:string" use="required"/>
```

Valid xml:

```
<employ id="emp01"/>
```

Not Valid xml:

```
<employ />
```

```
<xs:attribute name="id" type="xs:string" use="optional"/>
```

Valid xml:

```
<employ id="emp01"/>
```

Valid xml:

```
<employ />
```

<xs:attribute name="married" type="xs:boolean" default="false"/>

Ex: <employ married="fasle">

<xs:attribute name="id" type="xs:string" fixed="emp1"/>

Valid xml:

Ex: <employ id="emp1"/>

Not Valid xml:

Ex: < employ id="emp02"/>

XSLT

- XSL stands for EXtensible Stylesheet Language, and is a style sheet language for XML documents.
- XSLT stands for XSL Transformations. In this tutorial you will learn how to use XSLT to transform XML documents into other formats, like XHTML

CSS = Style Sheets for HTML:

- HTML uses predefined tags, and the meaning of each tag is well understood.
- The <table> tag in HTML defines a table - and a browser knows how to display it.
- Adding styles to HTML elements are simple. Telling a browser to display an element in a special font or color, is easy with CSS.

XSL = Style Sheets for XML

XML does not use predefined tags (we can use any tag-names we like), and therefore the meaning of each tag is not well understood

XSL describes how the XML document should be displayed!

XSL consists of three parts:

- XSLT - a language for transforming XML documents
- XPath - a language for navigating in XML documents
- XSL-FO - a language for formatting XML documents

XSLT = XSL Transformations:

XSLT is the most important part of XSL.

XSLT is used to transform an XML document into another XML document, or another type of document that is recognized by a browser, like HTML and XHTML. Normally XSLT does this by transforming each XML element into an (X)HTML element

Root of XSl Document: <xsl:stylesheet>

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

<xsl:stylesheet version="1.0" xmlns:xsl="www.w3schools.com ">

</xsl:stylesheet>
```

The <xsl:template> Element

The <xsl:template> element is used to build templates.

The match attribute is used to associate a template with an XML element

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

<xsl:stylesheet version="1.0" xmlns:xsl=" www.w3schools.com ">

   <xsl:template  match="/">

  </xsl:template>

</xsl:styesheet>
```

The <xsl:value-of> Element

The <xsl:for-each> Element

<xsl:sort> Element

<xsl:if> Element

<xsl:choose> Element

UNIT – I

1. INTRODUCTION TO WEB TECHNOLOGIES

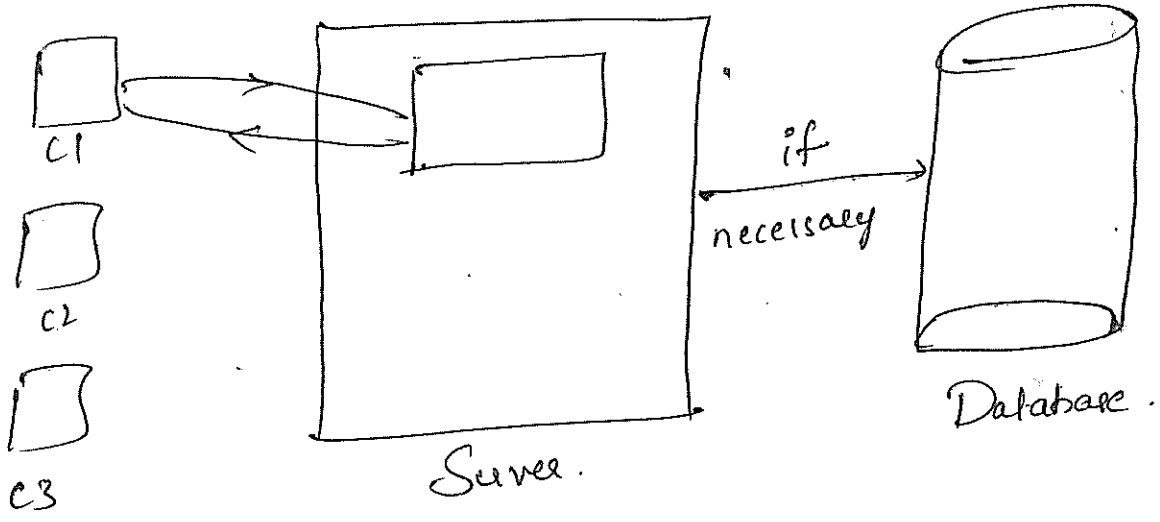
According to Sun micro Systems, 3 versions of JAVA are

- J2SE (Java 2 Standard Edition) - Technology
- J2EE (Java 2 Enterprise Edition) - Specification
- J2ME (Java 2 Micro Edition) - Specification.

Specification: It is a document i.e., released into public market before starting a project.

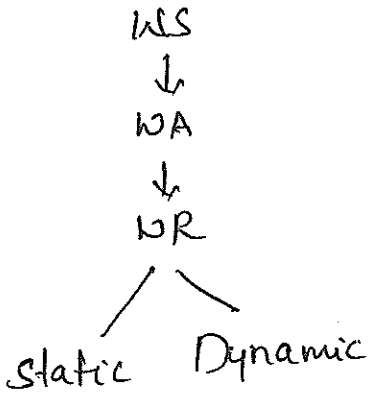
- J2EE Specification was given by Sun micro Systems, so different vendors takes the specification & they have implemented in their own way in terms of Application Servers & web servers.

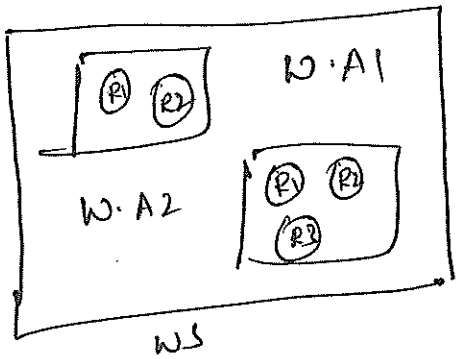
Web Server: It is a s/w that resides on a Server & listens to the client in HTTP format ie., it takes the request from web clients & sends response back to the web clients (web browsers).



Web Server contains a list of web applications. Each web application contains a list of web resources. Web resources are of two types.

1. Which generates a static content. (HTML Pgs)
2. Which generates a dynamic content (Server Site Technology programs)



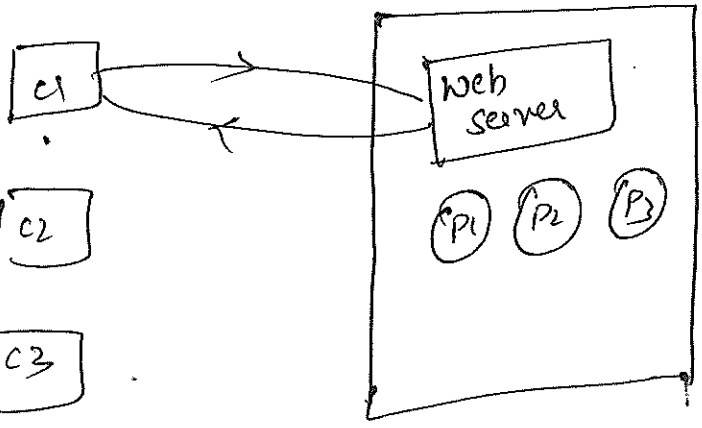


- Server site Technology programs are of two types:

  Process Based Technology (PBT)

  Thread Based Technology (TBT).

Process-Based Technology:



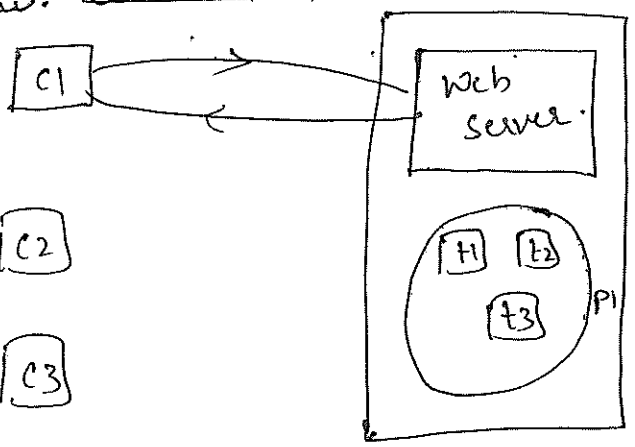
- for each & every client request it creates a seperate OS process. Th control jumping (or) context switching which takes a lot of time which degrades. It never uses the existing process if the request is coming from same client.

  Eg. CGI (Common Gateway Interface) - Specification by Apache Corporation.

This Specification was developed by C, C++, Pearl.

Note: Pearl is the most suitable language to develop CGI specifica

Thread Based Technologies:



- In TBT, for each client request a thread will be created as part of a chain process. The control jumping (or) Content Switching btwn the threads takes less amount of time. If the request is coming from the same client for which already thread is create the server will make use of same thread instead of new one. So, finally the scalability and performance of TBT is quite good when compared with PBT.

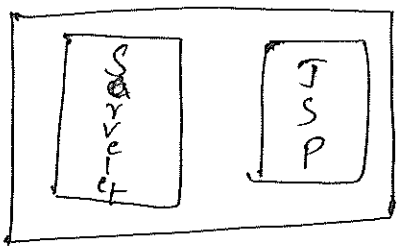
for Eg: Servlets, PHP, .Net, JSPs etc...

List of Web Severs:

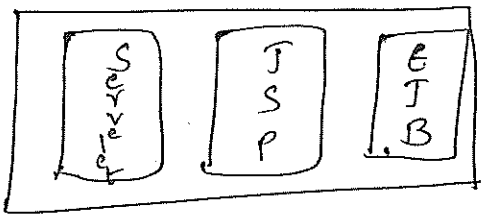
- JAVA Web Server (JWS) - Sun Micro Systems. - (3)
- Personnel Web Server (PWS) - Microsoft.
- Internet Information Server (IIS) - Micro Soft. - (2)
- Apache Web Server - Apache Corporation.
- Tomcat - Apache web Server - Apache Corporation. - (1)

Application Server : Web Server + EJB Container.

Web Server



Application Server



List of Application Severs:

- WLAS — Web Logic Application Server - BEA Company. (1)
- WSAS — Web sphere Application Server - IBM (2)
- OQAS — Archil Application Server - Archil Company (3)
- SunOne AS — Sun MicroSystems.
- JBOSS Application Server - Apache Corporation. (4)
- We have two types of Applications
  - → Web Applications (Servelets, JSP)
  - → Enterprise (EJB)

HTTP Request Message format:

GET/index.html HTTP/1.1 — Request line.

Date: — General Headers

Connection:

Host: — Request Headers

From:

Accept:

User-

Entity Headers

Message body.

Generally, we use GET / POST method for sending the request. It i
based on the type of server.

HTTP Response format:

HTTP/version code Text — Status line

Date: — General Headers

Connection:

Server: — Response Headers.

Accept:

Content-Type: — Entity Headers.

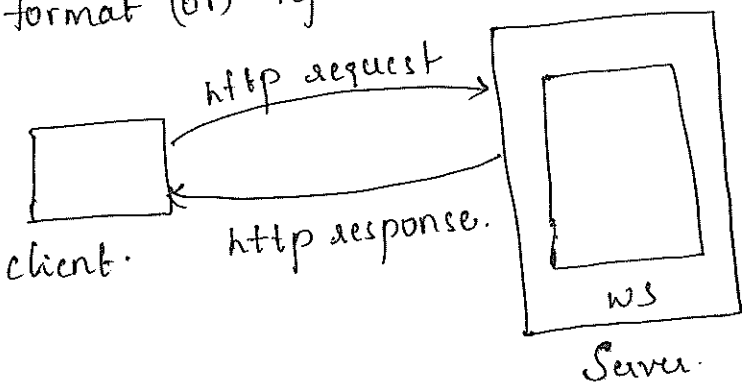
Content-Length:

Last-Modified:

Message Body.

HTML Code

General format (or) representation:



HTML

- An HTML is Hyper Text Markup Language, used to design the web pages.
- HTML is a mark-up language not a programming language.
- Markup language - contains markup tags.
- Markup language is used to present the data.
- HTML document contains HTML tags.
- HTML Tag is written in angular brackets.
- HTML Elements: The text present in between the pair of angle brackets is known as an HTML element.

  Syntax: < Element-name >

- HTML Tags are of two types. They are
  - → Paired Tags
  - → Unpaired Tags.
- Paired Tags: Tags which contain both starting & ending tag.

  Eg: < element-name > - - - - </element-name>
- Unpaired Tags: which has no closing tags. Also called as Self closing Tags (or) Single-ton tags.

  Eg: < element-name />

  Note: closing Tag is preceded by /
- H1

- HTML elements contain mainly two properties:
  - → Attribute
  - → Content.
- Attribute: It provides an additional information about an HTML element
  - → Attributes are always specified in the starting tag only.
  - → Attributes comes in pairs i.e., name="value".
- Content : It is placed in between the starting and ending tags.

Syntax of HTML element:

```
<element_name attribute_name="value"> CONTENT
</element_name>
```

Design of HTML DOCUMENT (OR) HTML PAGE (OR) HTML PROGRAM (OR) WEB PAGE

- → Open Notepad.
- → Write some HTML
- → Save the program with .html (or) .htm extension
- → View HTML Page in Browser.

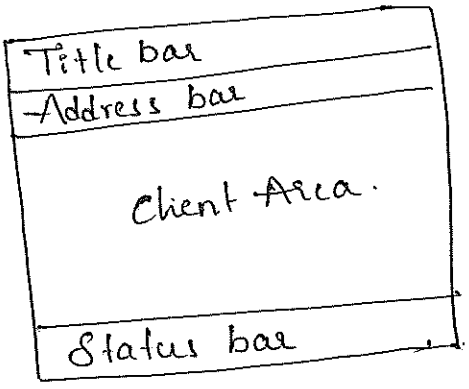
HTML DOCUMENT STRUCTURE:

```
<html>
<head>
   <title>        </title> </head>
<body>
   ....
</body>
</html>
```

- The text between <html> and </html> describes an HTML document
- The text between <head> and </head> provides info abt the document.
- The text between <title> and </title> provides title for the document.
- The text between <body> and <body> describes the visible page documen

Web Browser:

- It is a Sw program, used to access the info from web through internet.

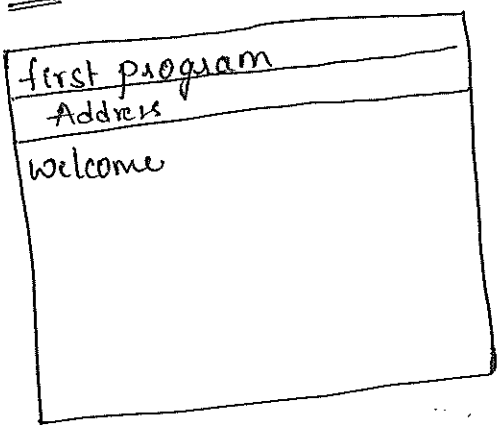


- The text between the title tags is displayed on title bar of web brow
- The text between body tags is displayed on client area of web brows

SAMPLE HTML PROGRAM:

```
<html>
<head>
<title> first program </title>
</head>
<body>
welcome
</body>
</html>
```

O/p:



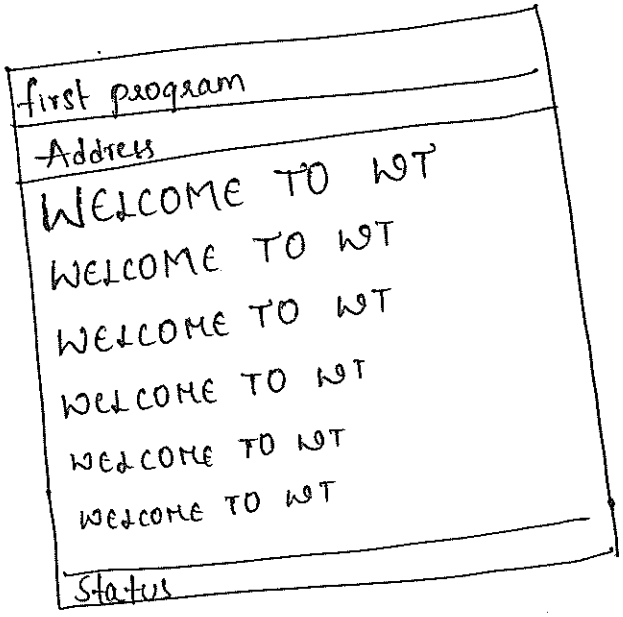
- HTML is not a case-sensitive language.

- HTML is a error free language.

USING HEADERS: We have mainly 6 headers in HTML.

```
<html>
 <head>
  <title> first program </title>
 </head>
 <body>
  <h1> WELCOME TO NT </h1>
  <h2> WELCOME TO NT </h2>
  <h3> WELCOME TO NT </h3>
  <h4> WELCOME TO NT </h4>
  <h5> WELCOME TO NT </h5>
  <h6> WELCOME TO NT </h6>
 </body>
 </html>
```

O/p:



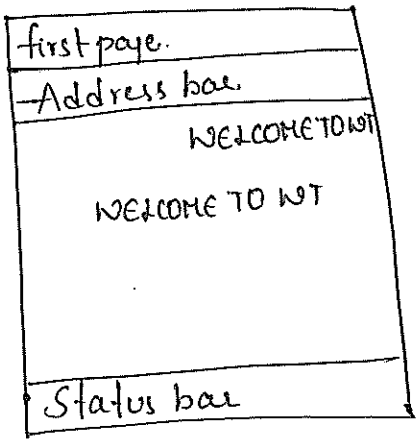
Attribute of header tags:

We can align text either left, right, center, justify.

Eg: `<h1 align="right"> WELCOME TO NT </h1>`

`<h2 align="center"> WELCOME TO NT </h2>`

O/p:



Body tag Attributes:

- bgcolor : Background Color

  Syntax: bgcolor = "red" (or) bgcolor = "#FF0000"

- Text: To change the font color.

  Syntax: text = "green" (or) text = "#00FF00"

font tag: It has two attributes and used to display the te
in different colors: Eg: S A I
red green Blue.

Syntax: `<font size="8" color="green">`

`</font>`

Eg: TEXT
red green blue yellow-

```
<html>
<head>
<title> Sample program </title>
<body>
<font size = "8" color = "red"> T </font>
<font size = "6" color = "green" > E </font>
<font size = "4" color = "blue" > X </font>
<font size = "2" color = "yellow" > T </font>
</body>
</html>
```

To change the font style:

Syn: welcome

Creating Paragraphs:

<P> is used to create paragraphs in HTML.

attributes: align = "left/right/center/justify"

Note: It adds some space before the paragraph and after the paragra

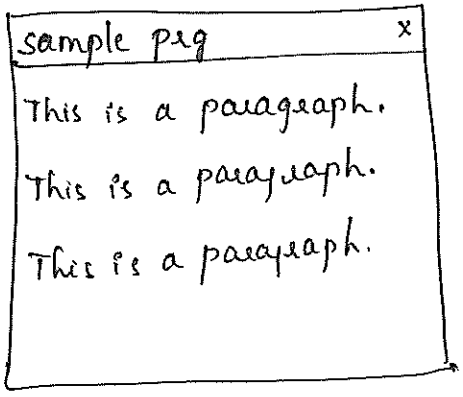
Syn: <P>
- - - - -
- - - - - -
- - - -
</P>

Program:

a)

```
<html>
<head>
<title> sample prg </title>
</head>
<body>
<p> This is a paragraph. </p>
<p> This is a paragraph. </p>
<p> This is a paragraph. </p>
</body>
</html>
```

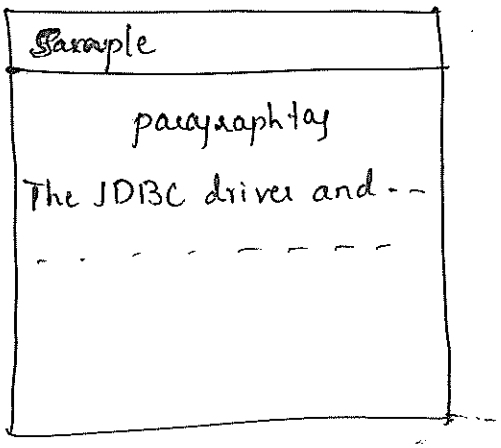
O/p:



b)

```
<html>
<head>
<title> sample </title>
</head>
<body>
<h1 align = "center"> paragraph tag </h1>
<p align = "justify"> The JDBC
Driver
and
---
</p>
</body>
</html>
```

O/p:



Pre Tag: It is used to display as it is in the web page.

Syn: <Pre>

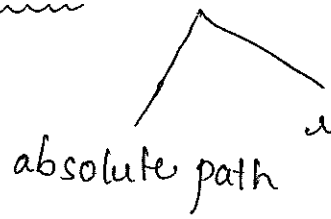
</Pre>

Sample program:

```
<html>
<head>
<title> sample </title> </head>
<body>
<pre>
The JDBC driver
uses
---
</pre>
</body>
</html>
```

Working with Images:

- We can insert images into the web page using the img tag. It is an unpaired tag.
- Attributes: src = "url" , alt = "text".



- Syntax: < img src = "url" alt = "text" > .

Sample program on absolute path:

```
<html>
<head><title> program</title><head>
<body>
<img src = " c:/ users/ sample pictures/ kavya.jpg"/>'</body>
</html>
```

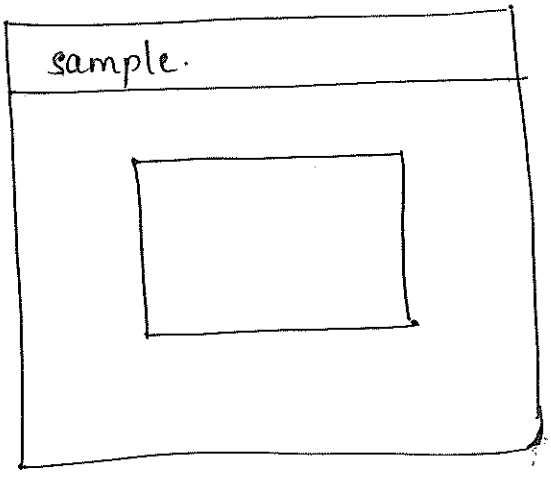
- If the image is not available, we can use alt to display a msg

Program on Relative path:

```
<html>
<head><title> sample</title></head>
<body> <center>
<img. src = " animal.jpg" alt = "Not available " width = "50%" leigth="50
border = "6" >
</center>
</body>
</html>
```

Note: The program and the image file must be in same folder (relative

O/p:



- Attribute: Align is also can be used in images, as a attribute.

align: "top/bottom/middle/left/right"

Creating Hyperlinks:

We can create hyperlinks using anchor tag i.e., <a> tag attribute is href.: href = "url"

Syn: content

Sample program:

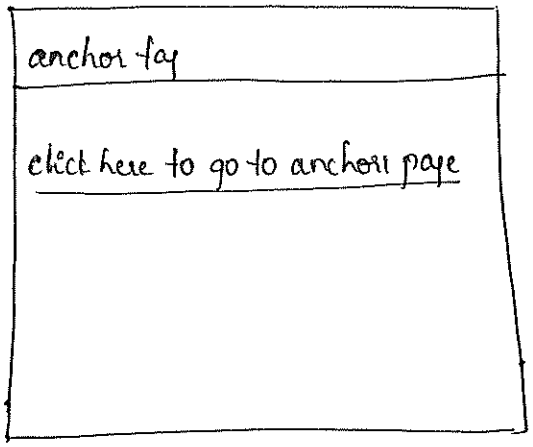
main anchor. html.

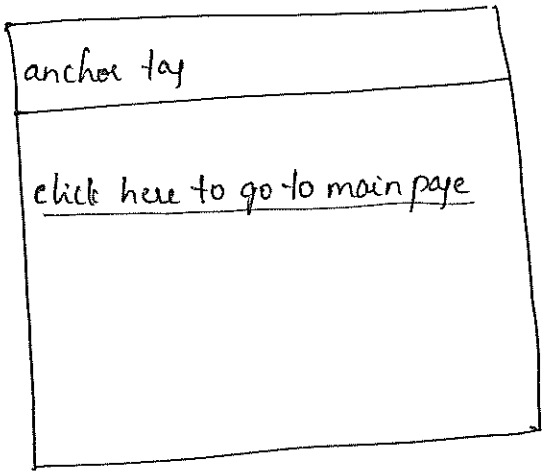
```
<html>
<head> <title> anchor tag </title> </head>
<body bgcolor = "cyan" >
<h1> Hyperlinks program </h1>
<a href = "anchor1.html" > click here to go to anchor1 page </a>
</body>
</html>.
```

anchor1. html:

```
<html>
<head> <title> anchor tag </title> </head>
<body bgcolor = "alicublue" >
<h1> Hyperlinks anchor 1 program </h1>
<a href = "mainanchor.html" > click here to go to main page </a>
</body>
</html>.
```

O/p:





To change the hyperlink text colors:

link, alink, vlink are the tags used to change the hyperlink text colors.

vlink → vistle link

alink → active link.

Syntax: link = "green" alink = "red" vlink = "yellow"

Note: These are the attributes of the body tag.

Sample program:

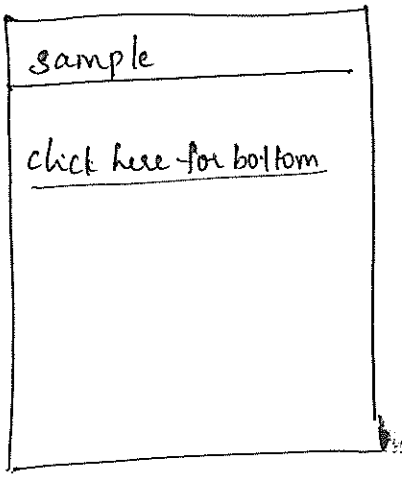
```
<html>
<head> <title> sample </title>
</head>
<body link = "green" vlink = "yellow" alink = "red" >
<h1> Hyperlinks anchor2 program </h1>
<a href = "mainanchor.html"> click here to go to main </a>
</body>
</html>
```

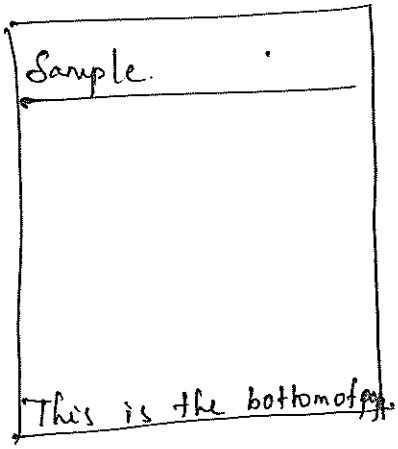
Sample program:

Aim: To move from one place to another in the same web page.

```
<html>
<head> <title> sample </title>
<body> link: "green" vlink = "red" alink: "yellow" >
<a href = "#bottom" > click here for bottom </a>
<br><br> - - - - -

— - - - — -

<br>
<hr>
<a name: "bottom" > This is the bottom of a page </a>

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

Olp:





Using images as links:

In this, images are used as a link.

```
<html>
</head> <title> sample </title>
<body>
<a href = "1.html" ><img src = "apple.jpg "></a>
</body>
</html>
```

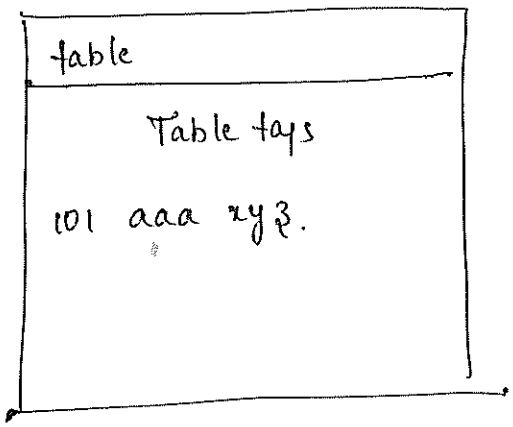
creating tables in HTML:

- Table tag is used to create tables in HTML. It is a paired tag.
- tr - table row which is used to insert rows.
- td - table data which is used to insert data values.

Sample program:

```
<html>
</head>
<title> table </title>
<body>
<h1> Table Tags </h1>
<table>
<tr>
<td> 101 </td>
<td> aaa </td>
<td> xyz </td>
<tr>
</table>
</body>
</html>
```

O/p:



th tag: (Table Header Tag)

- It is used to assign the name of columns.
- Using this tag, the text will be in bold format.

Eg: <th> Name </th>
<th> Rollnum </th>
<th> Address </th>.

Syn: <th> column-name </th>

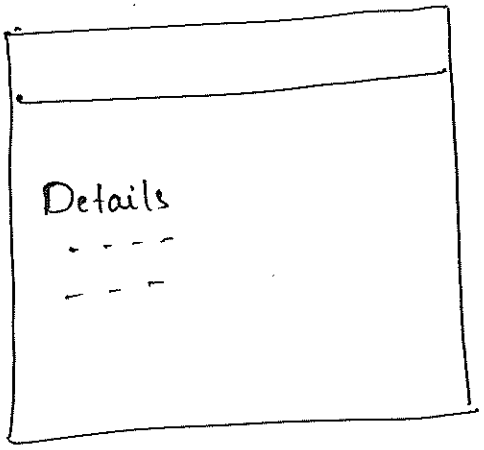
Caption tag:

It is used for writing the captions.

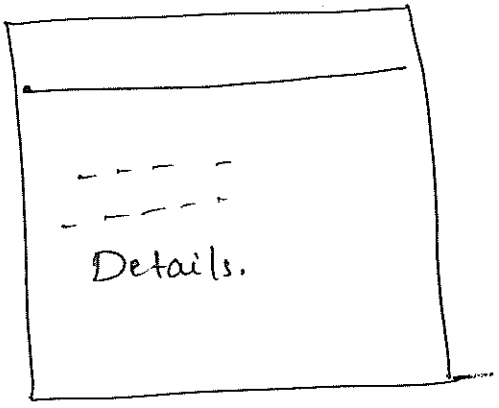
Caption tag contains an attribute named align.

align: "top | bottom" (By default: top)

Eg:
```
<table>
<caption>
<h1> Details </h1>
</caption>
....
```




```
<table>
<caption align="bottom">
<h1> Details </h1>
</caption>
----
```



Attributes of table tag:

- border : with this attribute we can assign borders to the table.

Syn: <table border = "value">

Eg.. <table border = "1">

O/p:

Name	Rollno	Address
abc	101	vjd

Aligning the table:

Syn: <table border = "value" align = "right/left">

Eg. <table border = "1" align = "right">

O/p:

Name	Rollno	Address
abc	101	vjd.

Width & height of a table:

Syn: <table border = "value" width = "value%"
height = "value%">

Eg. <table border = "1" width = "50%" height = "40%">

Changing the background of table:

Syn: <table border = "value" bgcolor = "value red">

Eg. <table border = "1" bgcolor = "red">

- for aliging the content in a cell use align in the td attribute.

Syn: <td align="centre/Right/left"> value </td>

Eg: <td align="center"> 101 </td>

Sample program:

```
<html>
<head>
<title> sample </title></body/head>
<body>
<h1 align="center"> Table tag </h1>
<table border="1">
<caption>
<h1> Details </h1>
</caption>
<th width="20%"> Rollno </th>
<th width="20%"> Name </th>
<th width="20%"> Addres </th>
<tr align="center">
<td> 101 </td>
<td> aaa </td>
<td> abc </td>
</tr>
```

```
<tr>
<td>101</td>
<td>aaa</td>
<td>xyz</td>
</tr>
</table>
</body>
</html>
```

Olp: Table tag

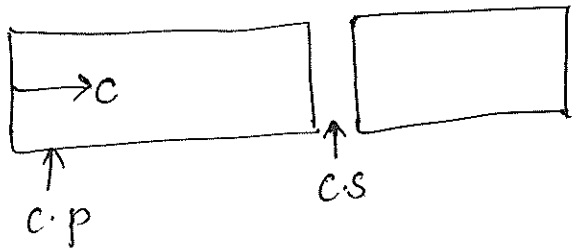
Details

Rollno Name	Name	Address
101	aaa	abc
101	aaa	xyz

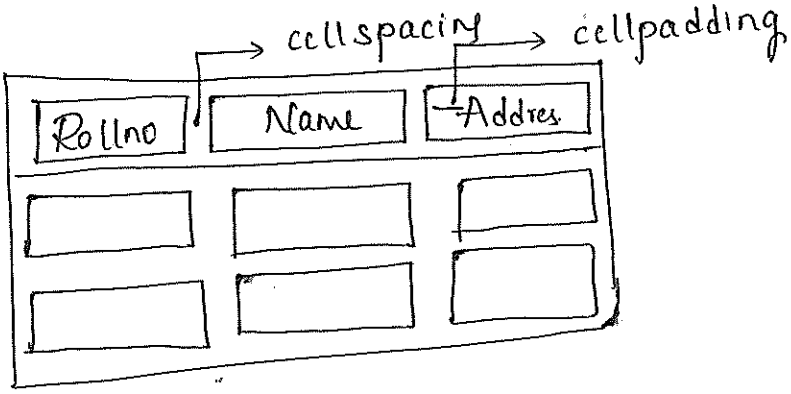
- Cell padding: It specifies the space between cell wall and cell content.

  cell spacing: specifies the space between the cells.

  Eg: `<table border="1" align="center" bgcolor="pink" cellpadding="10" cellspacing="5">`



```
<caption>
<h1> Details </h1>
</caption>
<th width = "20%" > Rollno </th>
<th width = "20%" > Name </th>
<th width = "20%" > Address </th>
<tr>
...
</tr>
```



Olp:

- Rowspan: used to combine the rows.

Eg:
```
<th rowspan = "2" > Rollno </th>
<th> Name </th>
<th> Address </th>
<tr>
<td> 100 </td>
<td> aaa </td>
</tr>
<tr>
<td> 101 </td>
<td> aaa </td>
<td> abc </td>
</tr>
```

Olp:

Rollno	Name	Addres
	100	aaa
101	aaa	abc

- colspan:
  Used to combine the columns.

Eg: <th rowspan = "2" > Rollno </th>
<th colspan = "2" > Name </th>

O/p:

Rollno	Name.	
	100	aaa
101	aaa	abc

- Develop a web page for your time table.

```
<html>
<head>
<title> sample program </title>
</head>
<body>
<h1 align = "center" > Time Table </h1>
<table border = "1" width = "20%" height = "20%" >
<th rowspan = "2" > Day </th>
<th colspan = "2" > Timings </th>
```

Working with frames:

Frame set tag is used to divide the webpage into different sections (or) to include different web pages in a HTML document.

Syntax: <frameset ----->

:

</frameset>

Dividing the webpage into different horizontal frames:

Eg: <frameset rows = " 30%, 30%, * ">

<frame src = "1.html" >

<frame src = "2.html" >

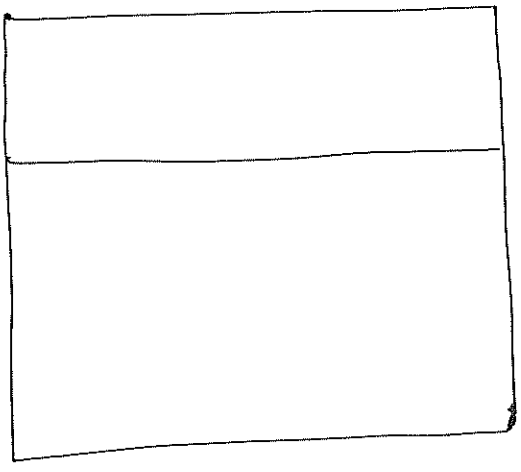
<frame src = "3.html" >

</frameset>

Sample Program:

```
<html>
<head>
<title> Sample prg </title>
</head>
<frameset rows = " 30%, *. ">
  <frame src = "1.html ">
  <frame src = "image.html ">
  </frameset>
</html>.
```

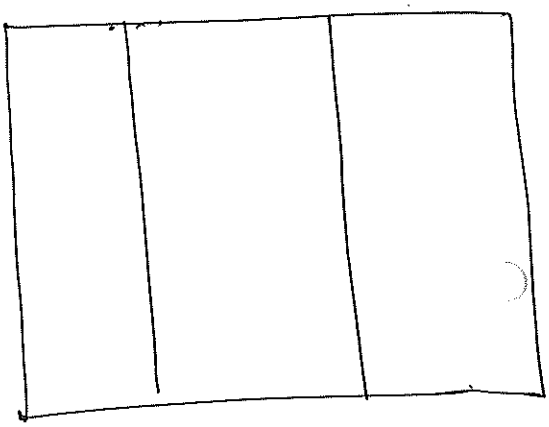
O/p:



Divide the main web page into 3 columns:

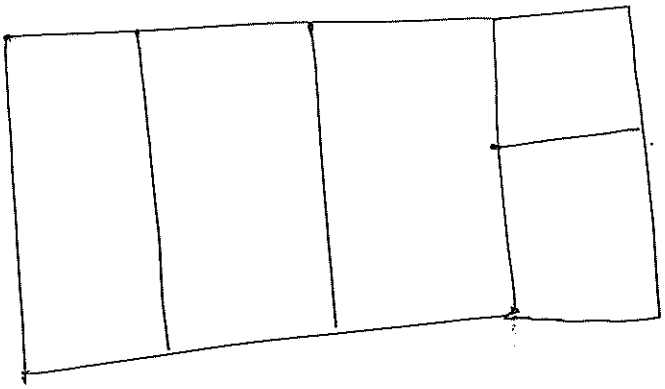
```
<html>
<head>
<title> sample </title>
</head>
<frameset columns = " 30%, 50%, * " >
    <frame src = "1. html">
    <frame src = "2.html">
    <frame src = "3. html">
</frameset>
</html>.
```

Olp:



Prg:

Olp:



```
<html>
<head>
<title> sample </title>
</head>
<frameset cols = " 30% 30% 20% 20% " >
    <frame src = "1.html">
    <frame src = "2.html">
    <frame src = "3.html">
```

```
<frameset rows = " 50%,* ">
  <frame src = " 4.html ">
  <frame src = " 5.html ">
</frameset>
</frameset>
</html>.
```

(or)

```
<html>
<head>
<title>sample</title>
</head>
<frameset cols = " 70%,* ">
    <frameset cols = " 30% 30% 40% ">
      <frame src = " 1.html ">
      <frame src = " 2.html ">
      <frame src = " 3.html ">
    </frameset>
  <frameset rows = " 50%,* ">
      <frame src = " 4.html ">
      <frame src = " 5.html ">
  </frameset>
</frameset>
</html>
```

Note:

- To remove the borders, attribute used is frameborder="0". It is an attribute of frameset tag.

Syn: <frameset rows="30%, *" frameborder="0">.

- To Open a Hyperlink webpage in a new tab

 click here

↓

It will open a webpage in new tab.

Eg: <html>

<frameset cols="70%, *">

<frame src="1.html">

<frame src="2.html" name="abc">

</frameset>

</html>.

1.html:

<html>

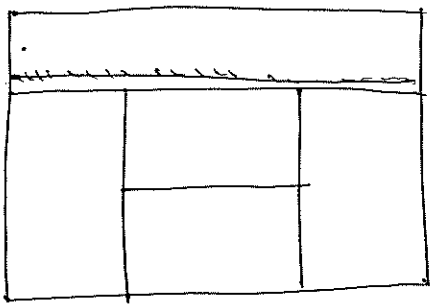
<body bgcolor="aliceblue">

<h1> Hyperlink program </h1>

 click here </body>.

</html>

Design a webpage for the given layout.



```
<html>
<frameset rows="20%,*">
    <frame src="image.html">
    <frameset cols="30% 40% 30%">
    <frame src="results.html">
     <frameset rows="50%,*">
     </frameset>
    <frame src="news.html">
    </frameset>
    </frameset>
</html>
```

<frame src="1.html">
<frame src="2.html">

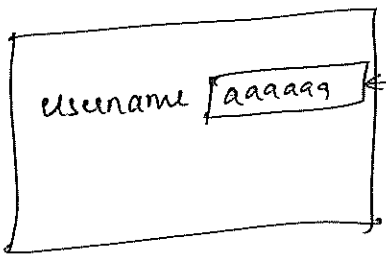
Using Forms:

- forms are used to create applications in the web page. It is a paired tag. And the input tag is the nested tag in the form tag.

- Syn. `<form>` ⋮ `</form>`

Eg.
```
<form>
  username <input type="text" size="30" maxlength="6"
                  name="un">
    :
</form>
```

Op:



. To create password Textboxes:

```
<form>
  <input type="password">
  :
</form>
```

The difference between password text boxes and the text boxes is that characters are hidden in pwd tb bcz for the security purp

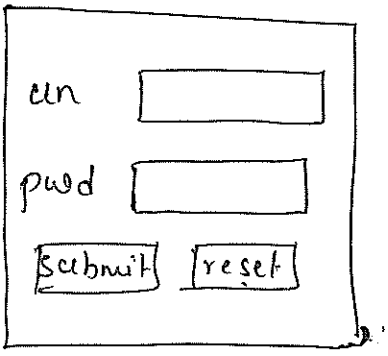
. To create submit Button:

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

. To create reset Button:

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

Design a web page using a Table tag:




```
<html>
<head> <title> sample </title>
</head>
<body>
<table>
<tr>
<td>
<form>
un <input type = "text" > </form>
</td> </tr>
<tr> <td> <form> pwd <input type = "pwd" > </form>
</tr> </td>
<tr> <td> <form>
<input type = "submit" value = "submit" > </form>
<input type = "reset" value = "reset" >
</form>
</td>
</tr>
</table>
</body>
</html>
```

action attribute & method:

- form tag has 2 attributes. action & method.

Syn: <form action="url" method="get/post">

- when the user clicks on submit button, the request goes to the url specified in the action attribute of a web server.
- method attribute is used to communicate with the server.
- ** The main methods are get/post. Default method is get.

Get	Post
• It appends the query string at the end of url.	• It doesn't append the query string.
• It can send the data to the server using query string.	• It sends the data using SSL (secured Socket Layer).
• It sends a limited data	• It is used to send unlimited amount of data.
• No Security	• Security is provided.
• Bit fast	• Bit slow.

To create checkboxes: (☐)

Eg: <h2> Hobbies </h2>

 reading books

<input type = "checkbox" name = "hobby" value = "books">

 playing

<input type = "checkbox" name = "hobby" value = "play">

 listening music

<input type = "checkbox" name = "hobby" value = "music">

.

- checkboxes are used for selecting the multiple values.

To create radiobuttons: (○)

Eg: male
<input type = "radio" name = "gender" value = "male">

 female

<input type = "radio" name = "gender" value = "female">

- used for selecting the single value.

Default text within the Text Box:

<input type = "text" name : un value : "enter" >

The drawback of this approach is that the text is appended at the end of enter. If we want we can delete and enter the new text.

(or)

<input type : "text" name : un placholder = "enter">

The enter is deleted when we press the key.

To create Text Areas:

<textarea rows = "10" cols : "10">

- - -

- - -

</textarea>

Selection Control Boxes:

used for creating menu-driven boxes. Created by using the select tag.

<select>

<optgroup label = "General Hobbies">

<option value : "playing" > Playing </option>

<option value : "music" > music </option>

</optgroup>

Js using text
s.r will tell
x
also app for

```
<optgroup label = " Educational Hobbies ">
    <option value = " search " > net </option>
    <option value = "stamps" > stamps collection </option>
</optgroup>
</select>
```

. Draw back is that, used to select only a single option.

for selecting multiple values: we can use multiple atkibute

< select multiple >.

Note: Use the control button to select multiple values.

Lists:

There are three types of lists.

- Unordered lists
- Ordered lists
- Definition lists.

UnOrdered Lists:

. To represent unordered lists we use the tag ul tag.

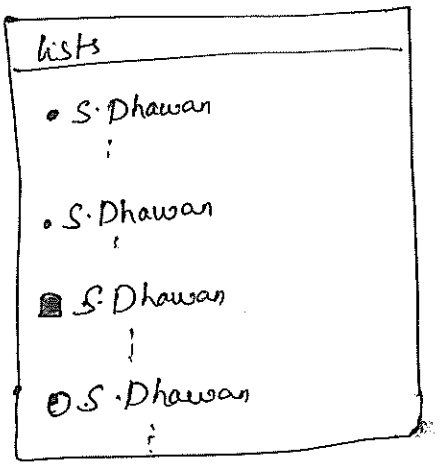
. ul tag is having a nested tag called li.

Syn:
```
<ul>
    <li>
    <li>
</ul>
```

Ex:

```
<html>
 <head>
 <title> lists </title>
 </head>
 <body bgcolor="aliceblue">
 <h2> Indian ODI team </h2>
 <ul>
   <li> S. Dhawan </li>
     :
 </ul>

 <br>
 <ul type="disc"> (Default)
 <li> S. Dhawan </li>
   :
 </ul>
 <br>
 <ul type="square">
 <li> S. Dhawan </li>
   :
 </ul>
 <br>
 <ul type="circle">
   <li> S. Dhawan </li>
     :
 </ul>
 </body>
 </html>
```

O/p:



Ordered Lists:

Syn: `<ol>`
`<li> .... </li>`
:
`</ol>`

Op:

tg:
```
<ol>
<li> kavya </li>
</ol>
```
1. kavya
:

```
<ol type="1" > (Default)
<li> kavya </li>
</ol>
```
1. kavya

```
<ol type="a" >
<li> kavya </li>
</ol>
```
a. kavya.

```
<ol type="A" >
<li> kavya </li>
</ol>
```
A. kavya

```
<ol type="I" >
<li> kavya </li>
</ol>
```
I. kavya

```
<ol type="i" >
<li> kavya </li>
</ol>
```
i. kavya

```
<ol type="i" start="4">
<li> kavya </li>
</ol>
```

4. kavya
5. :
6. :
7. :

- Definition lists:

Syn:
```
<dl>
  <dt> web Technologies </dt>
  <dd>
    abcdefghi - - - - - -
    - - - - - - - - - - -
    - - - - -
  </dd>
</dl>
```

O/p: Web Technologies
abcdefghi - - - - - - - -
- - - - - - - - - - - - -

Marquee Tag:

This is used to move the text continuously in a web page.

Syn:
```
<marquee> content </marquee>
```

Attributes of marquee tag:

- behavior = "scroll / slide / alternate"
  ↓
  Default

- alternate - back & forth
- slide - only one time
- scroll - infinite times.

- direction : "right / left / up / down"
  ↓
  By default.
- height & width: It can be represented in percentages or pixels.
- bgcolor : "rgb(120, 120, 120)"

** By using 3 ways we can assign color in webpage.

- colorname
- Hexadecimal
- rgb mode.

- loop : How many number of items, it will repeat.

  Syn: loop = "value".

- scrollamount : to move the text fast.

  Syn: scrollamount = "value" (integer value)

  The default value of scrollamount in marquee tag is 6.

- scrolldelay : to slow the moving of text.

  Default value is 85

  Syn: scrolldelay = "value" (Value is value in milliseconds).

- To include audio files:

  Audio tag is used to include audio files in the web page.

  Eg: <audio controls>
  <source src="055SHRIDI.mp3" type="audio/mpeg">
  </audio>

- To include video files:

With the help of video tag, we can include video files.

↳ directly it plays.

Eg: <video width="320" height="240" autoplay>

<source src="movie.mp4" type="video/mp4"> content...

</video>

(or)

↳ it is used for including the contro. like sound, screen... etc....

<video width="320" controls>

- To include youtube videos:

iframe is used to include the youtube videos.

↓

(inline frame)

1) Eg. <iframe width="420" height="345"

src="url" autoplay>

</iframe>

(or)

2) <html>

<body>

<object width="420" heigth="345"

data="url" autoplay>

</object>

</body>

</html>.

(or)

3) <html>
<body>
<embed width="420"
src="url" autoplay> </embed.
</body>
</html>.

- To make the text as Bold: Content will be displayed in Bold format.

Syn: content
(or)
 content .

- Italic:

Syn: content
(or)
 content → emphasis tag.

- Superscript:

Syn: content < sup > content. </sup>

Eg: x < sup > 2 </sup> O/p: x^2

- Subscript:

Syn: content < sub > content

Eg: x < sub > 3 </sub> O/p: x_3

- Horizontal Ruler: for drawing horizontal ruler.

<hr>

- bdo tag is used to write the paragraph from right to left.
  ↳ bidirectional overridden.

Eg: <html>
<body>
<bdo dir="rtl"> This paragraph will go right to left</>

</body>
</html>.

- Strikeoff text:
  s tag is used.

Syn: <s> web </s> o/p: ~~web~~

(or)

<del> web </del> o/p: ~~web~~

(or)

<strike> web </strike> o/p: ~~web~~

- To display the text in the bigger format using big tag.

Syn: <big> content </big>.

If we want, even big format use nested big.

Tags:

- <html> ... </html>
- <head> ... </head>
- <body> ... </body>
- <title> ... </title>
- ...
- <pre> ...
- <h1> ... </h1>
- <h2> ... </h2>
- <h3> ... </h3>
- <h4> ... </h4>
- <h5> ... </h5>
- <h6> ... </h6>
- <img ... >
-
- ...
- <$tong> ... </$tong>
- ...
- ...
- <u> ... </u>
- <hr> ...
- <font> ...
-
- <a> ...
- <table> ... </table>

- <caption> --- </caption>
- <th> --- </th>
- <tr> --- </tr>
- <td> --- </td>
- </frameset> --- </frameset>
- <frame>
- <form> --- </form>
- <input>
- <textarea> --- </textarea>
- <select> --- </select>
- <optgroup> --- </optgroup>
- <option> --- </option>
- <audio> --- </audio>
- <video> --- </video>
- <iframe> --- </iframe>
- <object> --- </object>
- <embed> --- </embed>
- <marquee> --- </marquee>
- ---
- ---
- <d> --- </dl>
- ---
- <dd> --- </dd>
- <dt> --- </dt>
- ₋₋₋
- ⁻⁻⁻
- <bdo> --- </bdo>

UNIT - II

The main purpose of CSS is to seperate content from presentation where css handle presentation & html deal with content.

Types of CSS:

- inline CSS - inside the tags.
- internal CSS - inside the program.
- external CSS - outside of the program

1) Inline CSS:

Syntax:

```
<tagname style="Property: value; Property: value">
```

Eg:

```
<html>
  <body>
   <h1 style = "color : green; margin-left : 30px">
     Welcome to CSS
   </h1>
   <p> Normal text </p>
  </body>
</html>.
```

2) Internal CSS:

Syntax:

```
<style>
  Selector { property: value;
             property: value;

</style>
```

Eg: <html> <head><style>

```
h1 { color : blue;
     margin-left : 30px;
   }
</style>
</head>
<body>
<h1> welcome to css </h1>
<p> Normal text </p>
</body>
</html>
```

3) External css : writing styles in seperate files. we have to link it to our prg.

Eg: style1.css

```
h1 { color: red;
     margin-left : 30px;
   }
```

```
<html>
<head>
<link rel: "stylesheet" type: "text/css" href: "style1.css">
</link>
</head>
<body>
<h1>welcome to css </h1>
<p> Normal text </p>
</body>
</html>.
```

Problems with Styles:

```
<html>
<head>
<style>
h1 { color: blue; background-color: alice blue;
}
</style>
</head>
<body>
<h1> HAI </h1>
<h1> WELCOME TO </h1>
<h1> CSS </h1>
</body>
</html>
```

O/p:

HAI

WELCOME TO

CSS

Here the problem is, for the header h1 the color displayed is red whenever using the h1 header.

To overcome this classes are used.

Whenever writing the clases . is used.

```
<html>
<head>
<style>
h1.red { color: blue ; background-color: alice blue ; }
</style>
</head>
```

```
<body>
<h1 class="fred"> HAI </h1>
<h1> welcome </h1>
</body>
</html>
```

Olp:

HAI → Blue

WELCOME → Black.

Anonymous Classes:

To apply same style for different html elements.

Syn: .class { styles }.

```
<html>
<head>
<style>
.fred= { color: blue ; background-color: aliceblue; }
</style> </head>
<body>
<h1 class="fred"> HAI </h1>
<p class = "fred"> This_is_a wonderful subject </p>
</body>
</html>
```

Olp:

HAI

This is a wonderful subject.

Using ID's:

```
<html>
<head>
<style>
#fred { color: red; background-color: aliceblue; }
</style>
</head>
<body>
<h1 ID="fred">HAI</h1>
<h1> WELCOME</h1>
</body>
</html>
```

*** Diff btwn class & ID:

Each element can have only one ID.

classes

```
<html>
<head>
<link rel="stylesheet" type="text/css" href="style1.css">
</head>
<body>
<h1 class="fred-fed">welcome</h1>
</body>
</html>
```

O/p:

WELCOME.

style1.css:

```
.fred { color: red;
background-color: aliceblu
}
.fed { margin-left: 50 px
```

id's

```
<html>
 <head>
 <link rel="stylesheets"  type="text/class"  href="style.css">
 </head>
 <body>
 <h1 id="fed fed"> HAI </h1>
 <h2 id="fred"> HAI </h1>
 <h1 id="fed"> WELCOME </h1>
 </body>
 </html>
```

O/p:

HAI

HAI → Red.

WELCOME → Sopra.

style

```
#fred {  }
#fed {  }
```

- for including the background image.

```
<html>
<body background="filename.jpg">
</body>
</html>
```

CSS Backgrounds:

- The css background properties are used to define the backgro effects for elements.

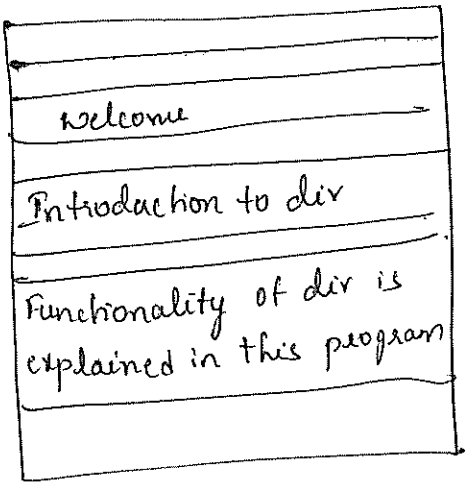
Properties:

1) background-color : color;
2) background-image : url("filename);

Using div tag:

```
<html>
<head>
<style>
h1 {
    background-color: green;
   }
div
   {
    background-color: lightblue;
   }
p {
    background-color: yellow;
   }
</style>
</head>
<body>
<h1> welcome </h1>
<div>
    Introduction to div
<p> Functionality of div is explained in this program
</p>
</div>
</body>
</html>.
```

O/p:



3) Background-repeat:

Syn: Background-repeat : repeat-(x/y); direction.

Eg: Background-repeat : repeat-x;

It will repeat image in specified direction only.

Background-repeat : no-repeat; Default: both

It will not repeat & displays once.

4) Background-attachment:

Syn: Background-attachment : fixed/scroll;

Eg: Background-attachment : fixed;

5) Background-position:

To change position of image.

Syn: Background-position : position;

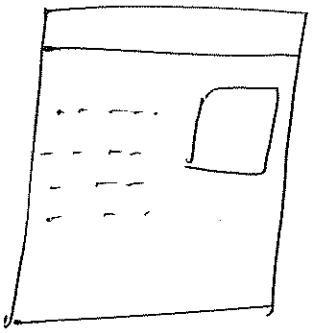
Eg: Background-position: right top;

Program:

```
<html>
 <head>
 < style>
  body {
      background-image : url ("200x00.jpg");
      background-repeat: no-repeat;
      background-position: right top;
          margin-right : 350px;
      }
</style>
```

```
</head>
<body>
<p>. . . . . </p>
</body>
</html>
```

Olp:



(3)

Shorthand Notation:

Syn: Background : color image repeat attach position.

program:

```
<html>
<head>
<style>
body {
      background : #ffff url("200x00.jpg") no-repeat fixed
                                    right bottom;
      }
</style>
</head>
<body>
<p>. . . . . </p>
</body>
</html>
```

. Order must be correct in shorthand notation i.e., color, image, repeat, attachment, position.

CSS colors:

Syntax: color: value;

Text styles:

- text-decoration: value

  values - none/underline/overline/line-through.

- text-transform: value

  values - none/uppercase/lowercase/capitalize.

- text-align: value;

  values - left/right/center/justify.

- text-indentation: value;

  value in percentages.

font styles:

- font-style: normal/italic/Oblique.
- font-size: value;

  values - small/medium/large/smaller/larger/in pixels/in %/inch
- font-weight: value;

  values - normal/bold/bolder/lighter/number(100, ... 900).
- font-family: value;

  "Times New Roman", "Times", "Serif".

  Eg: font-family: "Time-New Roman", Times, Serif.

Borders:

Border-style: dotted/dashed/solid/double/groove/inset/outset/none/hidden/ridge.

Margins:

- Margin-left
- margin-right
- margin-top
- margin-bottom.

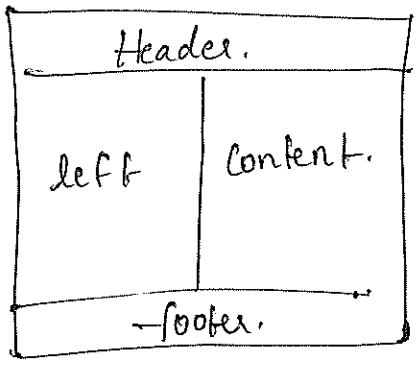
Shorthand Margin:

margin: top right bottom left;

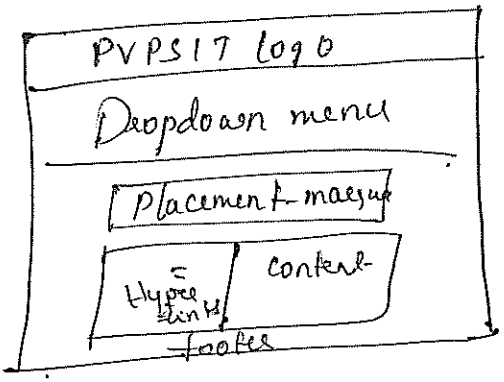
Eg: margin: 100px 200px 300px 150px;

Design web pages:

1)



2)



Text Styles:

letter-spacing: value; text-indent: value;

word-spacing: value; line-height: value(1.5);

(a)

```
<html>
<head>
<link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
<a href="1.html">click here</a>
</body>
</html>
```

style.css

```
a {
    text-decoration: none;
}
```

(b)

```
<html>
<head>
<style>
a:link { background-color: yellow; }
a:visited { back-ground-color: cyan; }
a:hover { background-color: lightgreen; }
a:active { background-color: pink; }
</style>
```

</head><body>

This is link

</body></html>.

Note: a:hover must come after a:link and a:visited in the css definit

in order to be effective.

a:active must come after a:hover in the css to be effective

Multiple styles:

```
<html>
<head>
<link rel="stylesheet" type="text/css" href="1.css">
<link rel="stylesheet" type="text/css" href="2.css">.
</head>
<body>
<h1>Welcome</h1>
<h2>Multiple styles</h2>
</body>
</html>.
```

1.css

```
body { background-color: blue;
       }

h1 { margin-left: 20px;
     color: red;
   }
```

2.css

```
h2
{ color: blue;
  margin-left: 20px
}
```

JAVA SCRIPT:

- HTML to define the content of web page.
- CSS to specify the layout of web pages.
- JAVA SCRIPT to program the behaviour of web pages.
- DHTML = HTML + CSS + Java Script.
- We can enhance the web pages by adding more dynamism and interactivity.
- Java script is an interpreted, client side and object oriented scripting language. (used to reduce the burden on the server site).
- We can embed java script into an html document by using <script> --- </script> tag.
- <script> tags can be placed in either body or in head.
- When you want the script to run while the web page is loading, place <script> tag in the body portion
- If you want the script to run only when the user performs an action, such as clicking a link, place the <script> tag in the head portion.

Client Side Scripting:

- Web Browser executes the client side scripting that resides at the user computer.
- The browser receives the page sent by the server and executes the client side scripts.

- Client side scripting cannot be used to connect to the databases on th web Server.
- Client side scripting can't access the file system that resides at the web server.
- Response from a client side script is faster as to compared to a server side script because the scripts are processed on the local computer.
- Client side scripting is possible to be blocked by the user.

Types of JAVA SCRIPT: 2 types

- internal < body / head.
- external.

The extension of this file is .js and placed within script tag.

Sample program on internal JS:

```
<html>
<head>
</head>
<body bgcolor="Aliceblue">
<script type="text/javascript"
  language="javascript">
document.write("hai hello");
</script></body></html>
```

Sample program on external Java Script:

Script contains a special attribute called as src.


```
<html>
<head>
</head>
<body bgcolor = "silver">
<script type = "text/javascript" language = "javascript" src = "first.js">
</script>
</body>
</html>
```

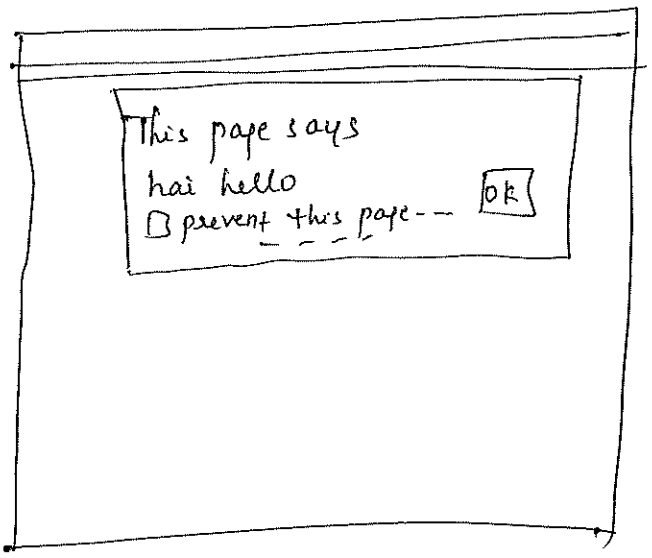
first.js

```
document.write("hello");
```

Creating alert dialog boxes:

```
<html>
<head>
</head>
<body bgcolor = "silver">
<script type = "text/javascript" language = "javascript">src = ""
  alert("hai hello");
</Bcript).
```

O/p:



Event Handling Techniques in JS:

① Onclick events: a) On click using User defined functions.

Eg:
```
<html>
  <head>
<scaipt type="text/javascript". language="javascript">
function findper(totalmarks)
{  if(totalmarks<100)
       document.write("per"=+(totalmarks/3)+"<br>")
}
</scaipt>
</head>
<body bgcolor="silver">
<form>
<input type="text" id="total"> <br><br>
<input type="button" id="percen" value="percen".
onclick="findper(total.value)"> <br><br>
</form>
</body>
</html>
```

b) Using Predefined functions:

```
<html>
<head>
</head>
<body bgcolor="silver">
```

```
<form>
<input type="text" id="field"><br><br>
<button
onclick="alert('Hi'+ document.getElementById("field").value)">
click</button>
<br>
</form>
</body>
```

2) Onload Event:

```
<html>
<head>
<script type="text/javascript" language="javascript">
function myfun()
{
    alert("welcome");
}
</script>
</head>
<body onload="myfun()">
</body>
```

3) Onmouseover Event:

```
<html>
<head>
<script type="text/javascript" language="javascript">
function moover()
{    alert("Welcome");  }
```

```
</script>
</head>
<body>
<h1 onmouseover = "mouorr()" > HAI</h1>
</body>
</html>.
```

④ onmouseout:

```
<html>
<head>
<script type = "text/javascript" language = "javascipt ">

function mouout()
 {    alert("Welcome");

 }
</script>
</head>
<body>
<h1 onmouseout = " mouout()" > HAI</h1>

</body>.
```

⑤ onreset event:

```
<html>
<head>
<script type= "text/javascript" language = "javascript" >

function rese()
 {    alert(" ARE YOU SURE TO RESET THE DATA");
 }    </script>
```

```
<body>
<form onreset = "resel()">
<input type = "text" value = "enter the name">
<br><br>
<input type = "reset" value = "reset">
</form
</body>
</html>
```


onsubmit event:

```
<html>
<head>
<script type = "text/javascript" language = "javascript">
function sub()
{ alert("ARE YOU SURE TO SUBMIT");
}
</script>
<body>
<form onsubmit = "sub()">
<input type = "text" value = "Name">
<br>
<input type = "submit" values "submit">
</form>
</body>
```

Draw of mouseover can be handled like this.

Eg: <html>

<body>

<div onmouseover="move(this)" onmouseout="mout(this)"

style="background-color: red; width: 12px">

mouse over me

<script>

function move(obj){

obj.innerHTML = "Thank you"

function mout(obj){

obj.innerHTML = "Mouse over me"

}

</script>

</body>

onfocus event:

Eg:

```
<html>
<body>
Enter your name:
<input type="text" onfocus="myfunction(this)">
<p> when the input field gets focused, a function is triggered
which changes the background color.</p>

<script>
function myfunction(x) {
    x.style.background = "green";
}
</script>
</body>
</html>.
```

(or)

```
<html>
<body>
<p> when input field gets focused, a function is triggered whi
changes the background color</p>
Enter your name: <input type="text" id="myInput"
onfocus="focusFunction()">
<script>
function focusFunction() {
  document.getElementById("myInput").style.background = "red
}
</script>
</body>
</html>.
```

- onblurr Event:

```
<html>
<body>
Enter your name:: onfocus = myfunc(this)
<input type = "text" onblur = mic(this)>
<p> when we click on any other parts of text box, the backgrou
color is changed </p>.
<script> function myfunc(x) { x.style.background = "yellow"; }
function mic(y){
    y.style.background = "alice blue";
}
</script>
</body>.
```

Variables in JavaScript:

Rules:

- Names can contain letters, digits, underscores and dollar signs.
- Names can begin with dollar, _ and letters.
- Names are case Sensitive (a and A are different).
- Reserved words cannot be used as names.

Syn: var variablename = "value";

Eg:

```
<html>
<head></head>
<body>
<script>
```

```
var a = "10";
var b = "Welcome";
var c = "20.5";
var d = 'a';
   document.write(a + "<br>");
   document.write(b + "<br>");
</script>
</html>.
```


Eg.
```
<html>
  <body>
   <script>
    var a;
     document.write(a);   // undefined value.
   </script> </body>
   </html>
```

Increasing the height of a text box:

```
<form>
<input type="text" style="height:50px; font-size:16pt;"
name="item" .align="left">.

</form>.
```

Using arrays in JavaScript:

An array is a collection of elements in JavaScript.

Syn: var arrayname = new Array(size);

Eg: var a = new array(3);

To initialize the arrays:

```
a[0] = 1 ;
a[1] = 2.5 ;
a[2] = "hai" ;
```

(or)

Syn. var array = new Array(values);

Eg: var b = new Array(1, 2.5, "hai");

Array properties:

length: to know the size of array.

Eg: a.length = 3

methods in array Objects:

- concat
- reverse
- join
- sort

** → join is same as concat. The only difference is the outpa
of join method is treated as a single entity.

```
<html>
<body>
<script>
var a = new Array(3);
var b = new Array("hai", "welcome", "to wT");

a[0] = 1;
a[1] = 2.5;
a[3] = "hai";
   document.write("array length is " + a.length + "<br>");
   document.write("concated string is " + b.concat() + "<br>");
   document.write("reverse string is " + b.reverse() + "<br>");
   document.write("joined string is " + b.join() + "<br>");
   document.write("Sorted order is " + a.sort() + "<br>");

</script>
</body>
```

O/p:

array length is 3

concated string is hai, welcome, to wT

reverse string is to wT, welcome, hai

joined string is hai, welcome, to wT

Sorted order is 1, 2.5, hai.

Date Object:

```
<html>
<head>
<script>
var dt = new Date();
var year = dt.getFullYear();
var date = dt.getDate();
var day = dt.getDay();
var month = dt.getMonth();
var days = new Array("sunday", "monday", "tuesday", "wednesd
"thursday", "friday", "saturday");
var months = new Array("January", "febeauary", "March", "April
"may", "June", "july", "august", "september", "october", "Novem
"December");
</script>
</head>
<body>
<script>
document.write("Today is " + days[day] + " " + date +
months[month] + " " + year);
document.write("<br>");
document.write(dt.toString());
</script>
</body>
</html>.
```

String Object:


```
<html>
<head>
</head>
<body>
<script>
var st = "welcome to javascript programming";
var str = "Happy programming";
document.write(st.length + "<br>");
document.write(st.bold() + "<br>");
document.write("The character at 11th position is " + st.charAt(11)
+ "<br>");
document.write(st.concat(str) + "<br>");
document.write(str.substring(0,19) + "<br>");
document.write(st.toLowerCase() + "<br>");
document.write(st.toUpperCase() + "<br>");
if(st.match(/java/)){
document.write("The string contains java " + "<br>");
}
if(str.repl.str.search(/Happy/) != -1)
{ document.write(str.replace(/Happy/, "Enjoy") + "<br>");
}
</script>
</body>.
```

Functions in Java Script:

a)

```
<body>
<script>
function func(a,b,c)
{
    var sum;
    sum=a+b+c;
    document.write(sum);
}
document.write("The function is to called.....");
func(10,20,30);
</script>
```

b) functions Using a retaen type:

```
<script>
function func(a, b, c)
{
    var sum;
    sum=a+b+c;
    retaen sum;
}
document.cosite("The function is :---");
  var a =func(10,20,30);
document.write(a);
</script>
```

Q: Write a JavaScript program to find the factorial of a number using functions:


```
<script>
function fact(n)
{
    var f;
    var i;
    f=1;
    for(i=1; i<=n; i++)
    {
        f=f*i;
    }
    return f;
}
document.write("factorial is :: ");
var n=fact(5); document.write(n);
</script>
```

Q: Java Script to find the factorial of a number using recursion.

```
<script>
function fact(n)
{
    if(n==0 || n==1)
        return 1
    else
        return (n* fact(n-1));
}
var x= fact(5); document.write(x);
```

Built-in functions:

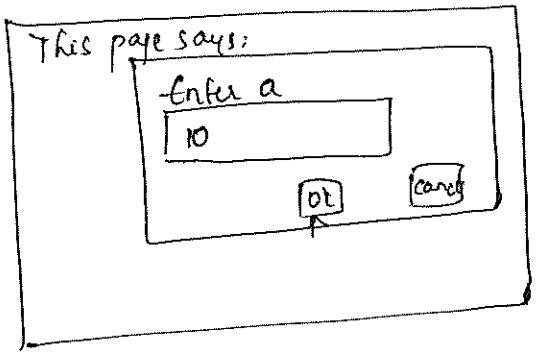
- alert : Eg: alert(hai);

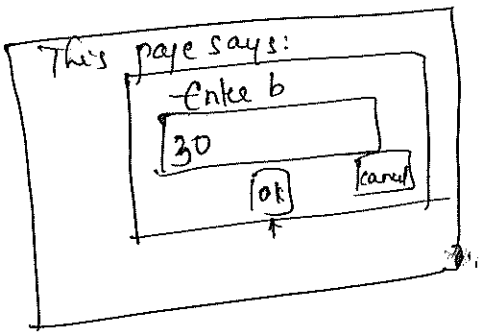
- prompt:

Eg: <script>

```
var a = prompt("enter a");
var b = prompt("enter b");
var s = a+b;
document.write("Sum is " + s);
```

</script>.





O/p: 1030.

- parseInt:

```
<script>
var a = prompt("enter a");
var b = prompt("enter b");
var c = parseInt(a);
var d = parseInt(b);
var sum = c+d)
document.write(sum); </script>.
```

- confirm: Eg: <script>

```
var n = confirm("Are you sure to delete this file");
if(n)
    document.write("The document is deleted");
else
    document.write("welcome");
</script>
```

(17)

- eval:

```
<body>
<script>
eval("x=25; y=20; document.write(x+y)");
document.write("<br>" + eval("20 * 4"));
</script>
</body>
```

O/p:

45
80

- isFinite:

```
<script>
document.write(isFinite(345));
document.write("<br>" + isFinite("HAI"));
</script>
</body>
```

O/p: true
false.

- is NaN (Not a Number):

Eg: <script>

document.write(isNaN(345));

document.write("
"+isNaN("HAI"));

</script>

</body>

O/p: false

true

- Number: is used to convert true/false to 0/1.

Eg: <script>

document.write(Number(Boolean(true)));

document.write(Number(Boolean(false)));

</script>

O/p: 1

0

- escape & unescape:

Eg: <script>

document.write(escape("Welcome to WT"));

document.write("
");

document.write(unescape("welcome%20to%20WT"));

</script>.

Control Flow Statements:

→ Sequential Control Statements - if, elsif, elsif ladder, switch cas

→ Loop " "

→ Jump " "

JavaScript program that reads a number from the user and checks whet given number is even or odd.

```
<html>
<script>
var a = prompaseInt(prompt("Enter a") );
if(a%.2 ==0)
    document.write("Even number");
else
    document.write("Odd number");
</script>.
```

Q: JavaScript to accept a number and generate the remainders in word format (accept number, divisor from user, if it is remainder then it should display four).

```
<html>
<script>
var a = parseInt( prompt ("Enter a number"));
var div = parseInt(prompt("Enter divisor value"));
if( a/div  var rem;
rem = a/div;
    document.write( toString(rem) );
var num = new Array("One", "two", "three" ---- "Nine");
</script>.
```

Loop Control Statements: while, dowhile, for loop.

Syn: inti
while(condt)
{ ------
inc/decre
}

Syn: initilization
do
{
----
incldec
} while(cond);

Syn:
for(initi; cond; inc/dec)
{
- - - -
- - - -
}

Q: JavaScipt to write a program on to calculate the individual digit of a number.

```
<html>
<script>
var a = paseInt(prompt("Enter a number"));
var rem, sum; sum = 0;
while(a != 0)
{
    rem = a % 10;
    sum = sum + rem;
    a = parseInt(a/10);
}
document.write("Sum of individual digits: " + sum);
</script>
```

Jump Control Statements:

- break

Eg:
```
for(i=0; i<5; i++)
{   if(i==2)
        break;
    document.write(i);
}
```
O/p: 0 1 2 3

- Continue:

Eg:
```
for(i=0; i<5; i++)
{   if(i==2)
        continue;
    document.write(i);
}
```
O/p: 0 1 3 4

Operators in JavaScript:

- Arithmetic Operators: +, -, *, /, %

  Eg: 45/10 = 4.5

- Increment & Decrement Operators: pre inc, pre dec, post inc, post dec

Eg: var a = 5;
var b = ++a;
O/p: b = 6

Eg: var a = 5
var b = a++;
O/p: b = 5

- Assignment Operator: =
- Shorthand Notations:

  Eg: a = a + 5

  Sh: a+ = 5

- Relational Operators: <, <=, >, >=, ==, !=
- Logical Operators: &&, ||, ~
- Conditional Operator: ? :
- HTML DOM objects:

  DOM: Document Object Models.

→ Window Object:

- document object
- navigator object
- screen object
- history object
- location object.

UNIT - III

3. WORKING WITH XML

Intro:

- XML stands for eXtensible MarkUp Language.
- XML is designed to transport and store the data.
- HTML is designed to display the data.
- XML is a markup language much like HTML.
- XML is designed to carry data, not to display data.
- XML tags are not predefined. You must define your own tags

Difference between XML & HTML:

- XML was designed to transport & store data, with focus on what data is.
- XML was created to structure, store and transport information
- HTML was designed to display data, with focus on how data loo

HTML — Predefined taps

XML — Usedefined tags.

Role of XML:

- Simplies Data Sharing
- Simplifies Data Transport
- Seperate Data from HTML.

XML Basics:

- XML Syntax
- XML declaration
- XML elements
- XML attributes
- Valid XML documents
- Viewing XML
- XML parser
- XML

XML Syntax:

- XML documents must have a starting tag and closing tag.
- XML tags are case Sensitive.
- XML tags must be properly nested.
- XML documents must have a root element and only ONE root eleme
- XML attribute values must be quoted.
- In XML, white space is preserved.

XML Declaration:

- <?xml version="1.0" encoding="UTF-8"?>
- XML declaration starts with <?xml and ends with ?>
- If it is included then it must include the version attribute. Rem

-ining attributes are optional.

- XML declaration must be at the begining of file.

XML elements:

- Elements can start with letters or the underscore, but not with numbers or other punctuation characters.
- After the 1st char, numbers are allowed, as the characters - and ..
- Element names can't contain spaces.
- Element names can't contain the : since it is the reserved word
- Element names can't start with xml letters.
- There cannot be a space after the opening < character. However there can be space before closing > character, if desired.

a) XML Example:

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

<employee>
    <firstname> HAI </firstname>

</employee>.
```

b) XML program on Book:

* Save the file with .xml extens

```
<?xml version="1.0" ?>

<book>
    <name> Data Base_And_Management_Systems </name>
    <author> ElmaSri </author>
    <pages> 898 </pages
</book>.
```

Viewing the XML document:

Double click on the particular file.

XML Attributes:

1) Syntax:

<tagname attributename = "value"> </tagname>

Eg: <firstname title = "HAI"> </firstname>.

2) Empty Elements:

Syn: EMPTYELEMENT: (i.e., without any tags).

<middlename />

Example program:

```
<?xml version = "1.0" ?>

<bookstore>
<book category = "cooking">
<title lang = "en"> Everyday Italian </title>
<author > Giada De Laurentiis </author>
<year > 2005 </year>
<price > 30.00 </price>
</book>

<book category = "children">
<title lang = "en" > Harry Potter </title>
<author > J. K. Rowling </author>
<year > 2005 </ year>
<price > 29.99 </price> </book. </bookstore>.
```

DTD: (Document Type Definition).

- It defines the legal building blocks of an XML document.
- A DTD defines the document structure with a list of legal elements and attributes.
- A DTD can be declared inside an XML document, or as an external reference.
- DTD is of two types: 1) Internal DTD
  2) External DTD.

External DTD:

Eg: Note.dtd

```
<!ELEMENT note (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from(#PCDATA)>
<!ELEMENT heading(#PCDATA)>
<!ELEMENT body (#PCDATA)>
<!ATTLIST to verified CDATA "yes">
<!ATTLIST body attach CDATA "yes">
```

XML Document:

```
<?xml version="1.0" ?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
   <to verified="yes">sree</to>
   <from>kavya</from>
```

<heading> Reminder </heading>

<body attach = "yes" > Dont forget me to meet this weekend! </body>

</note>.

Internal DTD:

```
<?xml version="1.0"?>
 <!DOCTYPE note [
 <!ELEMENT note (to, from, heading, body )>
 <!ELEMENT to (#PCDATA )>
 <!ELEMENT from(#PCDATA)>
 <!ELEMENT heading(#PCDATA)>
 <!ELEMENT body (#PCDATA)>
  ]>
 <note>
 <to>Tove</to>
 <from> KavyaSree </from>
 <heading> Reminder </heading>
 < body > Dont forget to meet the this weekend! </body>
 </note>.
```

Limitations of DTD:

- There is no concept of datatype.
- does not support Namespaces
- Defaults for elements cannot be specified.
- DTD are written in a strange (non-XML) format and are difficult to validate.

XHTML Document Structure:

- XHTML Doctype is mandatory.
- The xmlns attribute in <html> is mandatory.
- <html>, <head>, <title>, & <body> are mandatory.

XHTML Elements:

- Elements must be properly nested.
- Elements must always be closed.
- Elements must be in Lowercase.
- Documents must have one root element.

XHTML Attributes:

- Attributes must be in lower case.
- Attributes must be quoted.

Eg: <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head><title>
Title of document</title>
</head>
<body> some content
</body>
</html>.

Java API for XML Processing:

Packages:

- Javax.xml.transform
- Javax.xml.transform.dom
- Javax.xml.transform.sax
- Javax.xml.transform.stream.

XML Parser:

- It provides way how-to access or modify that data present in an XML document.
- Java provides multiple options to parse XML document.
- Following are various types of parsers which are commonly used to parse XML documents.

- **Dom Parser:** parses the document by loading the complete contents of the document and creating its complete hierarchial tree in memory
- **Sax Parser:** Parses the document on event based triggers. Does not load the complete document into the memory.
- **STAX Parser:** Parses the document in similar fashion to SAX parse but in more efficient way.

Difference between DOM & SAX :

- DOM parser loads whole xml document in memory while SAX only loads small part of XML file in memory.
- DOM parser is faster than SAX bcz it access whole XML document in memory.
- SAX is more suitable for large XML documents.
- We can insert & delete elements using DOM parser.

XSLT:

- XSL stands for Extensible Stylesheet Language, & is a style sheet language for XML documents.
- XSLT stands for XSL transformations. In this tutorial you will learn how to use XSLT to transform XML documents into other formats, like XHTML

CSS: Style Sheets for HTML:

- HTML uses predefined tags, & the meaning of each tag is well understood.
- The <table> tag in HTML defines a table - & a browser knows how to display it.
- Adding styles to HTML elements are simple. Telling a browser to display an element in a special font or color, is easy with CSS.

XSL = Style Sheets for XML

- XML does not use predefined tags (we can use any tag-names we like), & the meaning of each tag is not well understood.
- XSL describes how the XML document should be displayed!

XSL consists of 3 parts:

- XSLT - a language for transforming XML documents.
- XPath - a language for navigating in XML documents.
- XSL-FO - a language for formatting XML documents.

XSLT = XSL Transformations:

XSLT is the most important part of XSL.

XSLT is used to transform an XML document into another XML document, or another type of document that is recognised by a browser like HTML & XHTML. Normally XSLT does this by transforming each XML element into an(x) HTML element.

Root of XSL document: <xsl:stylesheet>

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="www.w3schools.com">
</xsl:stylesheet>.
```

The <xsl:template> Element.

. The <xsl:template> element is used to build templates.

, The match attribute is used to associate a template with an XML elemen

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="www.w3schools.com">
<xsl:template match="/">
</xsl:template>
</xsl:stylesheet>
```

The <xsl:value-of> Element

The <xsl:for-each> Element.

<xsl:sort> Element.

<xsl:if> Element.

<xsl:choose> Element..

UNIT - IV

UNIT - 4

Java Bean:

Java Bean is a java class that should follow following conventions:

- It should have a no-arg constructor.
- It should be Serializable.
- It should provide methods to get & set the values of properties, known as getter & setter methods.

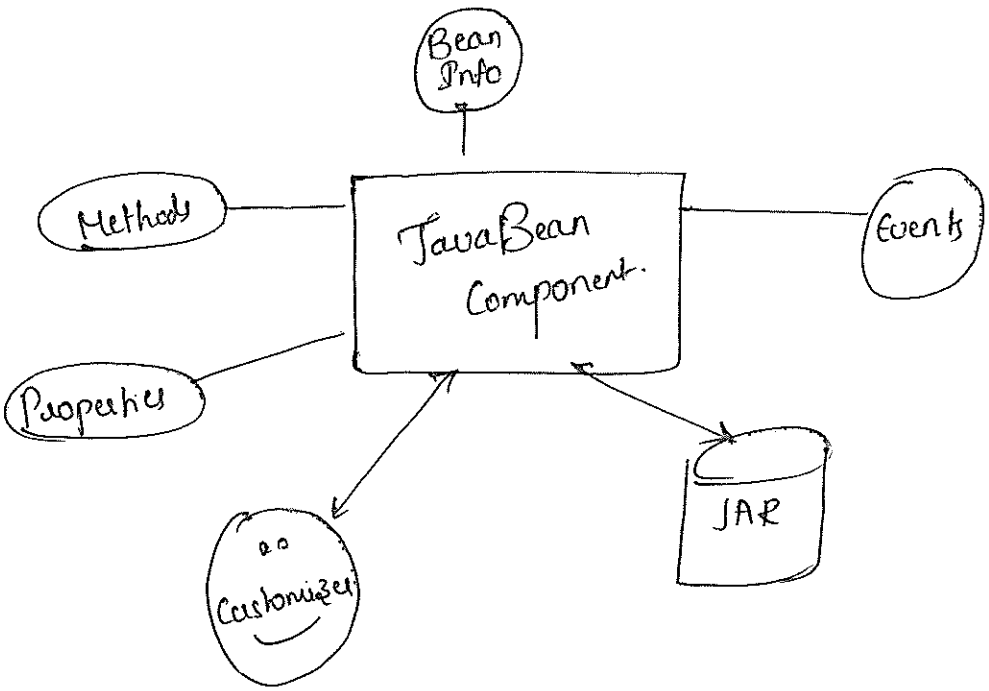
JavaBean Component:

- JavaBean is a reusable software component that can be manipulated visually in a builder tool.
- Graphic bean & Non-graphic bean.
- JavaBean is not distributed component like EJB.
- Interface of javabean is provided by..
  1. Design pattern(implicitly).
  2. Using a class to implement the BeanInfo or Customizer interfac

Advantages of Java Bean:

- Write once, run anywhere.
- The properties, events & methods of a bean that are exposed to an application builder tool can be controlled.
- They are the interface of the bean.
- They are plat-form independent.

- Configurations setting of a bean can be saved in persistent storage & restored later.
- Bean may register & receive events from other object & can generate event sent to other objects.



Design Pattern:

- All beans should implement the Serializable interface so that the state can be saved & later restored.
- Methods must be public.
- All exposed methods should be threadsafe, possibly synchronized to prevent more than one thread from calling method at a given time.
- Property X is exposed by public setX & getX methods.
- Boolean property may be exposed by is X method which returns a boolean value.
- The bean which may trigger event must provide addEventListener & removeEventListener methods for other bean to register with it to be notified

Deployment of Bean:

- All java classes can be converted to a bean.
- Beans is compressed & saved in the formal of jar file which contain manifest file, class files, gif files, & other information customization file
- Sun NetBeans, BDK, Visual Cafe, JBuilder, Visual Age are the bean builder tools.

Introspection:

- Process of analyzing a bean to determine the capability.
- Allows application builder tool to present info about a component to software designer.
- Naming convention implicit method.
- BeanInfo class to explicitly infer info of a bean.

Design pattern for Properties:

- Property is a subset of a bean's state which determines the appear -ce & behaviour of component.
  - Simple property.
  - Indexed property.
  - Bound property.
  - Constrained property.

Simple property:

- simple property has a single value.
- N is the name of the property & T is its type.

- public T getN();
- public void setN(T arg).
- For readonly property there is getN() method only.

Indexed property:

- One property may consist of multiple values stored in an array.
- public T getN(int index);
- public void setN(int index, T value);
- public T[] getN();
- public void setN(T values[]);

where N may be a double data[] & T is double.

Bound Property:

- It can generate an event when the property is changed.
- The event is of type PropertyChangeEvent & is sent to objects that previously registered an interest in receiving such notifications.
- bean with bound property - Event Source.
- Bean implementing listener - event target.

Constrained property:

- It generates an event when an attempt is made to change it value
- The event type is PropertyChangeEvent.
- The event is sent to objects that previously registered an interest in receiving an such notification.

- Those other objects have the ability to veto the proposed change.
- This allows a bean to operate differently according to runtime environme

Persistence:

- It has the ability to save a bean to storage & retrieve it at a later time.
- Configuration settings are saved.
- It is implemented by Java Serialization.
- If a bean inherits directly or indirectly from component class it is automatically Serializable.
- Transient keyword can be used to designate data members not be saved ex. Thread reference member.

Customizers:

- Property sheet may not be the best user interface for a complex component
- It can provide step-by-step wizard guide to use component.
- It can provide a GUI frame with image which visually tells what is changed such as radio button, check box, ----
- It can customize the appearance & behaviour of the properties.

Sample Program:

```
package colors;
  import java.awt.*;
import java.awt.event.*;
public class Colors extends Canvas
   transient private Color color;
```

```
private boolean rect;
public Colors() {
  addMouseListener(new MouseAdapter()
  { public void mousePressed(MouseEvent me) { change(); } });
  rect=false;
  setSize(200,100); change();
  }
  public boolean getRect() { return rect; }
  public void setRect(boolean flag) { this.rect=flag; repaint(); }
  public void change() { color=randomColor(); repaint(); }
  private Color randomColor()
  { int r=(int)(255*Math.random());
   int g=(int)(255*Math.random()); int b=(int)(255*Math.random());
   return new Color(r,g,b); }
   public void paint(Graphics g)
   {  Dimension d=getSize();
      int h=d.height;
      int w=d.width;
      g.setColor(color);
      if(rect) { g.fillRect(0,0,w-1,h-1); }
     else { g.fillOval(0,0,w-1,h-1); }
   }
}
```

Criteria to be a bean:

- Can this piece of code be used in more than one area?
- Can you quickly think of ways that this piece of code must be customized?
- Is this purpose of this code easy to explain?
- Does this code module contain all the info it needs to work itself?
- Does it have good encapsulation?

If you answer all "yes", You should make the class a bean.

JAR file:

- JAR file allows you to efficiently deploy a set of classes & their associated resources.
- JAR file makes it much easier to deliver, install & download. It is compressed.

Manifest file:

Manifest.tmp

Name: SimpleBean.class

Java-Bean: True

Creating & extracting a jar file:

- Create a jar file

  jar cfm simplebean.jar manifest.tmp *.class.

- Extracting files from a jar file.

  jar xf simplebean.jar.

Develop a New Bean:

- Create a directory for the new bean.
- Create the java bean source file.
- Compile the source file.
- Create a manifest file
- Generate a JAR file
- Start BDK.
- Test.

  Working-dir can be at <bdk>/demo where <bdk> is the installation dir for BDK.

UNIT - V

UNIT-5.

JDBC – Java to DataBase Connectivity.

Drawbacks of Files:

- Files may not be able to store large volumes of data.
- Files may not be able to store images (.JPEG, .GIF).
- On files, we are not able to apply complex conditions.
- Processing the data of multiple files is complex.
- Redundancy, Inconsistency, less Security

- JDBC API is one of the generic library developed by Sun Micro Systems & released to the industry to deal with any database without considering the vendors.
- For Eg: If C programmer wants to store the data permanently using MS Access databases, he must know the complete library of MS Access. If the same programmer wants to deal with the Oracle databases then he must know the complete library of Oracle DB, which is a complex task. Because, learning specific library for different DB's.
- At this stage, the industry programmers are expecting the library that deals with all DB's.
- Sun Micro Systems has developed a Generic library (or) Vendor Independent library known as JDBC API.

Vendor Specific library: (VSL)

If any library deals with particular database, that library is known as VSL or Native library.

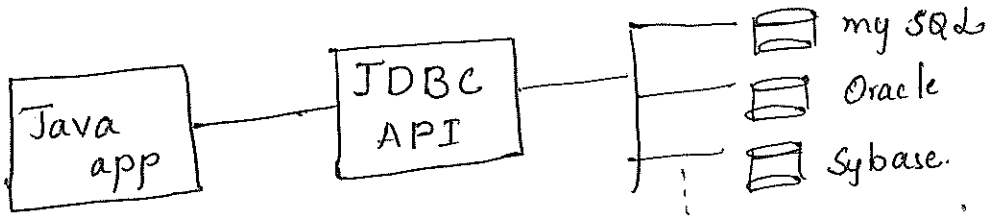
Generic Library: (GL)

If any library deals with all the database's, it is GL.

- JDBC API is one of the specification released by Sun Micro Systems to the industry. Bcz, of the more popularity of Sun Micro Systems many database vendors has taken the JDBC Spe-cification & started their implementation.
- Specification of JDBC API is a set of rules & guidelines in the form of interfaces (contains abstract methods).
- All the DB vendors has taken the interfaces of sun micro system developing classes with complete definition for abstract methods of interfaces.
- The classes developed by DB vendors are known as Vendor Specific Class. (VSC)

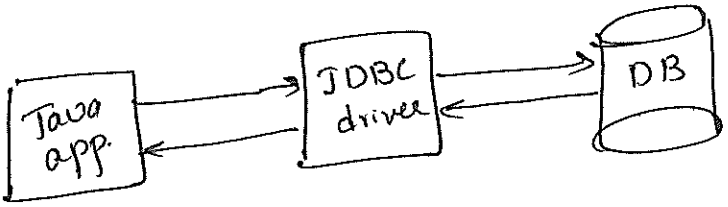
Learning all VSC by Java programmer is difficult. To eliminate, this difficulty SMS has collected all the vendor's Specific classes & prepared common classes for all the database vendors & released into the industry on the name of JDBC API.

C API - Set of functions
C++ API - Set of classes & functions
JAVA API - Set of classes & Interfaces



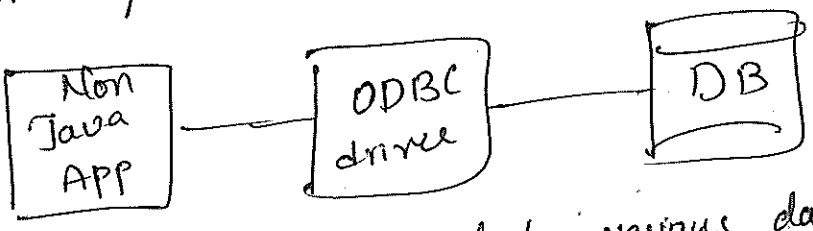
JDBC drivers:

JDBC driver acts as a interface between Java programmer and the back-end database.



ODBC driver:

Open DataBase Connectivity is developed by XOpen company & is a Specification. It is most suitable for non Java programmer to deal with any database.



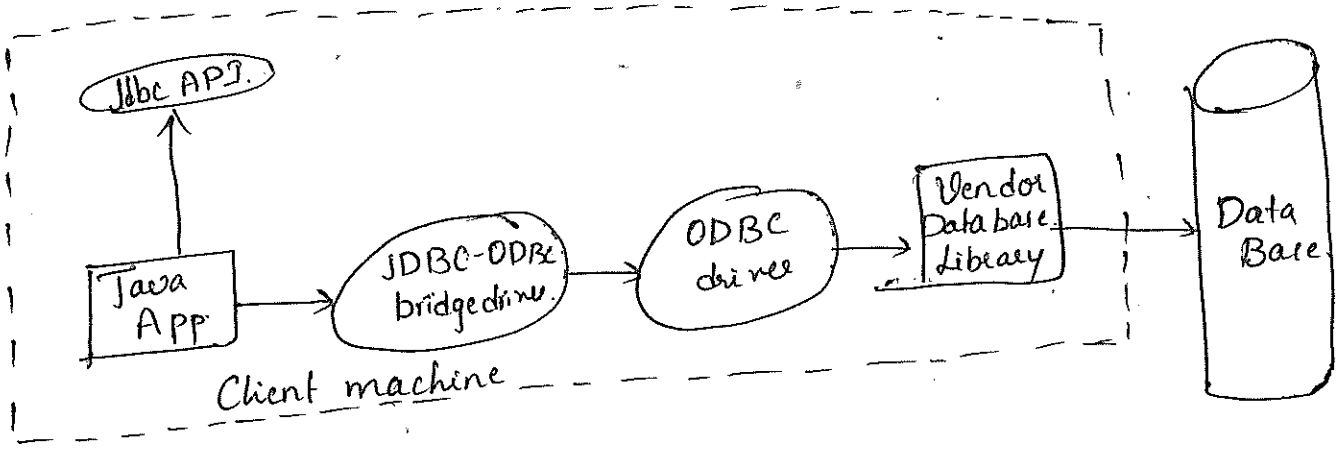
ODBC drivers are developed by various database developers & submitted to XOpen Company. XOpen Company tie up with the micro soft (All ODBC drivers are available in every Microsoft OS).

Note: All ODBC drivers are implemented by various DB vendors in C language.

Types of JDBC Drivers:

- Type1 driver (JDBC-ODBC bridge driver).
- Type2
- Type3
- Type4

Type 1:



Adv:

- easy to use
- can be easily connected to any database.

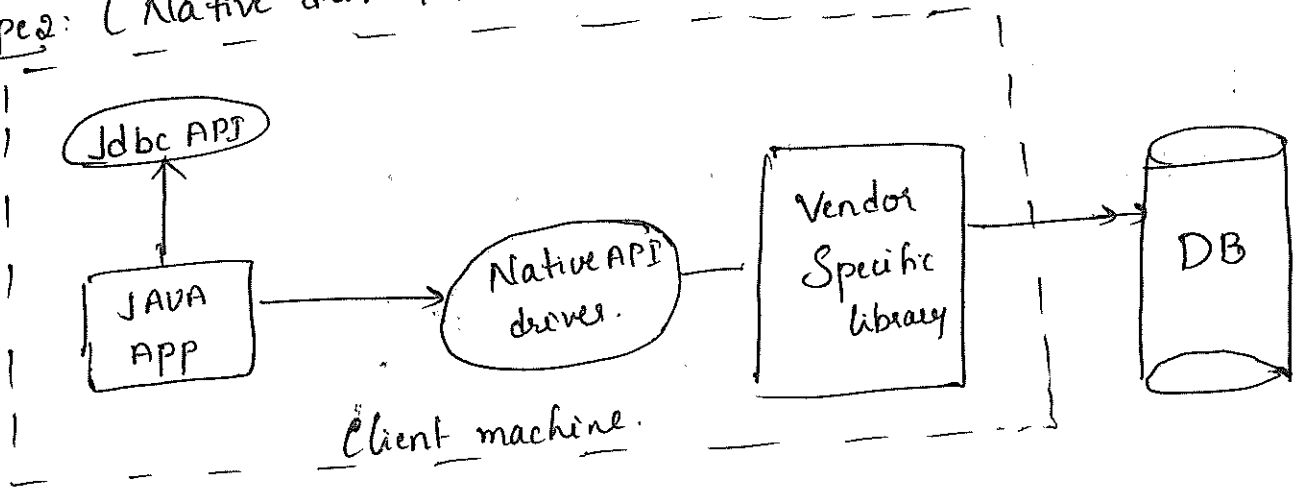
DisAdv:

- Performance degraded because JDBC method call is converted i

ODBC function calls.

- ODBC driver needs to be installed on the client machine.

Type 2: (Native driver / Partial Java Driver)



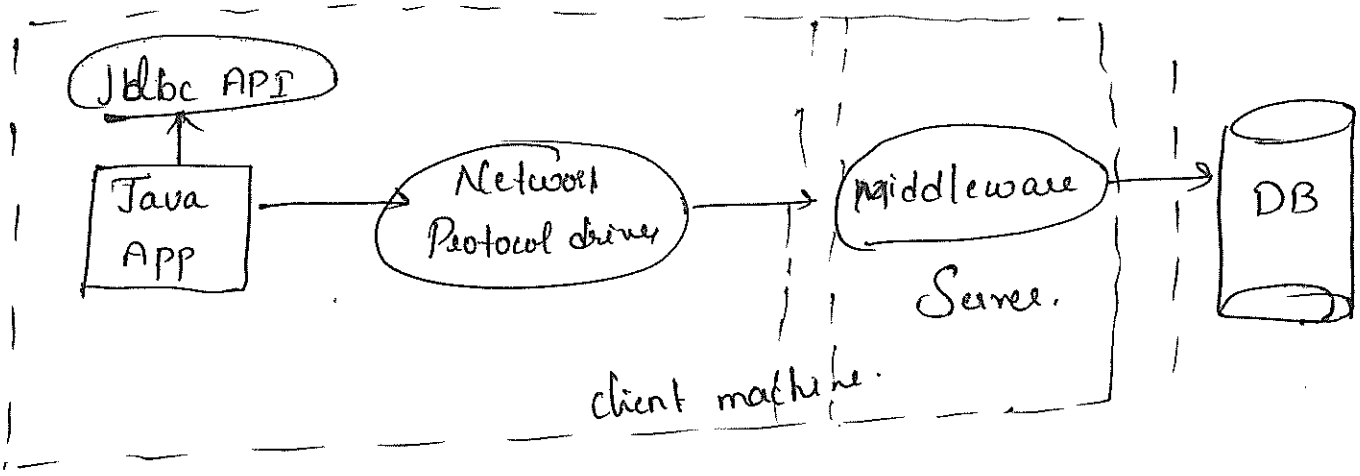
Adv:

- performance upgraded than Type-1 bridge driver.

DisAdv:

- Native driver needs to be installed on each client m/c.
- Vendor client library needs to be installed on client m/c.

Type 3: Net Protocol driver.



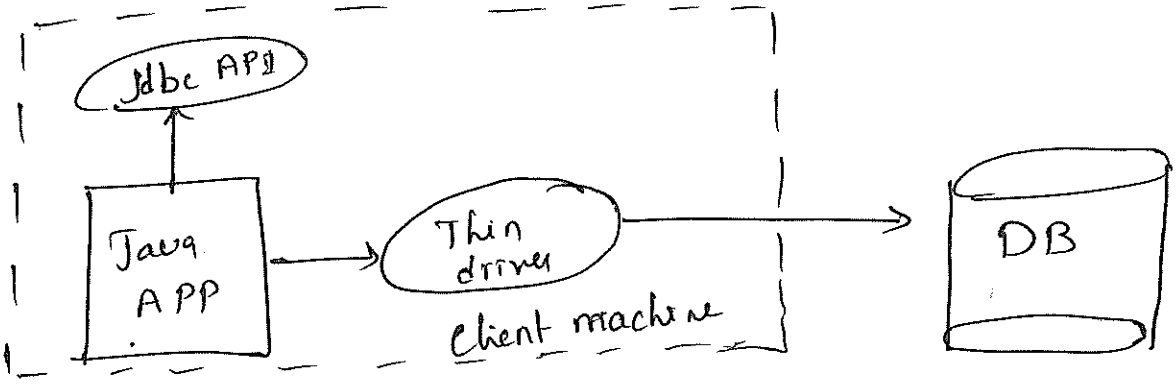
Adv:

- No client side lib is required bcz of app server may perform many tasks like load balancing, logging.

DisAdv:

- Network Support is required.
- Requires database Specific Coding to be done in the middle tier
- Maintanance is Costly.

Type 4: Thin driver / Pure driver.



Adv: • No s/w is required at client/server side.

• Better performance than previous drivers

DisAdv: Drivers depends on the Database.

Steps to develop a JDBC Application:

- Import packages.
- Load JDBC driver.
- Establish a connect to the database
- Create a Statement.
- Execute the Query
- Get the resultset
- Process the resultset.
- Close the connection.

Loading Driver: → at load time.

- DriverManager.registerDriver(new sun.jdbc.odbc.JdbcOdbcDriver())
- Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); → at run time

Establishing a Connection to the DataBase:

Connection con = DriverManger.getConnection("url+dsn", "un", "pas

url+dsn: jdbc:odbc:oradsn.

un: System

pwd: Oracle.

Creating the Statement: To send Queries to the database.

Statement st = con.createStatement();

Execute the Query:

st.executeUpdate("create table tablename (col1 datatype,
col2 datetype2)");

This is used for non selection Queries.

st.executeQuery("select *from abc");

This is used for selection Queries.

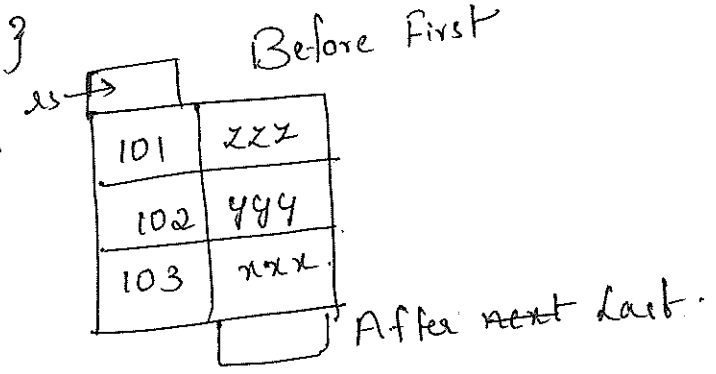
Get the Resultset:

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

Process the ResultSet:

```
while(rs.next())
{
    System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

Eg.



Close the Connection:

```
rs.close();
st.close();
con.close();
```

JDBC API Package:

Java.sql.* package.

- Classes
  - → DriverManager.

- Interfaces:
  - → Connection
  - → ResultSet
  - → Statement
  - →

JDBC program:

```
import java.sql.*;
class jdbc
{
    public static void main(String arg[]) throws IOException.
    {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        System.out.println("Driver loaded");
        Connection con = DriverManager.getConnection("jdbc:odbc:sec
         "system", "oracle");
         System.out.println("Connection Established");
         Statement st=con.createStatement();
         System.out.println("Statement Object is Created");
         st.executeUpdate("Create table Sample(no number, name
         varchar2(25), address varchar2(20))");
         System.out.println("Message Table created");
                                          values
         st.executeUpdate("insert into sample(1, 333, vjd))",
         System.out.println("1st row is inserted");
```

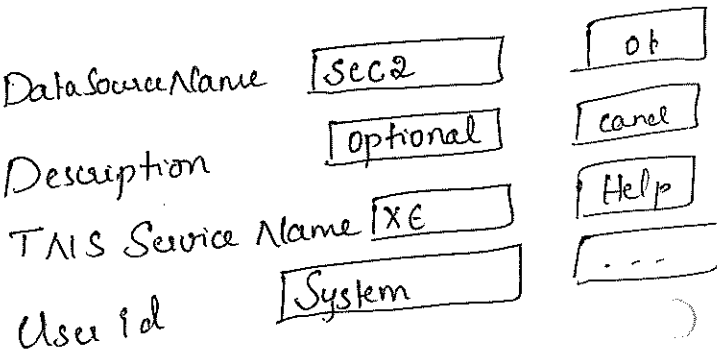
```
st.executeUpdate("insert into sample values(2, yyy, vijay));
System.out.println("2nd row is inserted");
ResultSet rs = st.executeQuery("select * from sample");
System.out.println("ResultSet is Generated");
while(rs.next())
{
System.out.println(rs.getInt(1) + "  " rs.getString(2)
   + "  " rs.getString(3));
}
rs.close();
st.close();
con.close();
}
}
```

Olp:

Driver Loaded.
Connection Established.
Statement Object is Created.
Table created
1st row is inserted
2nd row is inserted.
ResultSet is Generated.
1 xxx vjd.
2 yyy vijay.

How to Create dsn:

- go to Control panel
  ↳ Select Administrative tools.
- Select DataSources ODBC.
- Click on Add button.
- Select Oracle in XE
- Click on finish. ⟶ then



- Click on OK button.

JDBC program to create a table student with column names rollno, name, year, semester, marks.

```
import java.sql.*;
class jdbc1
{
    public static void main(String[] args) throws Exception
    {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver())
        System.out.println("Driver Loaded");
        Connection con = DriverManger.getConnection("jdbc:odbc
                                    student", "system", "oracle")
        System.out.println("Connection Established");
```

```
Statement st = con.createStatement();
st.executeUpdate("create table student(rollno number, name
varchar(20), year number, semester number, marks(number)");
System.out.println("Table created");
st.close();
con.close();
}
}
```

JDBC program to create a table student and insert two record into the table.

```
import java.sql.*;
class jdbc2
{
    public static void main(String[] args) throws Exception
    {
    class.forName("sun.jdbc.odbc.JdbcOdbcDriver()");
    Connection con = DriverManager.getConnection("jdbc:odbc:studen
                                    "system", "oracle");
    Statement st = con.CreateStatement();
    st.executeUpdate("create table student1(roll no number, name
    varchar2(20)");
    st.executeUpdate("insert into student1 values(A3", "kavyaSree");
    st.executeUpdate("insert into student1 values(56, "Yaswanth");
```

```
System.out.println("2 records inserted"); while(
 rs.close();
st.close();
con.close();
 }
}
                    ResultSet rs = st.executeQuery("select * from student1
                    while(rs.next())
                    {
                        S.O.P ln("rs.getInt(1)+" " rs.getString(2));
                    }
```

JDBC program to delete the row with rollno 101.

```
import java.sql.*;
class jdbc2;
{
    public static void main(string[] args) throws Exception
    {
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver()");
    Connection con = DriverManager.getConnection("jdbc:odbc:
                                    student","system","oracle");

    Statement st = con.createStatement();
    St.executeUpdate("insert into student1 values(101,"aaa")"
    string st.executeUpdate("Delete from student1"+"where rollname
                                                    =101");
    Resultset rs = st.executeQuery("sleet * from student1");
    while(rs.next())
    { SO.Pln("rs.getInt(1)+" "rs.getString(2));
    }
} } rs.close(); st.close(); con.close();
```

JDBC program to create a table using Type-4 driver.

```
import java.sql.*;
class jdbc4
{
    public static void main(String[] args) throws Exception
    Class.forName("oracle.jdbc.driver.oracleDriver");
    S.O.Pln("Driver Loaded");
    Connection con = DriverManager.getConnection("jdbc:oracle:
    thin:@localhost:1521:XE", "system", "oracle");
                                  ↳ service id of oracle.
    Statement st = con.createStatement();
    st.executeUpdate("create table abc(id number, name varchar2(
    SOPln("Table Created");
    st.executeUpdate("insert into abc(1, "MaryaSree");
    st.executeUpdate("insert into abc(2, "Banda");
    ResultSet rs = st.executeQuery("select * from abc");
    while(rs.next())
    {
        SOPln(rs.getInt(1) " " rs.getString(2));
    }
    rs.close();
    st.close();
}} con.close();
```

- Type-4 driver is also known as Pure Java / Thin driver.
- This driver will come automatically when we install oracle s/w.
- This driver is available in the form of 'jar' files. (jar - Java Archieve
  → (ojdbc14.jar file)

Location of Type4 driver:

C:
↓
oraclexe
↓
app
↓
oracle
↓
product
↓
10.2.0.
↓
server
↳jdbc
↳lib → ojdbc14.jar file ← Type4 driver.

- Copy ojdbc14.jar file
- Create a folder path in D drive and paste in D drive.

How to compile the program:

javac filename.java -cp d:\path\ojdbc14.jar

How to execute the program:

java -cp .;d:\path\ojdbc14.jar filename.

. → represents current directory

How to know the Service id & Service name of Oracle?

C:
↳ oracle XE
↳ app
↳ oracle
↳ product
↳ 10.2.0
↳ Server
↳ Network
↳ ADMIN
↳ tnsnames.ora

Q: JDBC program to update the student record by reading rollno from the user.

```
import java.util.*;
import java.sql.*;
import java.io.*;
class jdbc5
{
    public static void main(String[]args)throws Exception.

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

    S.O.Pln("Driver Loaded");

    Connection con= DriverManager.getConnection("jdbc:odbc:sree
"system", "oracle");

    /*Scanner s = new Scanner(system.in);

    String st = s.next(); */

    Statement st = con.createStatement();

    S.O.Pln("Statement Object Created");

    BufferedReader dis = new BufferedReader(new InputStreamRead
                                            (System.in));
```

```
S.O.Pln("Enter no to update");
int sno = Integer.parseInt(dis.readLine());
st.executeUpdate("create table abc:
String query = "update sample set name = 'aaa'" +
                "where no = " + sno;
int res = st.executeUpdate(query);
if(res == 1)
{
    S.O.Pln("one row Updated");
}
else
{
    S.O.Pln("failure");
}
st.close();
con.close()
}
}
```

Dynamic Queries:

- Prepared Statement:

. In previous applications, the queries are executed by sending throu Statement interface object.

. Any type of query which we send to the database by using Statement interface object are known as Static Queries.

. When we submit the static Query to the database those Queries wi participate in 3 phases of Database.

→ parsing
→ Compilation
→ Execution.

. All the queries are repeatedly participating in parsing & compila phases, which is not a recommended process and finally, the performance of jdbc application is reduced.

. In order to eliminate this drawback, we must use the concept of dynamic Queries.

. A dynamic Query is the one which always participates in one tim passing and one time compilation phases. But, multiple times it execu

. Because of this the performance of JDBC application is improv

. In order to execute the dynamic Queries, we use a predefined interface i.e., PreparedStatement.

Statement	PreparedStatement.
• It is one of the super Inteface for PreparedStatement.	• It is one of the sub interface of Statement.
• An object of Statement interface is always used for executing static Queries.	• An object of PreparedStatement interface is always used for executing Dynamic Queries.
• 1 Static Query - (1 Parse, 1 Compile, 1 execute).	• 1 Dynamic Query - (1 Parse, 1 compile, more executions).
• When we are executing multiple queries at diff times, it is recommended to execute such queries with Statement interface Object.	• When we are executing same type of queries at multiple times, either at same time or diff time, it is recommended to use Object of PreparedStatement interface.

Program on Dynamic Query:

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

class jdbc6
{
    public static void main(String args[]) throws Exception.
    {
        Connection con=null;
        PreparedStatement ps=null;
```

```
Class.forName("oracle.jdbc.driver.OracleDriver");
con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521
XE", "system", "oracle");
String query = "insert into sam values(?,?,?)";
ps = con.prepareStatement(query);
BufferedReader dis = new BufferedReader(new InputStreamReader
(System.in));
S.O.Pln("Enter number of students");
int n = Integer.parseInt(dis.readLine());
for(int i=1; i<=n; i++)
{
    S.O.Pln("Enter " + i + "student number");
    int no = Integer.parseInt(dis.readLine());
    S.O.Pln("Enter student name");
    String name = dis.readLine();
    S.O.Pln("Enter address");
    String add = dis.readLine();
    ps.setInt(1, no);
    ps.setString(2, name);
    ps.setString(3, add);
    int res;
    res = ps.executeUpdate();
```

```
if(res==1)
    S.O.Pln("success");
else
    S.O.Pln("failure");
} // for closing  con.close(); ps.close();
} // main closing
} // class closing.
```

** classes111.jar is the type4 driver for Oracle 8i/9i

Syn:

String Query = "insert into tablename values(?, ?, ?)";

The name of the '?' in the jdbc is Dynamic Substitute Operator)

Positional place holder.

Callable Statement interface:

It is used to invoke or call stored procedures in DB.

Syn: create or replace procedure procedurename (parameters) as

```
begin
. . .
end;
```

Eg: create or replace procedure first (x in number, y out number) as

```
begin
y:= x * x;
end;
```

Sample program on Callable Statement:

```
import java.sql.*;
class callstmt
{
    PSVM(String[] args) throws Exception.
    {
        class.forName("oracle.jdbc.driver.OracleDriver");
        Connection con = DriverManager.getConnection
("jdbc:oracle:thin:@localhost:1521:XE", "system", "oracle");
        CallableStatement cst = con.prepareCall("{call first(?,?)}")
    cst.registerOutParameter(2, Types.INETEGER);
     cst.setInt(1,20);
     cst.execute();
     int res = cst.getInt(2);
     S.O.Pln("Result is ::"+res);
     cst.close();
     con.close();
    }
}
```

Op: Result is :: 400.

for getting the columname of a Table:

```
ResultSet rs = st.executeQuery("select * from sample");
ResultSetMetadata rsmd = rs.getMetaData(0);
String name = rsmd.getColumnName(3);
  SOPln(name);   // Address.
```

UNIT – VI

6. WORKING WITH SERVLETS.

Web Based Application:

- A web-based application is any application that uses a website as the interface.
- Users access the application from any computer connected to the internet using a standard browser, instead of using an application that has been installed on their local computer.

Advantages:

- Web apps avoid the burden in deploying the client computer.
- Platform independent.
- Can run on any of the platforms OS
- Updates are easier
- Version checking is not necessary.
- Adaptability of mobile applications.
- No administrator permissions.
- Makes bug fixes are easier.
- Support & maintanance is easier.

Pro's of a web app over desktop app:

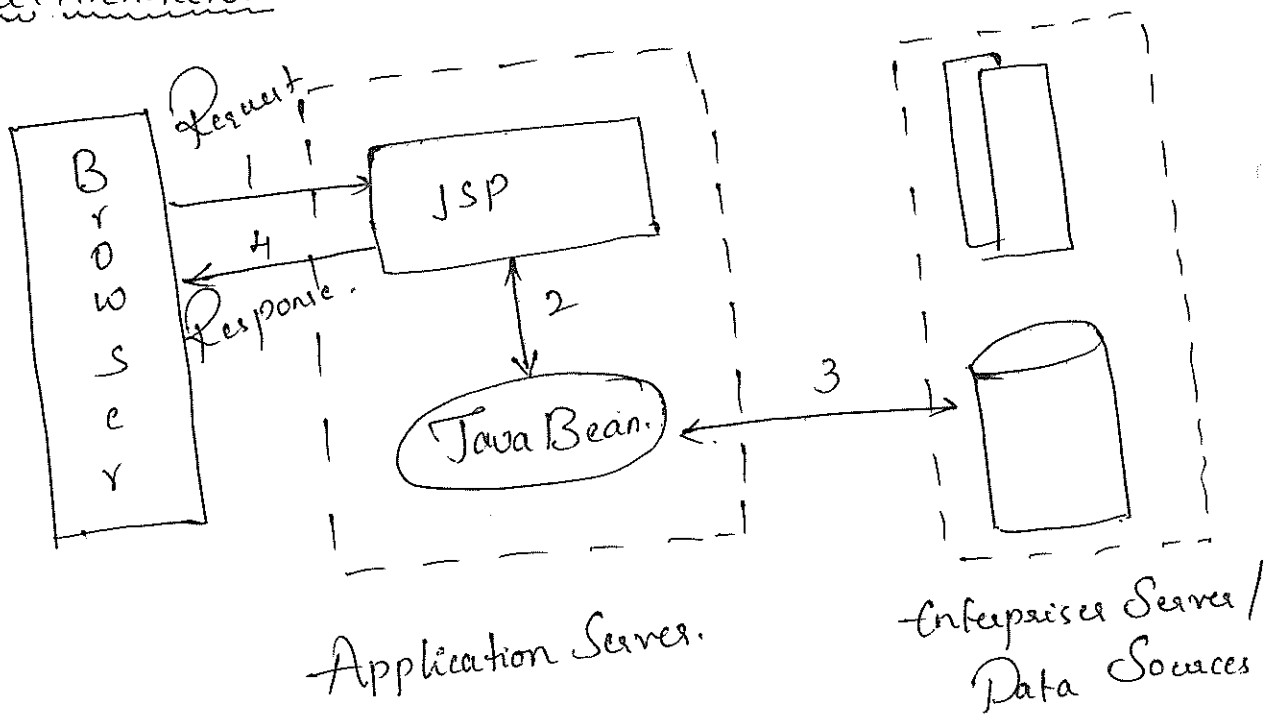
- Only one copy of a program will ever need to be updated.
- 99% of the code is platform independent.
- maintainance, support & patches are easier to provide.

- Less chance of finding restrictions that the clients computer may have imposed.

Web Architecture Models:

- Model1 Architecture
- Model2 (MVC) Architecture.

Model1 Architecture:



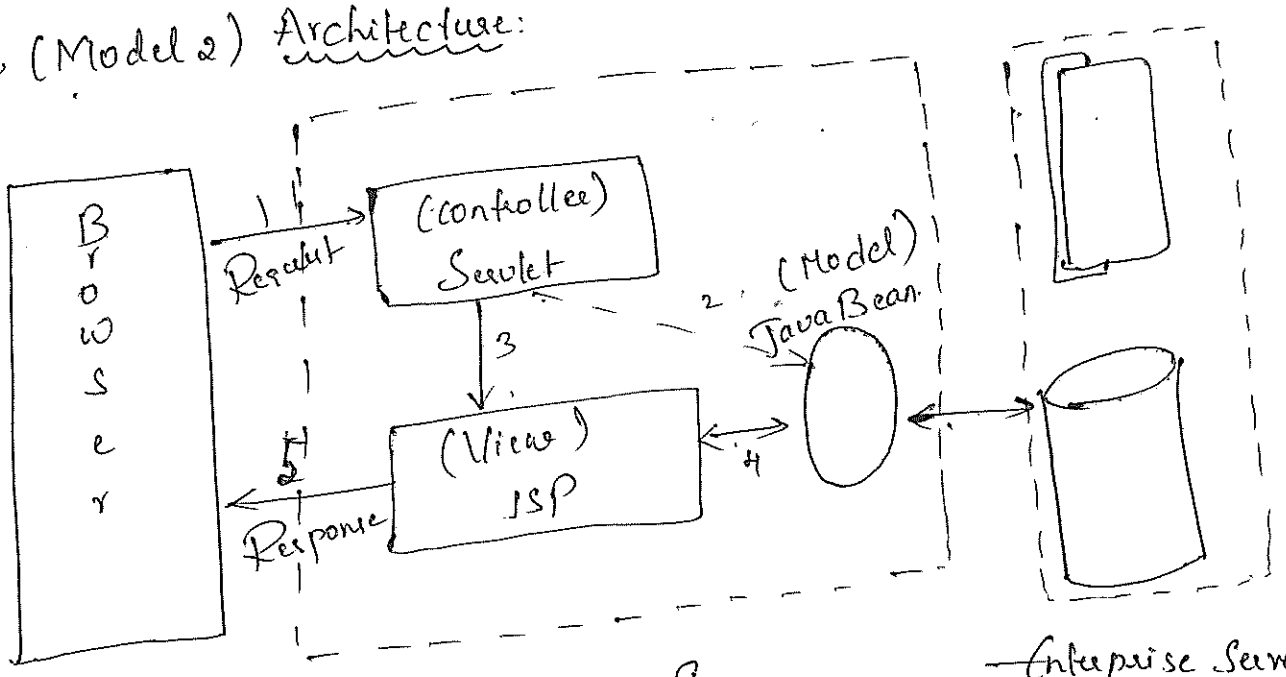
Process Flow:

- Browser sends the request for the JSP Page.
- JSP accesses Java Bean & invokes Business logic.
- JavaBean connects to the database and get/save data.
- Response is sent to the browser which is generated by JSP.

- Servlet & JSP are the main technologies to develop the web applications.

- Problem in Servlet Technology Servlet needs to recompile if any designi code is modified.
- It doesn't provide seperation of concern. Presentation & Business log are mixed up.
- JSP overcomes almost all the problems of Servlet.
- It provides better seperation of concern, now presentation & busines logic can be easily seperated.
- You don't need to redeploy the application if JSP page is modifi
- You don't need to develop web application using Java Bean, cus
- JSP provides support to develop web application using Java Bean, cus tags & JSTL so that we can put the business logic seperate fro our JSP that will be easier to test & debug.

MVC (Model 2) Architecture:



* Model - Business logic - Java Bean.
* View - Presentation logic - JSP
* Controller - Request Processing logic - Servlets.

Web Server:

Apache Tomcat webserver:

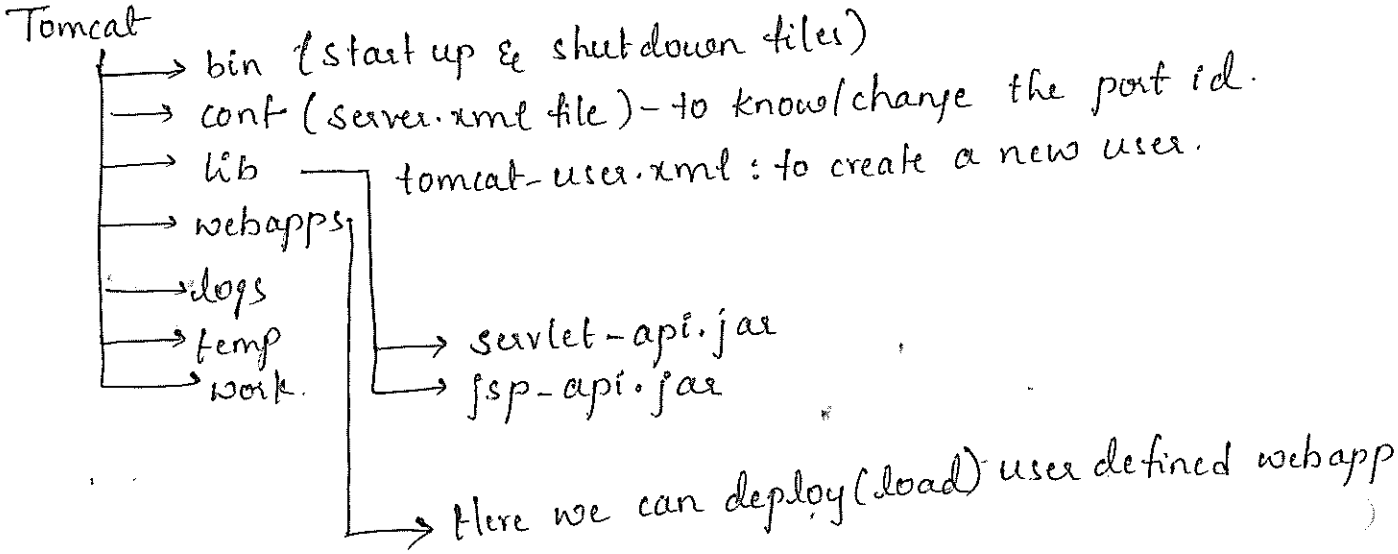
Servlet container which resides in Apache Tomcat is CATALINA & JSP Container is JASPER.

There are 2 types apache tomcat webservers available in tomcat websi

→ coreversion (.rar file).

→ installable version (.exe)

Directory structure: C:↳ programfiles
↳ Apache slw foundation
↳ Tomcat 6.0



How to start the apache tomcat webserver:

Start ↳ All programs
↳ Apache tomcat 6.0
↳ configure tomcat
Click on that then

displays a window & then click on start.

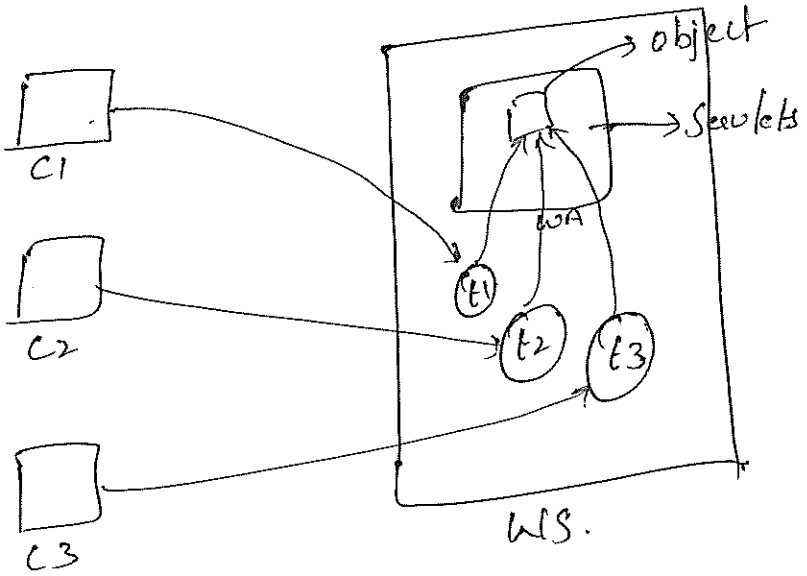
How to give request to the webserver:

http://localhost:2000

↓ Ip addr ↓ Port number.

Servlets:

. A Servlet is a Single Instance, Multiple Thread Technolo



What does Servlet do?

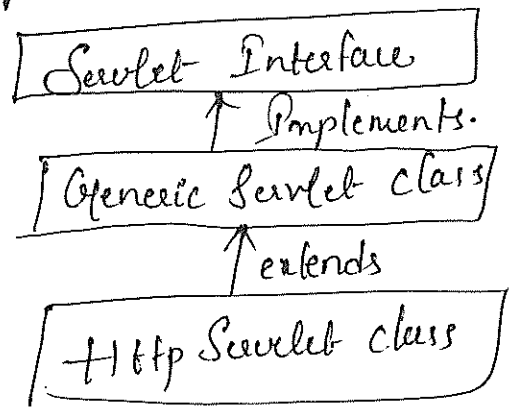
. It receives the client request (http request).

. Extract some info from the request.

. Do content generation or business logic process.

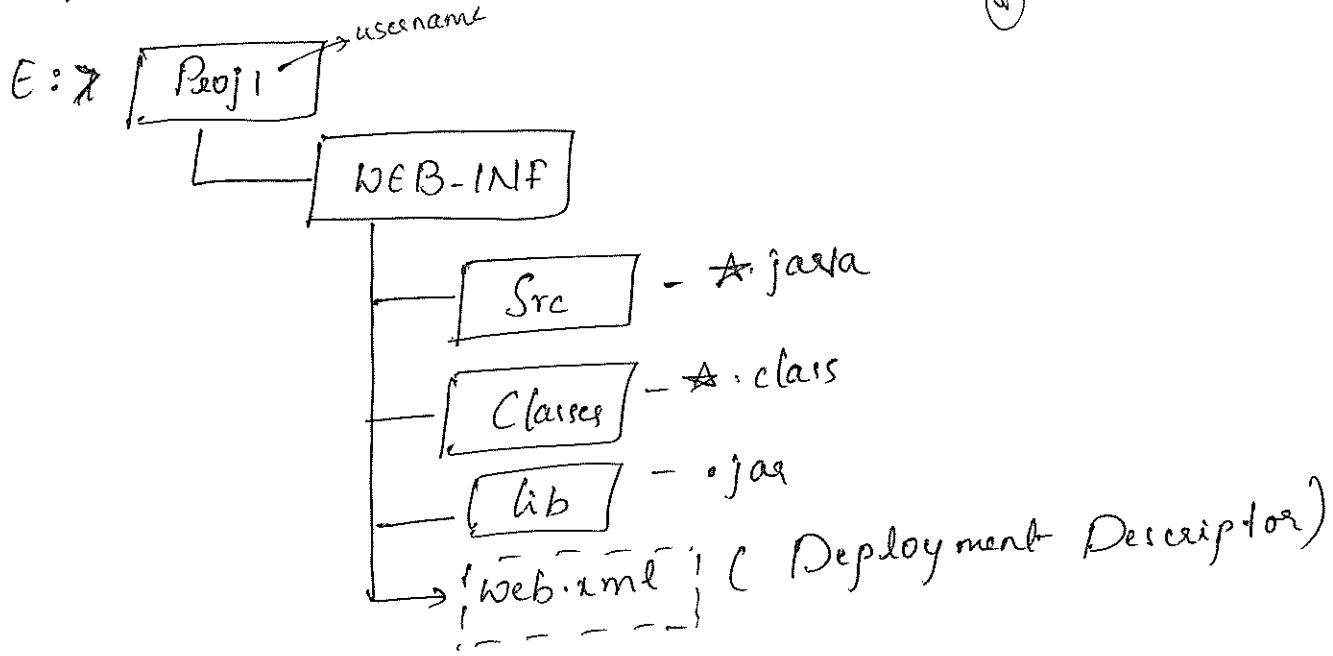
. Sends the response back to client.
create a

There are 3 ways to create a servelet application.



- Any servlet class should implement servlet interface (Javax.servlet.Servlet).
- Methods present in this interface are:
  - → public void init()
  - → public void service()
  - → public void destroy()
  - → public void String getServletInfo()
  - → public ServletConfig getServletConfig().
- When an implementing servlet interface, we must overwrite all the five methods of that interface.
- But, most of the developers may be interested to provide def. th for request processing method i.e., service method but, not the other methods.
- *** Generic Servlet is an abstract class & implements servlet interface overwrites the methods of all servlet interface except service meth
- Problems with Generic Servlet are No state management, no sess -on management.
- HTTP Servlet is a subclass of a Generic Servlet and it is also an abstract class.
- Most of the developers prefers extending HTTP Servlet class because, this is protocall specific class & we can do both the session, state managements.

Directory Structure of Webapplication:

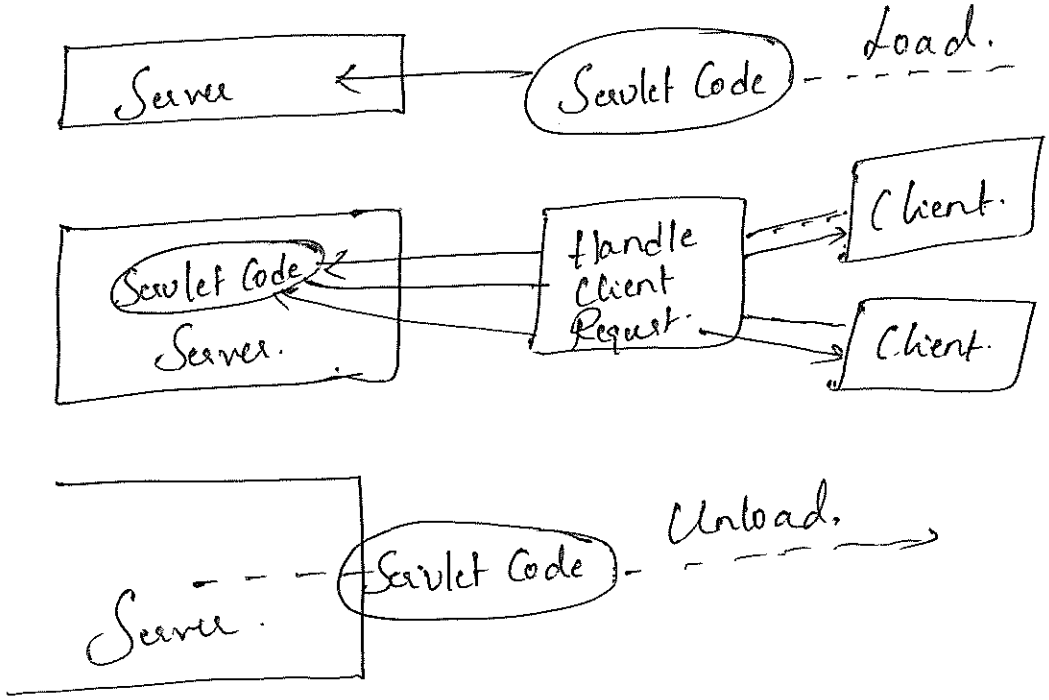


Servlet Life Cycle:

- **init:**
  - Executed once when the servlet is first loaded, or at server start. Not called for each request.
- **Service:**
  - called in a new thread by server for each request.

  Dispatches to doGet, doPost, etc...

  Don't override this method!
- **doGet, doPost, doXxx**
  - Handles GET, POST, etc.. requests.
  - Overrides these methods to provide desired behavior.
- **destroy.**
  - Called when server deletes servlet interface. Not called after each request.



Developing a Servlet Application:

```
import java.io.*;
import javax.servlet.*;
public class First implements Servlet {
ServletConfig config=null;
public void init(ServletConfig config)
{   this.config=config;
    System.out.println("Servlet is initialized");
}
public void service(ServletRequest req, ServletResponse res)throws
                        IOException, ServletException{
{
     res.setContentType("text/html");
     PrintWriter out=res.getWriter();
     out.print("<html><body>");
     out.print("<b>hello simple servlet</b>");
     out.print("</body></html>");
}
public void destroy()
{   System.out.println("Servlet is destroyed");
}
public ServletConfig getServletConfig()
{       return config;
}
```

```
public String getServletInfo()
{   return "copyright 2007-2010";
}
```

web.xml file:

```
<web-app>
<servlet>
<servlet-name>abc</servlet-name>
<servlet-class>First</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>abc</servlet-name>
<url-pattern>/Welcome</url-pattern>
</servlet-mapping>
</web-app>.
```

Directory structure:

E:> proj1 > WEB-INF

- classes (.class files).
- lib (.jar files).
- src (.java files)
- web.xml.

** Copy the web application to the web apps folder of tomcat directory. & then start the server. & then give the request from client as.

http://localhost:2000/servletproj2/hello

Generic Servlet:

```
import java.io.*;
import javax.servlet.*;
public class Second extends GenericServlet
{ public void service (ServletRequest req, ServletResponse res) throws
                    IOException, ServletException
  {
      res.setContentType ("text/html");
     PrintWriter out = res.getWriter();
     out.print ("<html><body>");
     out.print ("<b> hello generic servlet</b>");
     out.print ("</body></html>");

  }
}
```

How to compile & execute: copy servlet-api.jar file from tomcat -

* javac -cp : d:\path\servlet-api.jar filename.java. // will create .class fi

* java -cp d:\path\servlet-api.jar filename.

- installation directory (: c/ program files / AFS / Tomcat 6.0 (lib) & paste into the path-folder of d drive.

UNIT – VII

HTTP Servlet Class:

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class Third extends HttpServlet
{
  public void doGet(HttpServletRequest req, HttpServletResponse
                                                    res)
  res) throws ServletException, IOException.
  {
    res.setContentType("text/html"); //setting content type
    PrintWriter pw = res.getWriter(); // get the stream to write d

  //writing html in the stream
  pw.println("<html><body>");
  pw.println("Welcome to Servlet");
  pw.println("</body></html>");
  pw.close(); // closing stream
  }
}
```

Working with Initialising Parameters:

- An object of ServletConfig is created by the web container for each servlet.
- this object is used to get configuration information from web.xml file.

Advantages:

If the Configuration information is modified from the web.xml file, we don't need to change the servlet. So it is easier, to manage the web application, if any specific content is modified from time to time.

Initializing Servlets:

- Common in real-life Servlets
  - E.g: initializing database connection pools.
- Use ServletConfig.getInitParameter to read initialization parameters
  - Call getServletConfig to Obtain the ServletConfig Object.
- Set Init Parameters in web.xml (ver 2.2/2.3)
- It is common to use init even when you don't ~~read~~ read. init parameters.

Eg program for initializing Parameters:

```
import java.io.*;
import javax.servlet.http.*;
public class DemoServlet extends HttpServlet
{
 public void doGet(HttpServletRequest request, HttpServletRespo
 se response) throws ServletException, IOException
 {
   response.setContentType("text/html");
   PrintWriter out = response.getWriter();
   ServletConfig config = getServletConfig();
   String driver = config.getInitParameter("driver");
   out.println("Driver is : " + driver);
   out.close();
 }
}
```

web.xml:

```
<web-app>
  <servlet>
  <servlet-name>DemoServlet</servlet-name>
  <servlet-class>DemoServlet</servlet-class>
  <init-param>
  <param-name>driver</param-name>
  <param-value>sun.jdbc.odbc.JdbcOdbcDriver
  </param-value>
```

</init-param>

</servlet>

</servlet-mapping>

<servlet-name>DemoServlet</servlet-name>

<url-pattern>/servlet</url-pattern>

</servlet-mapping>

</web-app>

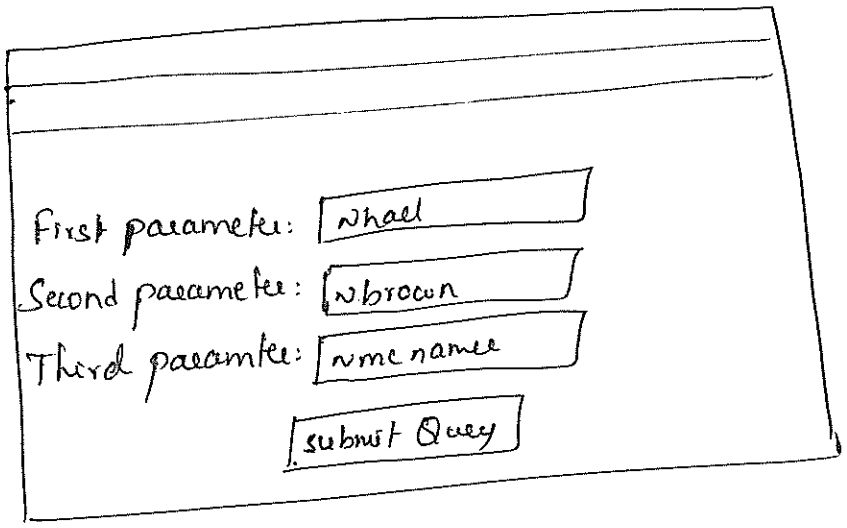
Reading Form Data: / Servlet Request Interface

- getParameter("name")
  - Returns value as user entered it. i.e., URL-decoded value of first occurrence of name in query string.
  - works identically for GET & POST requests.
  - Returns null if no such parameter is in query.
- getParameterValues("name")
  - Returns an array of the URL-decoded values of all the occurrences of name in query string.
  - Returns a one-element array if param not repeated.
  - Returns null if no such parameter is in query.
- getParameterNames()
  - Returns Enumeration of request params.

HTML form with 3 parameters:

```
<FORM ACTION = "/servlet/cwp.ThreeParams">
First Parameter : <INPUT TYPE = "TEXT" NAME = "param1"><BR>
Second Parameter: INPUT TYPE = "TEXT" NAME = "param2"><BR>
Third parameter: INPUT TYPE = "TEXT" NAME = "param3"><BR>
<CENTER><INPUT TYPE = "SUBMIT"></CENTER>
</FORM>
```

O/p:



Prg:

```
public class ThreeParams extends HttpServlet
{  public void doGet(HttpServletRequest request, HttpServletResp
                     response) throws ServletException, IOExceptio

   {  response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String title = "Reading Three Request Parameters";
      out.println(ServletUtilities.headWithTitle(title) +
```

```
"<BODY BGCOLOR=\"#FDF5E6\">\n" + "<H1 ALIGN=CENTER>
+title + "</H1>\n" + "<UL>\n" +
"<LI><B>param1</B>: "+request.getParameter("param1") + "\n"
"<LI><B>param2</B>: "+request.getParameter("param2") + "\n" +
"<LI><B>param3</B>: "+request.getParameter("param3") + "\n" +
"</UL>\n" + "</BODY></HTML>");
}
}.
```

Servlet Request Dispatcher:

Purpose:

- Forwards a request from one servlet to another.
- Have first servlet do some of the work & then pass on to another.
- Can even forward on to a static source like html.

Request Dispatcher:

- The RequestDispatcher is a Object is used to send client request to any resource on the server.
- Such a response resource may be dynamic (eg. a Servlet or a JSP file) or static (e.g., a HTML document).

To Send a request to a resource X, use:

getServletContext().getRequestDispatcher("X");

Request Dispatcher Methods:

- void forward(ServletRequest request, ServletResponse response)
  → Forwards a request from a servlet to another resource.
- void include(ServletRequest request, ServletResponse response)
  → includes the content of a resource in the current response.

Passing on Data:

- 3 different ways to pass parameters for the forwarded Servle
or JSP.
  - Data that will be used only for this request:
    request.setAttribute("key", value);
  - Data that will be used for this client (also for future reques
    session.setAttribute("key", value);
  - Data that will be used in the future for every client.
    context.setAttribute("key", value);

Example: (Login.java)

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

```
public class login extends HttpServlet
{
  public void doPost(HttpServletRequest request, HttpServletResponse
   response) throws ServletException, IOException
   {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

    String n = request.getParameter("userName");
    String p = request.getParameter("userPass");
    if(p.equals("servlet"))
    {  RequestDispatcher rd = request.getRequestDispatcher("servlet2");
       rd.forward(request, response);
    }
    else
    {
      out.println("Sorry UserName or Password Error!");
      RequestDispatcher rd = request.getRequestDispatcher("index.h
                                                          tml");
      rd.include(request, response);
    }
}
```

Example : (WelcomeServlet.java)

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

```
public class WelcomeServlet extends HttpServlet {
    public void doPost (HttpServletRequest request, HttpServletResponse
                        response) throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String n = request.getParameter("userName");
        out.print("Welcome "+n);
    }
}
```

Describing Request Headers:

- General-purpose way
  - getHeader, getHeaders, getHeaderNames.
- Specialized - commonly used headers
  - getCookies
  - getAuthType and getRemoteUser
  - getContentLength, getContentType.
  - getDateHeader, getIntHeader.
- Related info - main request line.
  - getMethod, getRequestURI, getProtocol.

HTTP Request Headers:

- Accept
  - Indicates MIME types browser can handle.
  - can send different content to different clients.
- Accept-Encoding
  - Indicates encodings browser can handle. (eg: gzip).
- Authorization:
  - User identification for password-protected pages.
  - Instead of HTTP authorization, use HTML forms to send username password. Store in session object.
- Connection:
  - In HTTP 1.0, keep-alive means browser can handle persistent connection. In HTTP 1.1, persistent connection is default. Persistent connection means that the server can reuse the same socket over again for requests very close together from the same client.
  - Servlets can't do this unilaterally; the best they can do is to give the server enough info to permit persistent connections. So, they should set content length with setContentLength.
- Cookie:
  - Gives cookies previously sent to client.
- Host:
  - Indicates host given in original URL.
  - This is a required header in HTTP 1.1. This fact is important

to know if you write a custom HTTP client or telnet to a server & use the HTTP 1.1 version.

- If-Modified-Since.
  - Indicates client wants page only if it has been changed after specified date.
  - Don't handle this situation directly.
- Referer
  - URL of referring web page
  - useful for tracking traffic; logged by many servers.
  - can be easily spoofed.
- User-Agent:
  - String identifying the browser making the request.
  - Use sparingly.
  - Again, can be easily spoofed.

HTTP Status Codes:

- Changing the status code lets you perform a number of tasks not otherwise possible.
  - Forward client to another page
  - Indicate a missing resource.
  - Instruct browser to use cached copy.

- set status before sending document.

SC General Categories.

- 100-199
  - Indicate the client should respond with some other actions.
- 200-299
  - Indicate the request was successful.
- 300-399
  - Usually include a location header.
- 400-499
  - Indicate an error by client.
- 500-599
  - Indicate an error by the server.
- 200 (OK)
  - Everything is fine; document follows.
  - Default for servlets.
- 204 (No content)
  - Browser should keep displaying (various) previous document, no new document is available.
- 301 (Moved Permanently)
  - Requested document permanently moved elsewhere.
  - Browsers go to new location automatically.

HTTP Response Headers:

purposes:

- Give forwarding location
- Specify cookies.
- Supply the page modification date.
- Instruct the browser to reload the page after a designated interva
- Give the document size, so that persistent HTTP connections can b used.
- Designate the type of document being generated. - Etc...

Common HTTP 1.1 Response Headers:

- Cache-Control (1.1) & Pragma (1.0)

  - A no-cache value prevents browsers from caching page. Send both headers or check HTTP version.

- Content-Encoding:

  - The way document is encoded. Browser reverses this encoding before handling document.

- Content-Length:

  - The number of bytes in the response.

  - Use ByteArrayOutputStream to buffer document so you can determine size.

- Content-Type:
  - The MIME type of document being returned.
  - use SetContentType to set this header.
- Location:
  - URL to which browser should reconnect.
  - Use sendRedirect instead of setting this directly.
- Refresh
  - The number of seconds until browser should reload page. Can also include URL to connect to.
- Set-cookie:
  - The cookies that browser should remember. Don't set this header directly; use addCookie instead.
- Last-Modified.
  - The time document was last changed.
  - Don't set this header explicitly; provide a getLastModified method instead.
- Expires:
  - The time at which document should be considered out of date & thus should no longer be cached.
  - Use setDateHeader to set this header.

Problems on Servlets:

The need to generate HTML code is also present in CGI.

From the point of view they are the same.

The drawbacks of servlets are few, but they include:

- Servlets need a special "servlet container" to run servlets, & most wel servers don't include this.
- Servlets need a Java Routine environment on the server to run servlets. CGI is a completely language independent protocol, so you can write CGI's in whatever languages you have available.
- Sometimes there are no Java interfaces to particular resources or de In this case writing a CGI in a language which does have native su -port for access to these resources or devices can be a lot simpler th setting up JNI to call through another language. And it might not even be possible through JNI.
- Although each CGI request usually requires a process start a C can be written in a fully compiled language & use fully compiled & optimized low-level system calls, so complex, long, or machin dependent processing can be faster & simpler using CGI's.
- You can easily test a CGI on the command-line of the server; just set a few environment variables & run it to see the output. This is not easy with servlets.

UNIT - VIII

UNIT-8

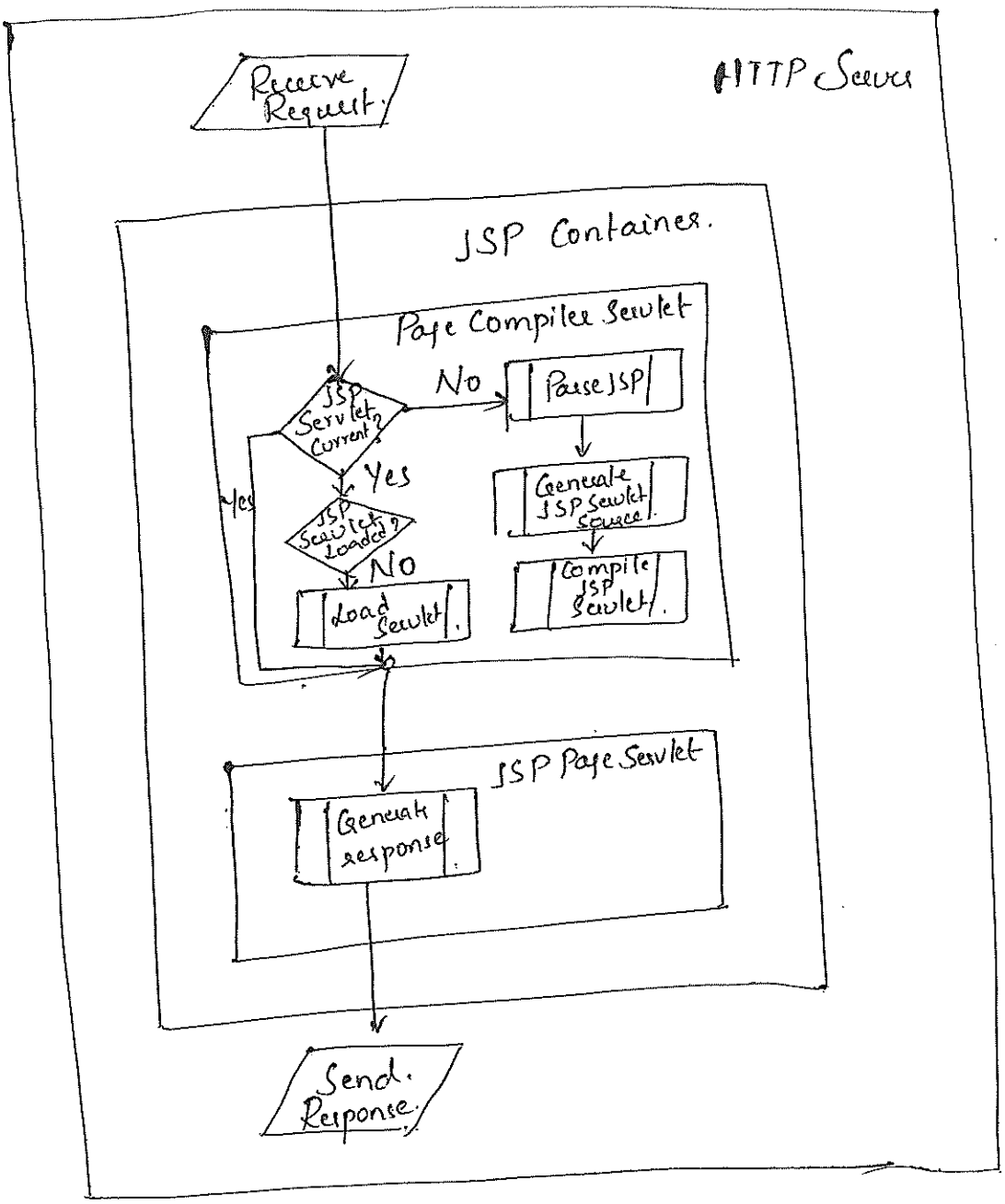
WORKING WITH JSP

Introduction to JSP:

- Java Server Pages are HTML pages embedded with snippets of Java code.
  - It is an inverse of a Java Servlet.
- Four different elements are used in Constructing JSP's.
  - Scripting Elements
  - Implicit Objects.
  - Directives.
  - Actions.

Architecture:

- JSPs run in two phases
  - Transition Phase
  - Execution Phase.
- In transition phase, JSP page is compiled into a servlet - called JSP Page Implementation class.
- In execution phase the compiled JSP is processed.



Types:

- 3 types of scripting elements:
  - → Declarations
  - → Scriplets
  - → Expressions.

Declarations:

Basics

- Declarations are used to define methods & instance variables

- Do not produce any output that is sent to client.
- Embeded in <%! and %> delimiters.

Eg:

```
<%!
   public void jspDestroy {
   S.O.Pln("JSP Destroyed");

}
public void jspInit()
  {
    SOPln("JSP Loaded");

}
int myVar=123;
%>
```

- The functions & variables defined are available to the JSP Page.

as well as to the servlet in which it is compiled.

Scriplets:

Basics:

- Used to embed java code in JSP pages.
  - Contents of JSP go into JSP page service() method.
  - Code should comply with syntactical & semantic construct of JA

- Embeded in <% & %> delimiters.

Eg: <%
int x=5;
int y=7;
int z=x+y;
%>

Expressions:

Basics:

- Used to write dynamic content back to the browser.
  - If the output of expression is Java primitive the value is printed back to the server.
  - If the output of an object then the result of calling toString on the object is output to the browser.
  - Embeded in <%= and %> delimiters.

Example:

- <%= "Fred" + " " + "Flinstone" %>
  prints "fred flinstone" to the browser.
- <%= Math.sqrt(100) %>
  prints 10 to the browser.

Java Implicit Objects:

Scope:

- Implicit objects provide access to server side objects.
  - e.g: request, response, session etc....
- There are four scopes of the object.
  - Page: Objects can only be accessed in page where they are referenced.
  - Request: Objects can be accessed within all pages that serv the current request.
  - Session: Objects can be accessed within the JSP pages for which the objects are defined.
  - Application: Objects can be accessed by all JSP pages in a given context.

List:

- request: Reference to the current request.
- response: Response to the request.
- session: session associated with current request.
- application: Servlet context to which a page belongs.
- pageContext: Object to access request, response, session & applic associated for the page.

- config: Servlet configuration for the page.
- out: Object that writes to the response output stream.
- page: instance of the page implementation class (this).
- exception: Available with JSP pages which are error pages.

Example:

```
<html>
<head>
<title> Implicit Objects </title>
</head>
<body style = "font-family : verdana; font-size : 10pt">
<p>
Using Request parameters ---- <br>
<b> Name : </b> <% = request.getParameter ("name") %>
</p>
<p>
<% out.println ("This is printed using the out implicit variable");
%>
</p>
<p>
Storing a string to the session ... <br>
<% session.setAttribute ("name", "Meeraj"); %>
Retrieving the string from session ... <br>
```

Name: <% = session.getAttribute("name") %>

Storing a string to the application. ---

<% application.setAttribute("name", "Meeraj"); %>

Retrieving the string from app ---

Name:

<% = application.getAttribute("name") %>

Storing a string to page content---- .

<% pageContext.setAttribute("name", "Meeraj"); %>

Retrieving the string from page content ---</br>

Name:

<% = pageContext.getAttribute("name") %>

</body>

</html>.

Directives:

Basics & Types:

- Messages sent to the JSP Container.
  - Aids the container in page translation.

- Used for
  - Importing tag libraries.
  - Import Required classes.
  - Set output buffering options.
  - Include content from external files.

- The jsp specification defines 3 directives:

  - Page: provides info about page, such as scripting language ie, used, content type, or buffer size.
  - Include - used to include the contents of external files.
  - Taglib - used to import custom actions defined in tag libraries

Page Directives:

Basics & Types:

- Page directive sets page properties used during translation.
  - JSP Page can have any number of directives.
  - Import directive can only occur once.
  - Embedded in <%@ and %> delimiters.

- Different directives are.

— Language: Defines server side Scripting Language.

— Extends: Declares the class which the servlet compiled from JSP needs to extend.

— Import: Declares the packages & classes that need to be import for using in the java code.

— session: Boolean which says if the session implicit variable i allowed or not.

— Buffer: defines buffer size of the JSP in kilobytes.

— autoFlush: When true the buffer is flushed when max buffer size is reached.

— isThreadSafe: If false the compiled servlet implements Single ThreadModel Interface.

— Info: String returned by the getServletInfo( ) of the compile servlet.

— errorPage: defines the relative URI of the web resource to which the response should be forwarded in case of an exception.

— ContentType: Defines MIME type for o/p response.

— isErrorPage: True for JSP pages that are defined as error pages.

- pageEncoding: defines the character encoding for the jsp page.

Eg:

```
<%@
  page language = "java"
  buffer = "10kb"
  autoflush = "true"
  errorPage = "/error.jsp"
  import = "java.util.*, javax.sql.RowSet".
%>
```

Include Directive:

Basics:

- Used to insert template text & JSP code during the translation phase.
  - The content of the included file specified by the directive is included in the including JSP page.

  Eg. <%@ include file = "included.jsp" %>

JSP Actions:

Basics & Types:

- Processed during the request processing phase.
  - As opposed to JSP directives which are processed during translati
- Standard actions should be supported by J2EE compliant webcon
- Custom actions can be created using tag libraries.

Different actions are:

- Include
- forward
- param
- useBean
- getProperty
- setProperty
- plugIn

Include:

- Include action is used for including resources in JSP page

→ Include directive includes resources in a JSP page at a translation time.

→ include action includes response of a resource into the response of JSP page.

→ Same as including resources using RequestDispatcher interface.

→ changes in the included resource while accessing the page.

→ Normally used for including dynamic resources.

• Example.

- <jsp:include page="includedPage.jsp">
- includes the output of includedPage.jsp into the page where this is included.

Forward:

• Forwards the response to other web specification resources.

- Same as forwarding to resources using RequestDispatcher interface.

• Forwaded only when content is not committed to other web applicatio resources.

- Otherwise an IllegalStateException is thrown.
- Can be avoided by setting a high Buffer size for the forwarding jsp page.

• Example:

- <jsp:forward page="Forwarded.html">
- forwards the request to Forwarded.html.

Param:

• Used in conjunction with Include & Forward actions to include additional request parameters to the included or forwarded resource.

• Example:

```
<jsp:forward page = "Param2.jsp">
  <jsp:param name = "first name" value = "Sanjay">
</jsp:forward>
```

- This will result in forwarded resource having an additional parameter FirstName with a value of Sanjay.

useBean:

• creates or finds a Java Object with the defined scope.

- Object is also available in the current JSP as a scripting variable

• Syntax:

```
<jsp:useBean id="name"
  scope="page/request/session/application"
  class="className" type="typeName"/
  bean="beanName" type="typeName"/
  type="typeName" />
```

- At least one of the type & class attributes must be present.
- We can't specify values for both the class & bean name.

• Example:

```
<jsp:useBean id="myName" scope="request" class="java.lang.Str
    <% firstName="Sanjay"; %>
</jsp:useBean>
```

getProperty:

- getProperty is used in conjunction with useBean to get property val of the bean defined by the useBean action.

- Example (getProperty)
  - <jsp:getproperty name="myBean" property="firstName" />
  - Name corresponds to the id value in the useBean.
  - Property refers to the name of the bean property.
  - setProperty is used to set bean properties.
  - Example (setProperty)
    - <jsp:setProperty name="myBean" property="firstName" value: "Sanjay" />
    - sets the name property of myBean to Sanjay Example (setproperty)
    - jsp:setProperty name="myBean" property="firstName" param="fname" />
    - sets the name property of myBean to the request parameter
    - <jsp:setProperty name="myBean" property="*" >
    - Sets property to the corresponding value in request.

- plugIn:
  - Enables the JSP Container to render appropriate HTML to:
    - Initiate the download of the Java plugin.
    - Execution of the specified applet or bean.

- plugIn standard action allows the applet to be embedded in a browser neutral fashion.

• Example:

```
<jsp: plugin type = "applet" code = "MyApplet.class" codebase = "/
    <jsp: params>
        <jsp: param name = "myParam" value = "122" />
    </jsp: params>
    <jsp: fallback > <b> Unable to load applet </b> <jsp: fallback
</jsp: plugin>
```

Working with JSP Standard Tag Library (JSTL):

• JSP is the technology that helps to seperate the front end presentation from the middle and backend tiers. The standard tag library is a powerful feature of JSP v1.1 that aids in that seperation. This technology is valuable to anyone who is building production quality web applications, & it is very applicable in today's market.

• Standard tag libraries allow the Java programmer to write the code that provides data access and other services, & they make those features available to the JSP author in a simple to use XML-like fashion. In short, a tag library is a collection of custom actions presented as tags.

Standard tags have many features that make them attractive to use from any JSP. Standard tags can

- be customized via attributes passed from the calling page, either statically or determined at runtime.
- have access to all the objects available to JSP pages including request, response, in & out.
- modify the response generated by the calling page.
- communicate with each other, you can create & initialize a JavaBean component, create a variable that refers to that bean in one tag, & then use the bean in another tag;
- be nested within one another, allowing for complex interactions within a JSP page; &
- encapsulate both simple & complex behaviors in an easy to use syntax & greatly simplify the readability of JSP Pages.

Composition of a Tag Library:

There are two types of components for a tag library: the tag library descriptor file & the tag handlers. With these a JSP is able to use tags contained in the library within its page.

The TLD file:

A tag library descriptor (TLD) file is an XML document that describes the library. TLD contains info abt the library as a whole & about each tag contained in library. TLDs are used by

a JSP container to validate the tags. (2)

There is typically some header information followed by elements used to define the tag library. The elements are.

<taglib> tag library itself
<tlibversion> tag library's version.
<jspversion> The JSP specification version the tag library depends on.
<shortname> A simple default name with a mnemonic value.
<uri> A optional URI that uniquely identifies the tag library.
<info> Descriptive information about tag library.

• There is only one TLD element required for all tags, & that is the one used to specify a tag handler's class: <tagclass> class name </tagclass>.

Introduction to AJAX (Asynchronous Java Script & XML).

AJAX-framework - is no programming or script language, no new invention & no seperate web service, module or plug-In. In common it is a marketing term for Remote Scripting with Javascript, CSS & DOM. It is an algorithm with old technologies similar to the Dynamic HTML.

• Ajax allows to create server connections in the background. while a user is interacting with a Web-front end.

• These connections can be created asynchronously, which means that the user need not wait until the server replies.

Eg: when a user types an email address into an input form, it is psble via AJAX to create a server connection in the background, check if the address is valid or not & give the information back to the user via an output.

Advantages:

- No pushing on a submit button & reloading of a complete website are needed. So the interactivity & speed for users are more efficient.
- AJAX is used for creating user friendly applications.

Disadvantages:

- As AJAX is a very new combination of old technologies, no one ca be sure if it is only a marketing type now or if it will really be established in some years.
- One big problem is the compatability.
- Another, Javascript can be switched off in browsers, because of security reasons.

* . To establish AJAX connections a little bit client-server web knowi g like parameter passing is needed.