z-index: It draws order for overlaid boxes.
(larget number drawn last)

```
#text{ z-index: 1 }
```

o/p:

Position properties:

display: value none means that element and its descedants are not related to and do not affect normal flow

Visibility: value hidden (intial value is visible) means that element and its descendants are not rendered but still do affect normal flow.

XML

What is XML?

-> XML stands for extensible Markup Language.

-> It helps information systems share structured data

-> A meta language that gives meaning to data that other applications can use.

-> Application and platform independent.

-> It allows various types of data

-> Extensible to accommodate new tags and

processing methods.

-> Allows user defined tags.

-> Represents data hierarchically (in a tree)

Advantages of XML

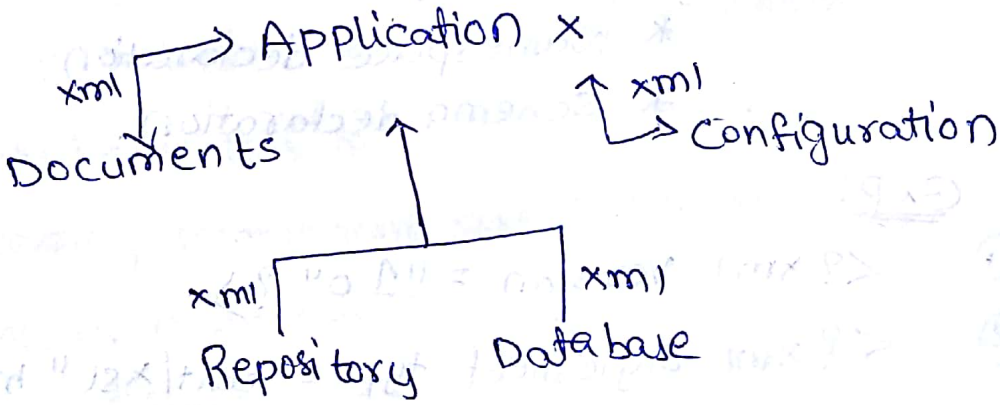
-> Simpler version of Standard Generalized Markup Language (SGML)

-> Its easy to understand

-> It is supported by a large number of platforms

-> It used across open standards. of W3C.

-> XML is a "use everywhere" data specification.



-> Strict rules

- -> Syntax
- -> Structure
- -> Case sensitive

* Parsers in XML

-> XML parser provide way to access or modify data present in an XML document. Java provides multiple ways to parse XML document.

Ex: DOM, SAX (Structure of An XML), XPath, JDOM etc

↳ Document object model

XML Syntax and Structure

-> Elements:

* Each element has a beginning and end tag. <tag> .... </tag>
* elements can be empty <tag/>

-> Attributes.

* describes an element eg: data type, data range etc.
* can only appear on beginning tag.

-> Processing instructions.

* Encoding specification (unicode by default)
* Namespace declaration
* Schema declaration.

Exp:-

```
1  <?xml version = "1.0" ?>
2  <?xml stylesheet type = "text/xsl" href=
       "template.xls" ?>
3  <ROOT>
4    <Element 1>
5      <subele 1/><subele 2/>
6    </Element 1>
7  <Element 2> </Element 2>
8  <Element 3 type = 'String'> </Element 3>
9  <Element 4 type = 'integer' value = '9.3' ></Element 4>
10 </ROOT>
```

-> 1, 2 lines are processing instructions (prolouge)
-> 3, 10 lines are root element (each xml doc should contain atleast one root element)
-> line 4 & 6,7 elements
-> line 5 is nested element
-> lines 8 & 9 are elements with attributes.

Rules for well-formed XML

-> There must be only one root element.
-> Sub-elements must be properly nested
  -> A tag must end within the tag in which it was started
-> Attributes are optional
  -> Defined by a optional schema.
-> Attributes values must be enclosed with " or '''
-> Processing instructions are optional.
-> XML is case sensitive
  - <tag> and <TAG> are not the same type of element

Namespace :-

-> Xml namespace is mechanism to avoid name conflicts by differentiating elements or attributes within an XML document that may have identical names, but different definitions.

Exp:- Lets take two RDB (Relation Database) tables of the following structure.

Employee table

empID	secID	name
E001	S001	Raju
E002	S002	Ram
E003	S003	Krishna

Section table

secID	name
S001	Exam
S002	Admin
S003	Network

```
SELECT employee.empID, section.name, employee
  name from employee, section where
  employee.secID = section.secID
```

-> However since the name column and secID colu exist in both tables, we would have to designate th table name before the name and secID columns c designate the alias of the table in order to clarify the table of the name and secID colum

```
SELECT e.empID, s.name, e.name from employee
  e, section s where e.secID = S.secID;
```

empID	e.name	s.name
E001	Raju	Exam
E002	Ram	Admin
E003	Krishna	Network

So what happen if data expressed in XML

employee XML

```
<employee>
<Person info>
<empID> E001 </empID>
<secID> S001 </secID>
<name> Raju </name>
</Person info>
<personinfo>
<empID> E002 </empID>
<secID> S002 </secID>
<name> Ram </name>
</personinfo>
.
.
.
</employee>
```

section XML

```
<section>
<sectionInfo>
<secID> S001 </secID>
<name> Exam </name>
<sectionInfo>
<sectionInfo>
<secID> S002 </secID>
<name> Admin </name>
</sectionInfo>
.
.
.
</section>
```

→ If we merge these two documents; the name element expressing employee name and the name element expressing department name conflicts.

* To avoid these name conflicts we need to use namespace in following structure.

<elementname xmlns:prefix = "namespaceIdentifier">

prefix ↑ arbitary text string

namespaceIdentifier ↑ URI (Uniform Resource Identifier)

Exp: <emp:employee xmlns:emp="urn:corp:emp">

<emp:personInfo>

:

</emp:personInfo>

</emp:employee>

In this example we have declared the namespace prefix as "emp", and the namespace identifier (URI) as "urn:corp:emp". This means that element names and attributes names with the "emp" prefix (including the employee element) all belong to the urn:corp:emp namespace.

Namespace declaration scope:

-> A separate namespace declaration can be made for the descendant element of an element for which a namespace declaration has already made, and a different namespace prefix within that scope can be used as well.

Exp: <list:employeeList xmlns:list = "urn:corp:list">

<list:personList xmlns:emp = "urn:corp:emp">

<emp:empID> E001 </emp:empID>

namespace "urn.corp.sec" scope ←

<sec:name xmlns:sec = "urn:corp:sec">

Exam </sec:name>

<emp:name> Raju </emp:name>

</list:personList>

</list:employeeList>

Namespace with attribute:

```
<emp:employee xmlns:emp="urn:corp:emp">
Exp: <emp:personInfo emp:empID="E001">
    <emp:secID>S001</emp:secID>
    <emp:name>Raju</emp:name>
  </emp:personInfo>
</emp:employee>
```

-> Attribute can either belongs to the same name-space of the element for which the attribute is defined, or belong to a completely different namesp-ace than that to which the element belongs.

* In above example the attribute belongs to same namespace as the element.

Exp:2: Attribute belongs to different namespace than the element.

```
<emp:employee xmlns:emp="urn:corp:emp">
  <emp:personInfo work:empID="E001"
      xmlns:work="urn:corp:work" >
    <emp:secID>S001</emp:secID>
    <emp:name>Raju</emp:name>
  </emp:personInfo> </emp:employee>
```

Exp3: Attribute does not belongs to any namespace.

```
<emp:employee xmlns:emp="urn:corp:emp">
<emp:personInfo empID="E001" >
  .
  .
</emp:personInfo></emp:employee>
```

Default Namespace:-

→ A "default namespace" is a namespace declaration that does not use a namespace prefix.

→ The scope of the default namespace is the element for which the namespace was declared and the related content.

Exp:- <elementname xmlns="namespaceidentifier">
↑ URI

```
<employeeList  xmlns = "urn:corp:list"
               xmlns:emp = "urn:corp:emp"
               xmlns:sec = "urn:corp:sec" >
  <personList empID = "E001" >
    :
  </personList>
</employeeList>
```

In above example default namespace declaration namespace applies to employeeList element and personList element but it does not applies to the empID attribute

Notation to cancel default name space

<elementName xmlns="" >

The xmlns="" designation frees an element within the namespace scope from belonging to any namespace

Exp: <secName xmlns="">Sales</secName>

CDATA :- (Character DATA)

-> CDATA sections can be used to "block-escape" literal text when replacing prohibited characters with entity refference is undesirable.

-> CDATA section can appear inside element content and allow < and & character literals to appear.

-> A CDATA sections begins with the characters sequence <![CDATA and ends with the character sequence]]>. Between the two character sequences, an XML parser ignores all markup characters such as <, > and &.

-> The only markup an XML parser recognizes inside a CDATA section is the closing character sequence]]>

-> CDATA section cannot be nested.

Exp:- <sometext>

<![CDATA[They are saying "x<y" & "z>y" so I guess that means that z>x]]>

</sometext>

Note: Whenever your using CDATA in xml document, your document should not contain the string "]]>" anywhere in the XML document.

XML DTD :-

-> The purpose of DTD is to define the legal building blocks of an XML document. It defines the document structure with a list of legal elements.

-> A DTD can be declared inline in your XML document, or as an external refference.

Internal DTD :-

-> This is an XML document with a document type definition.

Exp

DTD

```
① <?xml version = "1.0"?>
② <!DOCTYPE note [
③ <!ELEMENT note (to, from, heading, body)>
④ <!ELEMENT to (#PCDATA)>
⑤ <!ELEMENT from (#PCDATA)>
⑥ <!ELEMENT heading (#PCDATA)>
⑦ <!ELEMENT body (#PCDATA)>
]>
```

XML

```
<note>
<to> Ram </to>
<from> Sita</from>
<heading> Reminder</heading>
<body> don't forget to meet me this weekend!
</body> </note>
```

The DTD is interpreted like this

-> !ELEMENT note (in line 3) defines the element "note" as having four elements: "to, from, heading, body"

-> !ELEMET to (line 4) defines the "to" element to be of the type PCDATA. (Parsed Character DATA)

* lines 5, 6 & 7 are also type of PCDATA.

External DTD :-

-> DTD should be stored in external file lets name it as "note.dtd".

-> XML file should in following format to include dtd file.

Ex: <?xml version = "1.0" ?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
<to> Ram </to>
<from> Sita </from>
<heading> Reminder </heading>
<body> don't forget to meet me this weekend! </body>
</note>

Why to use a DTD ?

-> XML provides an application independent way of sharing data. With a DTD, independent group of people can agree to use a common DTD for interchanging data.

-> Your application can use a standard DTD to verify that data that you receive from the outside world is valid. You can also use a DTD to verify your own data.

XML Schema:-

What is XSD (XML Schema definition) file?

-> XSD is a recommondation of the W3C, it specifies how to formally describes the elements in an XML document. It can be used by programmers to verify each piece of item content in a document.

-> The purpose of an XML schema is to define the legal building blocks of an XML document.

Why to use XML schema :-

-> To provide list of elements and attributes in a vocabulary.

-> to associates types, such as an integer, string etc, or more specifically such as hatsize, sock-color etc with values found in document.

-> to constrain where elements and attributes can appear, and what can appear inside those elements, such as saying that a chapter

title occurs inside a chapter must consist of a chapter title followed by one or more paragraphs of text.

-> to provide documentation that is both human-readable and machine processable.

-> to give formal description of one or more documents.

A Comparision of complex data types in DTD and XML Schema

XML document

```
<Book>
  <title> XML </title>
  <author> Ram </author>
</Book>
```

DTD

```
<Book>
  <title> XML </title>
  <author> Ram </author>
</Book>
```

XML Scheema

```
<Book>
  <title> XML </title>
  <author> Ram </author>
</Book>
<!ELEMENT Book (title, author)>
<!ELEMENT title (#PCDATA)>
```

```
<!ELEMENT author (#PCDATA)>
<element name = 'Book' type = 'Booktype'/>
<complexType name = 'Booktype'>
    <element name = 'title' type = 'string'/>
    <element name = 'author' type = 'string'/>
</complexType>
```

Note: In above example, it conforms to both DTD and XML schema fragments, there is a big difference between them. In a DTD, all elements are global, whereas the XML schema is title and author to be defined locally. -- to accure only within the element Book.

* To exactly duplicate the effect of the DTD declaration in XML schema the element title and author must have a global scope.

Exp: Complex type defined with global simple types.

```
<element name = 'title' type = 'string'/>
<element name = 'author' type = 'string'/>
<element name = 'Book' type = 'Booktype'/>
    <complexType name = 'Booktype'>
        <element ref = 'title'/>
        <element ref = 'author'/>
    </complexType>
```

In previous example BookType is global and can be used to declare other elements.

Exp: Hiding BookType as a local type.

```
<element name='title' type='string' />
<element name='Book' >
    <complexType>
        <element ref='title' />
        <element ref='author' />
    </complexType>
</element>
```

→ We can also use namespace in XML schema.

```
Exp: <xs:rootElement xmlns:xs='urn:corp:
        schemans' >
    <xs:element name='title' type='string'/>
    <xs:element name='Book' >
    <xs:complexType>
        <xs:elementt ref='title' />
        <xs:element ref='author' />
    <xs:complexType>
</xs:element>
```

→ We can add multiple namespaces & we can import namespaces using from outside of the document.

→ We can set minOccurs & maxOccurs values in schema.

Document Object Model (DOM):

What is the DOM?

-> The DOM is an "application programming interface"

it can useful to interact with web pages.

(i) Add content to a HTML, XML documents.

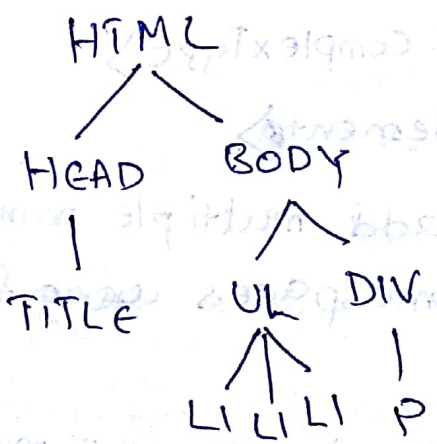
(ii) Delete content from a HTML, document

(iii) Change the content of a HTML document

-> Every HTML element in the document is an object.

<head> </head>, <body> </body>, <ul> ,   etc are objects.

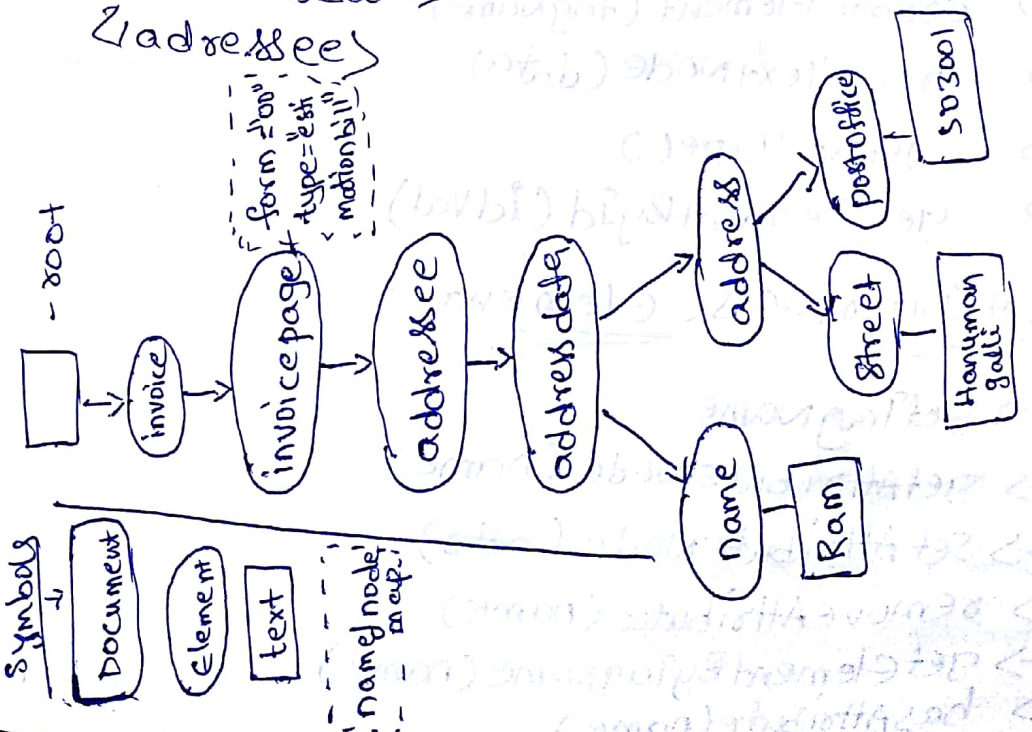
The model



→ Everything we can change in the document is a node
- elements, text within elements, HTML attributes etc.

DOM ~~interfaces~~ Structure Model

```
<invoice>
  <invoicepage form="00" type="estimation-bill">
  <addressee>
    <addressdata>
      <name> Ram </name>
    <address>
      <Street> Hanuman galli </Street>
      <postoffice> 503001 </postoffice>
    </address>
  </addressdata>
  </addressee>
```



DOM Interfaces: NODE

-> getNodeType
-> getNodeValue
-> getOwnerDocument
-> getParentNode
-> hasChildNode ; getChildNode
-> getFirstChild; getLastChild
-> getPreviousSibling ; getNextSibling
-> hasAttributes; getAttributes;
-> appendChild (newChild);
-> insertBefore (newChild, refChild)
-> replaceChild (newChild, OldChild)
-> removeChild(oldChild)

DOM Interfaces: Document:

-> getDocumentElement
-> createAttribute(name)
-> createElement (tagName)
-> createTextNode(data)
-> getDocType()
-> getElementById('idVal')

DOM Interfaces: Element:-

-> getTagName
-> getAttributeNode (name)
-> setAttributeNode (attr)
-> removeAttribute (name)
-> getElementByTagName (name)
-> hasAttribute(name)

Exp: document.getElementByClassName("Content")

It will retrive all elements which has class name content.

to store these elements create variable.

→ var myContent = document.getElementBy ClassName("Content");

Exp:② var H2 = document.getElementByTagName("h2")

→ H2[0].innerHTML = "YoYo"

It retrives content of first H2 tag in HTML.

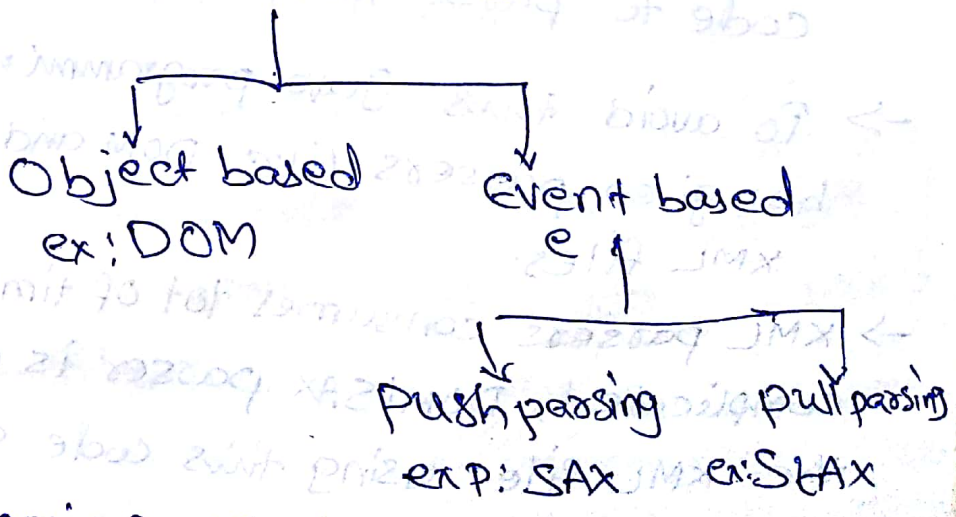
Exp③ var ele = document.getElementBy Id("page-title");

→ var ele stores content of element which contains id as "page-title".

* You can use all the given methods in previous page based on your requirement in program.

XML Parser Using Java

→ Reading XML file is called parsing.



StAX (Streaming API for XML)

DOM --> Document Object Model -> tree based
SAX --> Simple API for XML -> event based

Difference b/w DOM and SAX parsers.

DOM	SAX
1> DOM reads the entire document	1> SAX reads node by node
2> DOM is useful when small to medium size XML files	2> SAX is used when big XML files needs to be parsed
3> DOM is tree based parser	3> SAX is event based parser
4> DOM is little slow as compared to SAX	4> SAX is faster than DOM
5> DOM can insert and delete nodes.	5> SAX cannot insert and delete nodes

Why DOM and SAX?

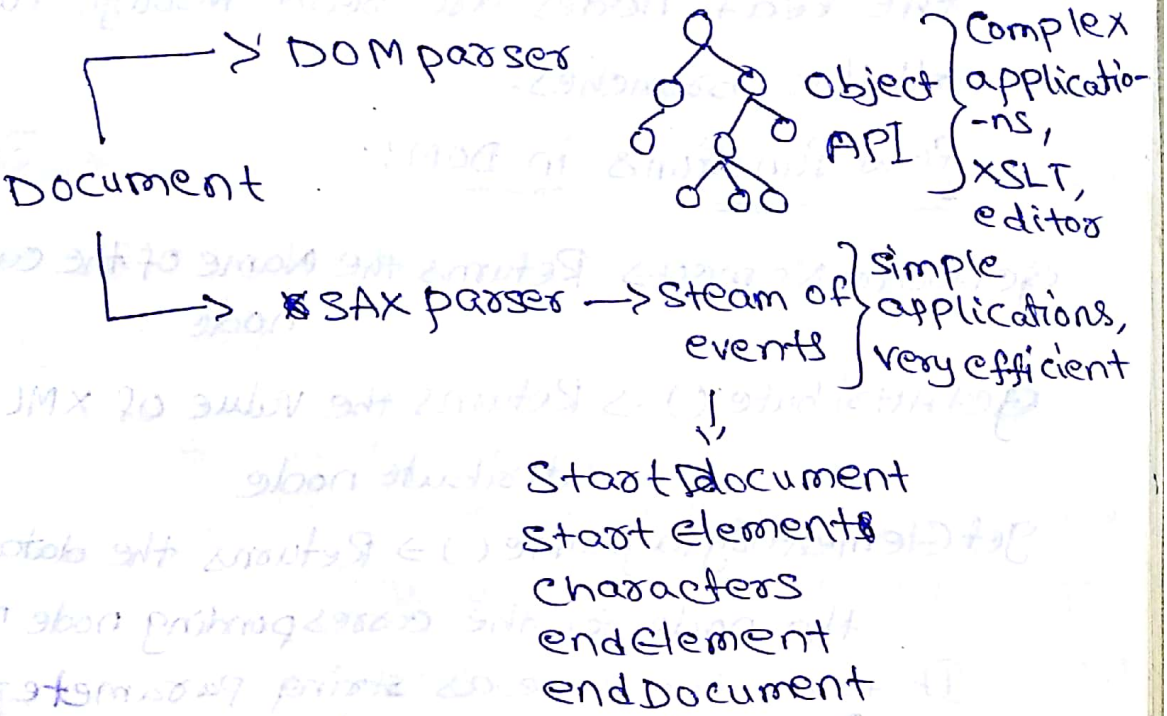
-> From the program, if the xml file data is to be read then developer has to write their own code to parse the xml file. It is complicated.

-> To avoid this Java programming language has given parsers like DOM and SAX to read the XML files.

-> XML parsers consumes lot of time and it is complicated, DOM/SAX parser is used to parse the XML file. Using this code can be developed fast and accurate.

Features of DOM and SAX

-> We know that DOM reads the XML file and stores in tree format; SAX reads in event based format.



DOM to create an XML file

-> In the program, document object is created.

-> Elements/nodes of the XML file are created.

-> Data to be placed in between the nodes are created in text object.

-> Text and elements (nodes) are linked.

-> Nodes are linked to the root node.

-> Finally, transformer Transform() method is used to convert the document object to XML file.

Parsing XML file using DOM :-

-> XML file is read into tree using DOM parser().

-> Number of branches are determined and all the leaf nodes are read through iterating all the branches.

few functions in DOM :-

getNodeName() -> Returns the Name of the current node.

getAttribute() -> Returns the value of XML attribute node

getElementByTagName() -> Returns the data of the node for the corresponding node name. It takes tag name as string parameter

SAX Parser :-

-> In the main() method, XML file is parsed by passed SAXparser object and is attached with a ReadXMLSAX class

-> For every node or data read, it calls a corresponding method in ReadXMLSAX class.

-> startDocument() and endDocument methods are called when an XML document is started to read and after reading the entire document.

-> When start tag is read, startElement() method is invoked.

-> When end tag is read, endElement() method is

invoked.

-> When the data in-between the tag is read, characters method is called.

CGI (Common Gateway Interface):-

-> The common gateway interface (CGI) is a standard way for a web server to pass a web user's request to an application program and receive data back to forward to the user.

-> When the user requests a webpage, the server sends back the request page. However when a user fills out a form on a web page and sends it in, it usually needs to be processed by an application program.

-> The web server typically passes the information to a small application program that processes the data and may send back confirmation message.

-> This method or convention for passing data back and forth between the server and the application is called the CGI. It's part of HTTP.

-> CGI developed in 1993 at NCSA (National Center for Super Computing Applications)

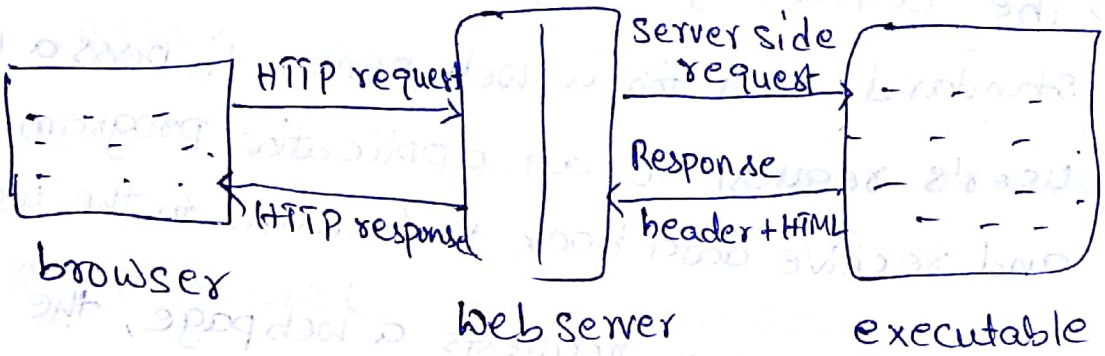
-> CGI script must be an executable (have 'x' rights) and must have the .cgi extension.

-> CGI script must be placed in the cgi-bin directory in the public_html

directory of the user.

Server-Side Web programming:

-> The HTTP Response consists of the output of an external program located on the server machine.



Drawbacks of CGI:

-> No special ~~Language~~ web-oriented language is used for writing CGI script errors are highly probable and so, security vulnerabilities due to these problems.

-> Usually a new process is created for each run of a CGI script; this increases the load on the server.

-> CGI scripts are executable files; they can write/ delete from the local disk. so this is a security vulnerability

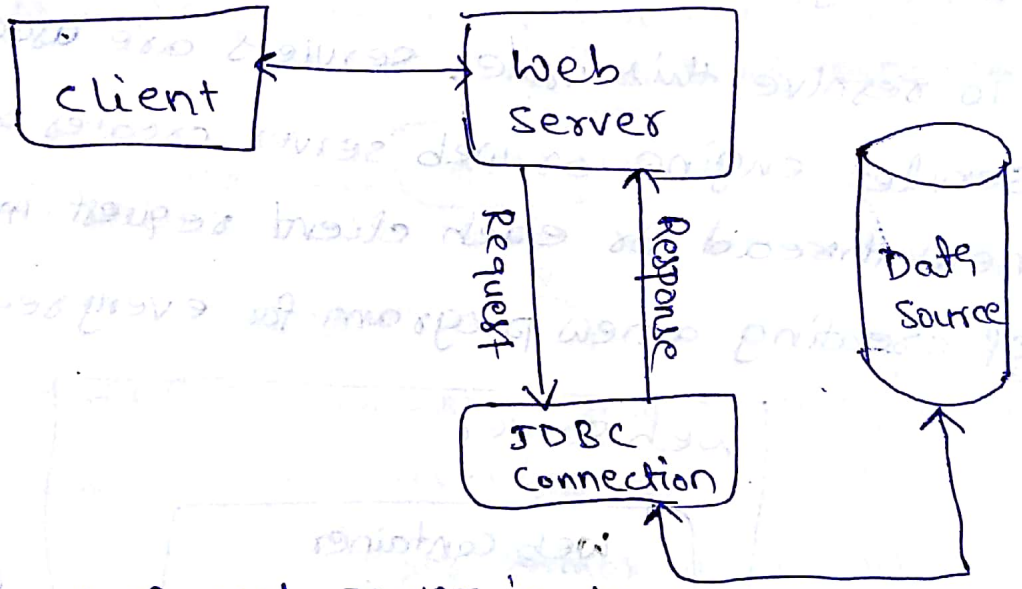
Web Programming with Servlets.

What is Servlet?

-> Servlet is a java technology to create dynamic web pages.

-> servlet is a web component which is deployed on web server/app server to create dynamic web pages.

-> When the client sends the request, server receives it and sends to the corresponding servlet program. servlet program interprets the request and responds accordingly.



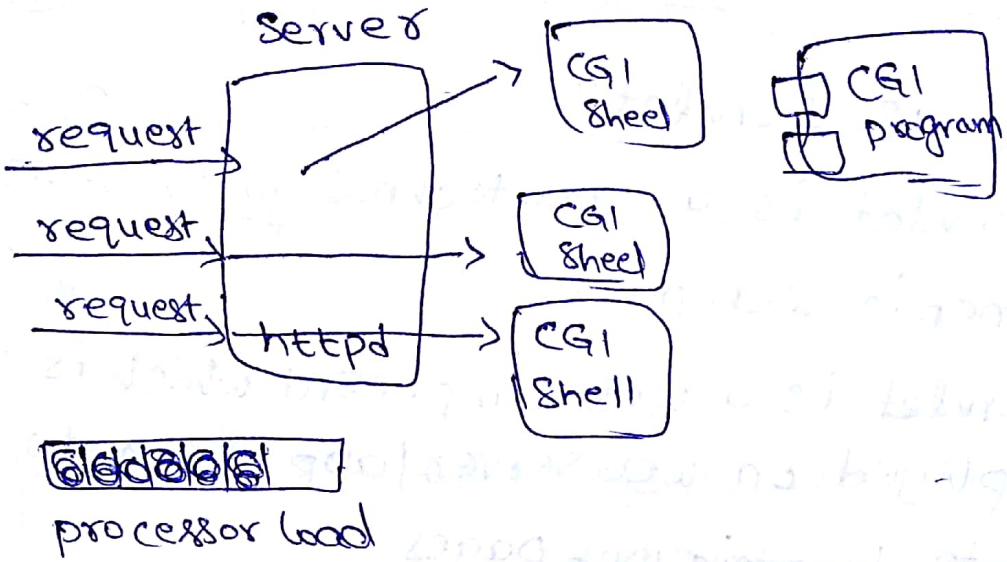
Exp: of web server is Apache tomcat or Glass fish ect.

Why servelts are used over CGI?

-> for every request from the client a new CGI program is created in the server system.

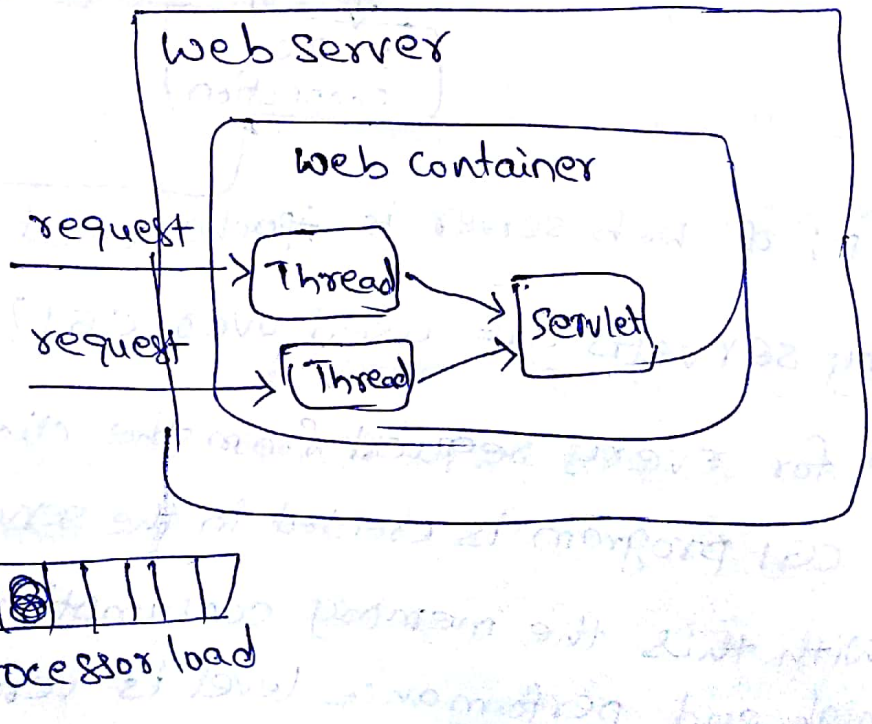
With this the memory consumption is very high and performance level is very slow.

There are high chance of system crash too.



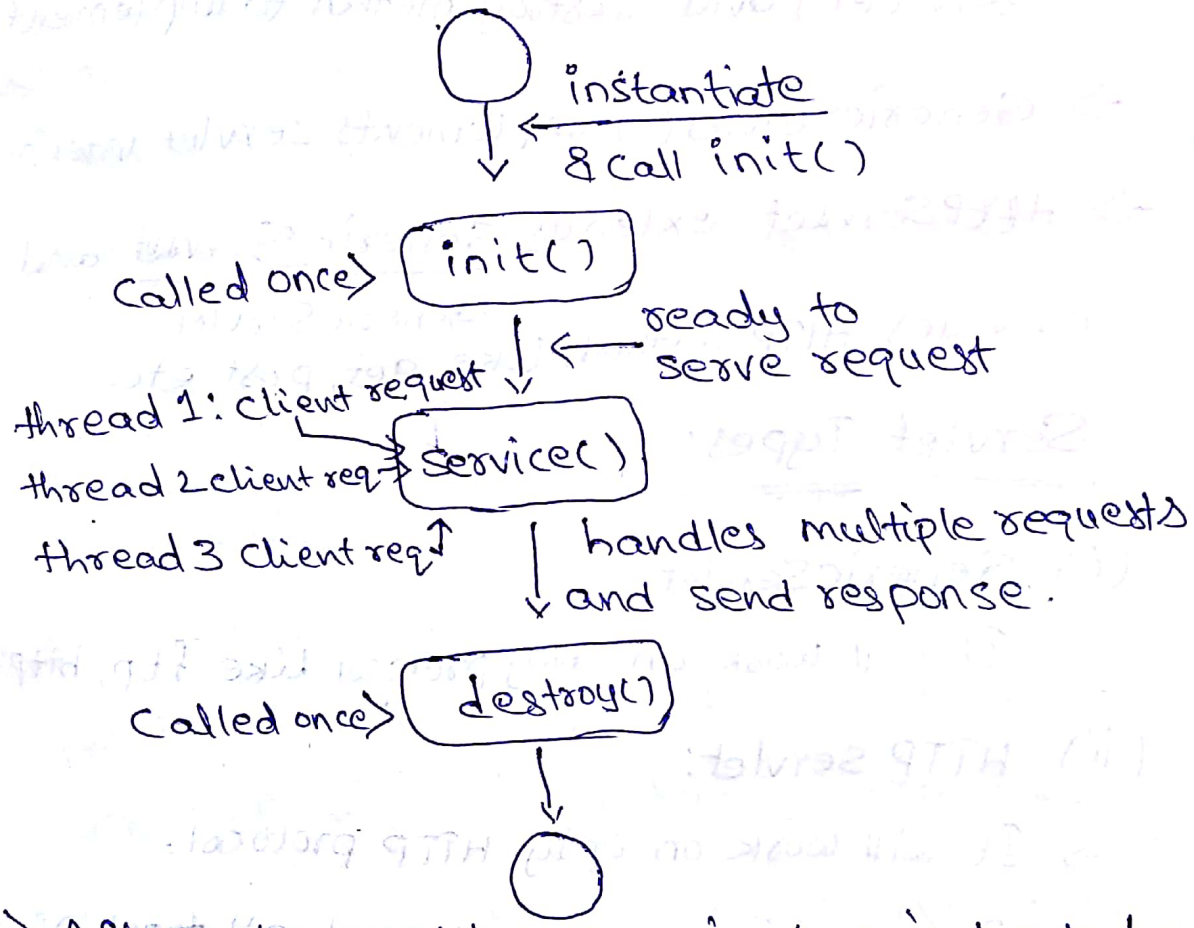
→ CGI will starts a new process in each request so that processor load gets increase and sometimes processor will get lack of memory issue or system crash.

→ To resolve this issue, servlets are used. Servlet engine or web server creates a new thread for each client request instead of creating a new program for every request.



→ When a new thread created for each request, memory saved and server can serve many clients and performance of the project is very good.

Servlet Life Cycle:-



→ After the servlet program is done, it has to be deployed, in server (Apache tomcat)

→ When the server comes up, servlet is loaded in the memory. At this time init() method is called.

→ This is for intializing the attributes of the servlet. Its like constructor.

→ service(), when any client request comes service() method is invoked.

-> Destroy() method is called when servlet is about to be removed from memory.

Servlet Interface:-

-> Servlet is a interface, it provides init(), service() and destroy method to implement.

-> GenericServlet() implements servlet interface.

-> HTTPServlet extends GenericServlet and provides HTTP methods like get, post etc.

Servlet Types:-

(i) GenericServlet:
It will work on any protocol like ftp, http etc.

(ii) HTTP Servlet:
-> It will work on only HTTP protocol.
-> It supports get, post and all types of methods where genericServlet does not support these protocols.

Servelet Configuration with eclipse:

Open Eclipse -> click on window menu -> preferences -> server -> Run time environment -> Add -> select the tomcat version 7.0 -> provide the directory of tomcat installed directory -> OK.

-> or you can work with netbeans which is comes with glassfish server.

-> Most of the people will use tomcat because its small in size and easy to deploy.

-> Tomcat is very simple. it does not require any installation. It s free.

-> Tomcat is good for small to medium size project.

-> If its a big size project ~~or~~ then its better to use highend servers which are paid servers exp: websphere.

Project Structure :- in NetBeans IDE

-> All the html pages will stored in web pages folder. Jsp files also stored in this folder.

-> Web-Inf folder will contains all .xml files

-> Source Packages folder will contains all .java files.

-> Libraries folder will contains all libraries & servers

-> Configuration files folder will contains MANI-FEST.MF file. which contains configuration info.

Note: Apart from above folders you can create your own folders in Web pages to store js, css, ~~ico~~ files and images or other files.
js folder for js files, .css folder for .css files,

images folder to image files and files folder → to store other files like pdf, txt, sql etc.

Servelet Exceptions:-

→ Servlet has the following two exceptions.

(i) ServeletException: defines a servlet exception that a servlet throws while processing the client request.

(ii) UnavailableException: Defines a servlet exception that thrown by a servlet, when a servlet is unavailable.

Cookies in Servlet:-

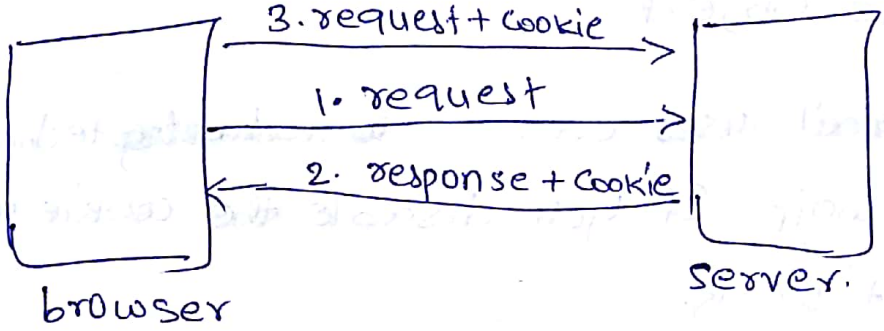
→ A cookie is a small piece of information that is persisted between the multiple client requests.

→ A cookie has a name, a single value and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

How Cookie works:-

→ By default each request is considered as a new request. In cookies technique, we add cookie with response from the servlet.

→ So Cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus we recognize the user as old user.



Types of Cookies:-

(i) Non-persistent Cookie:-

→ It is valid for single session only. It is removed each time when the user closes the browser.

(ii) Persistent Cookie:-

→ It is valid for multiple sessions. Its not removed each time when user closes the browser. It's removed only if user logout.

Advantages of Cookies:-

→ Simplest technique of maintaining the state

→ Cookies are maintained at client side.

Disadvantages of Cookies:-

1 -> It will not work if cookie is disabled from the browser.

-> Only textual information can be set in cookie object.

Note:- Gmail uses cookies ~~to technology~~ technique for login. If you disable the cookie gmail wont work.

Cookie class:-

-> javax.servlet.http.Cookie class provide the functionality of using cookies. It provides a lot of useful methods for cookies.

Constructor of Cookie class:-

-> Cookie() -> Construct a cookie

Cookie(String name, String value) -> Construct a cookie with a specified name and value.

Useful methods in Cookie class:-

Method description.

-> public void setMaxAge(int expiry) : Sets the maximum age of the cookie in seconds.

-> public String getName(): It returns the name of the cookie. The name cannot be changed after creation.

-> public getValue(): return the value of the cookie.

-> public void setName(): changes the name of the cookie.

-> public void setValue(): changes the value of the cookie.

Note: for adding cookie or getting the value from the cookie, we need some methods provided by other interfaces they are:

(i) public void addCookie(Cookie ck); method of HTTPServletResponse interface is used to add cookie in response object.

(ii) public Cookie[] getCookies; method of HTTPServletRequest interface is used to return all the cookies from the browser.

Examples:-

Create Cookie:-

-> Exp: Cookie ck = new Cookie("user", "Ram"); // creating cookie object

-> response.addCookie(ck); // adding Cookie in the response.

Delete Cookie :-

Exp :→ Cookie ck = new Cookie("user", "");// deleti
value of a cookie

→ ck.setMaxAge(0); // Changing the maximum age 0 (zero) seconds.

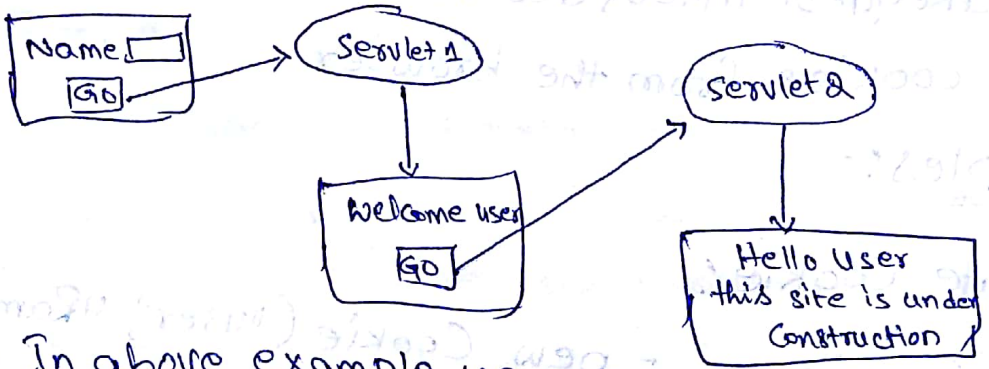
→ response.addCookie(ck); // adding cookie in the response.

get Cookies :-

Exp.

→
```
Cookie ck[] = request.getCookies();
for(int i=0; i<ck.length; i++)
{
out.print("<br>" + ck[i].getName() + " " +
ck[i].getValue()); // printing name and value of
                   Cookie.
}
```

Exp of Servlet Cookie :-



→ In above example we are storing the name of the user in the cookie object and accessing it in another servlet. As we know that sessions correspondent to the particular user. So if you access it from too many browser with different values, you will get the different value.

HTTP Session

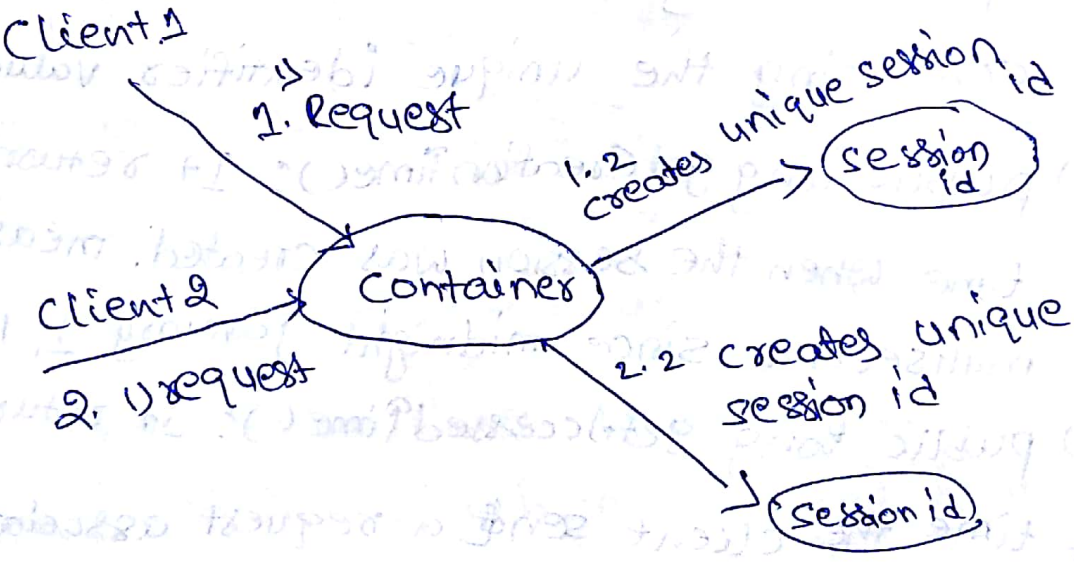
→ An HTTP session object can hold conversation-nal state across multiple requests from the same client. In other words, It persists for an entire session with a specific client.

HTTPSession Interface :-

→ Container creates a session id for each user. The container uses this id to identify the particular user. An object of HTTPSession can be used to perform two tasks:

(i) bind objects

(ii) View and manipulate information about a session such as the session identifier, creation time, and last accessed time.



How to get the HTTPSession Object!.

-> HTTPServletRequest interface provides two methods to get the object of HTTPSession.

(i) public HTTPSession getSession(): It returns the current session associated with this request, or if the request does not have a session creates one.

(ii) public HTTPSession getSession(boolean create): It return the current HTTP session associated with this request or, if there is no current session and creates is true, return a new session

Commonly used methods of HTTPSession!-

(i) public String getId(): It returns a string containing the unique identifier value.

(ii) public long getCreationTime(): It returns the time when the session was created, measured in milliseconds since midnight January 1, 1970 GMT.

(iii) public long getAccessedTime(): It returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.

(iv) public void invalidate(): It invalidates this session the unbinds any object bound to it.

Connecting Database using JDBC:-

→ ~~Java soft~~ JavaSoft developed a single API for database access -- JDBC. As part of this process, they kept three main goals in mind:

(i) JDBC should be an SQL-level API
(ii) JDBC should capitalize on the experience of existing database APIs
(iii) JDBC should be simple.

→ An SQL level API means that JDBC allows us to construct SQL statements and embeded them inside Java API calls.

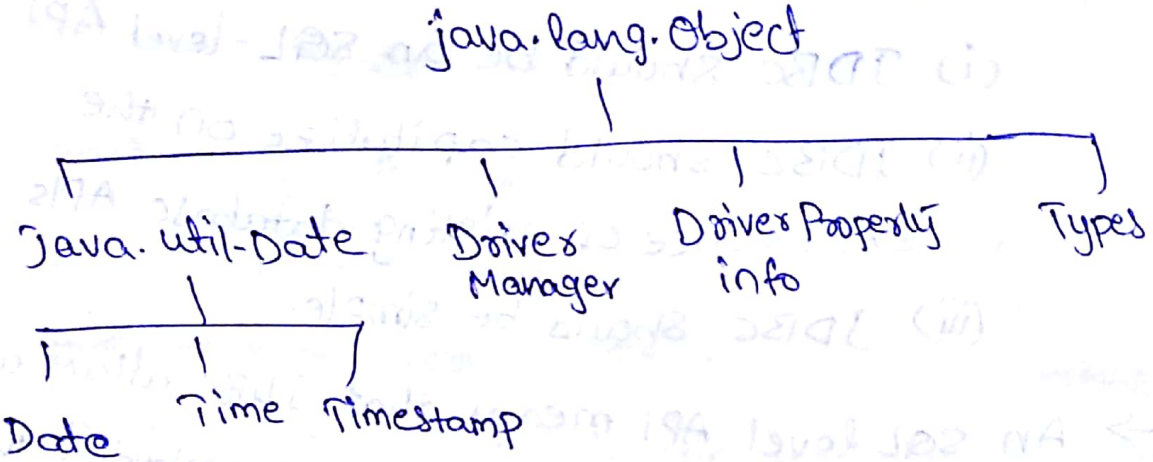
→ JDBC lets you smoothly translate between the world of the database and the world of Java application. Your results from the database, for instance, are returned as Java variables and access problem get thrown as exceptions.

→ The JDBC accomplishes its goals through a set of Java interfaces, each implemented differently by individual vendors. The set of classes that implemented the JDBC interfaces for a particular database engine is called a JDBC driver.

The structure of JDBC

Driver Statement connection ResultSet ResultSet Meta Data

Statement
|
Prepared statement Database MetaData
|
Callable Statement



The classes and interfaces of java.sql, the JDBC API Package shown above.

5 Steps to connect to the database in Java

(i) Register the driver class:

-> The forName( ) method of class, class is used to register the driver class. This method is used to dynamically load the driver class.

Syn:- public static void forName(String class Name) throws ClassNotFoundException

Exp: Class.forName(driver);

where driver is a string which holds a value "com.mysql.jdbc.Driver". this driver value will get change based on sql driver.

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

Class.forName(driver);

(ii) Create the connection object:

-> The getConnection method of DriverManager class is used to establish connection with the database.

Exp: Connection con = null;

con = DriverManager.getConnection(url+db, user, pw);

in above example url+db, user, pw are strings.

- String url = "jdbc:mysql://localhost:3306/";
  String db = "test";
  String user = "root";
  String pw = "password";

Note: Database name (: value of db), username (: value of user) and password (: value of pw) may get changed based on existing databases and sql configuration.

(iii) Create the Statement Object:

-> The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

Syn:- public Statement createStatement()
throws SQLException

Exp:- Statement stmt = con.createStatement();

(iv) Execute the query:

-> The ~~execute query of method~~ executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

Syn:- public ResultSet executeQuery(String sql)
throws SQLException

Exp:- ResultSet rs = stmt.executeQuery("Select * from emp");

```
While (rs.next())
{
    System.out.println(rs.getInt(1)+ " " + rs.get
    String(2));
}
```

v) Close the Connection Object.

-> By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

Syn: public void close() throws SQLException

Exp: con.close();

JavaServer Pages (JSP)

Introduction:-

-> JSP technology is used to create web applications just like Servlet technology. It can be thought of an extension to servlet because it provides more functionalities than servlet such as expression language, jstl (JSP Standard Tag Library) etc.

-> A JSP page consist of a HTML tags and JSP tags. The JSP pages are easier to maintain than servlet because we can separate designing and development. It provides some additional features such as Expression language, Custom Tag etc.

Advantages of JSP over Servlet :

(1) Extension to servlet:

-> JSP technology is the extension to servlet technoly. We can use all the features of servlet in JSP. In addition to we can use implict objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.

(2) Easy to maintain:

-> JSP can be easily managed because we can easily separate our business logic with presentation logic. In servlet we mix our business logic with the presentation logic.

(3) Fast Development: No need to recompile and redeploy:-

-> If JSP page is modified, we don't need to recompile and redeploy the project. The servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

(4) Less Code than Servlet:-

-> In JSP we can use a lot of tags such as action tags, jstl, custom tags etc. that reduce the code. Moreover we can use implict objects.

LifeCycle of a JSP page:

The JSP page follow these phases:

(i) Translation of JSP page

(ii) Compilation of JSP page

(iii) Class loading (class file is loaded by the classloader)

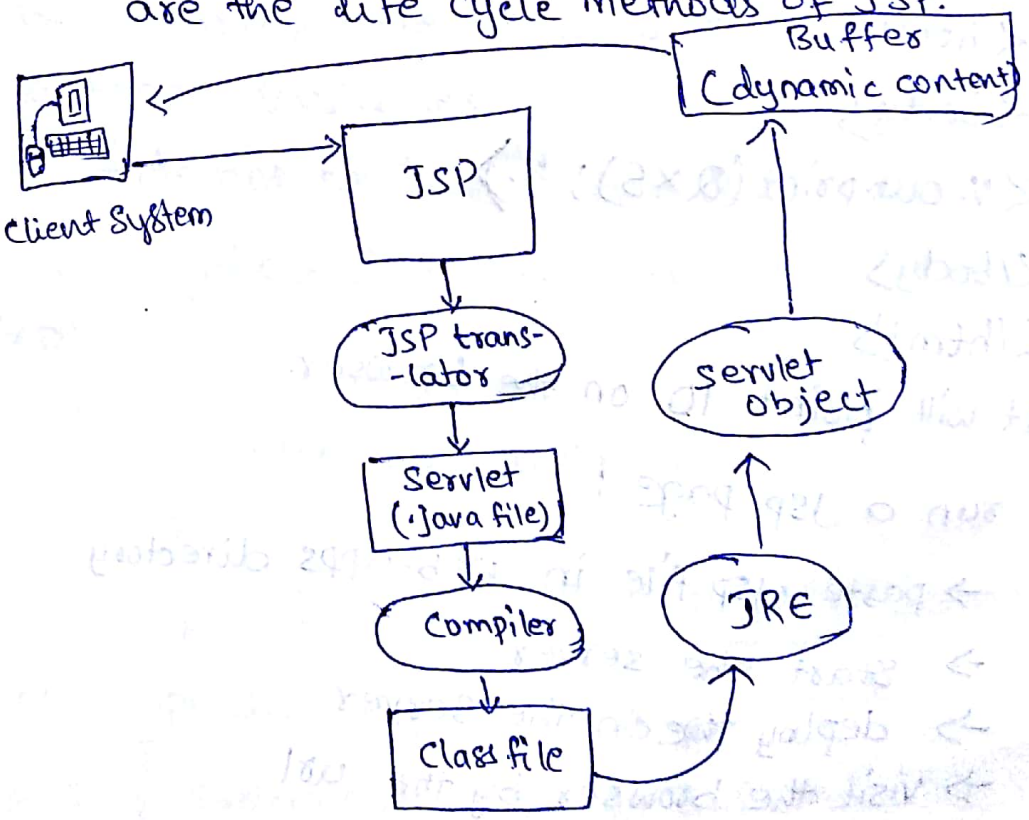
(iv) Instantiation (Object of the generated Servlet is created)

(v) Intialization (jspInit() method is invoked by the container)

(vi) Request processing (-jspService() method is invoked by the container)

(vii) Destroy (jspDestroy() method is invoked by the container).

Note: jspInit(), -jspService(). and jspDestroy() are the life cycle methods of JSP.



→ As depicted in the previous diagram, JSP page is translated into servlet by the help of JSP translator.

→ The JSP translator is a part of webserver that is responsible to translate the JSP into servlet.

→ Afterthat servlet page is compiled by the compiler and get converted into the class file.

→ Moreover, all the processes that happens in servlet is performed on JSP later like intialization, committing response to the browser and destroy.

Simple JSP page:-

```
index.jsp
<html>
<body>
<% out.print(2*5); %>
</body>
</html>
```

*It will print 10 on the browser.

To run a JSP page:

→ paste .jsp file in web-apps directory
→ Start the server
→ deploy on the server
→ visit the browser by the url

http://localhost:8080/myappliation/index.jsp

Note: myapplication is a project name.

JSP Processing:

-> four different ~~types~~ elements are used in constructing JSPs.

- -> Scripting elements
- -> Implict Objects
- -> Directive
- -> Actions

Scripting elements:

Types:-

-> There are three kinds of scripting elements

- -> Declarations
- -> Scriplets
- -> Expressions

Declarations:

-> Declarations are used to define methods & instance variables.

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

Exp:

```
<%!
    public void jspDestroy()
    {
    {  System.out.println("JSP destroyed");
    }
public void jspInit()
{
System.out.println("JSP loaded");
```

```
int myVar = 123;
%>
```

- The functions and variables defined are available to the JSP page as well as to the servlet in which is compiled.

Scriptlets:

Scriptlets are

-> used to embed java code into JSP page.

-> Contents of JSP go into _JSPpageservice() method

-> Code should comply with syntactical and semantic construct of java

-> Embeded in <% and %> delimiters.

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

Expressions:

-> Expressions are used to write dynamic content back to the browser.

- If the output of expression is Java primitive the value is printed back to the browser.
- If the output is an object then the result of calling toString on the object is output to the browser.

- Embeded in delimiter <%= and %>.

Exp: <%= "Rama" + " " + "Krishna" %>
prints Rama Krishna to the browser.

<%= Math.sqrt(100) %>
prints 10 to the browser.

Java Implict Objects:

-> Implict objects provide access to ~~the~~ server side objects.

Exp: request, response, session etc.

-> There are four scopes of the object

- Page: Objects can only be accessed in the page where they are referenced.
- Request: Objects can be accessed within all pages that serve the current request.

(Including the pages that are forwarded to and included in the original jsp page)

- Session: Objects can be within the JSP pages for which the objects are defined.
- Application: Objects can be accessed by all JSP pages in a given context.

List of Java implict objects:

(i) request: Reference ~~Request~~ to the current request

(ii) response: Response to the request

(iii) session: session associated with current request.

(iv) Application: Servlet context to which a page belongs.

(v) pageContext: Object to access request, response, session and application associated with a page.

(vi) Config: Servlet configuration for the page

(vii) out: Object that writes to the response output stream.

(viii) page: instance of the page implementation calls (this)

(ix) exception: Available with JSP pages which are error pages.

Directives:-

Basics & types:

-> Directives are used for
- importing tag libraries
- Import required classes
- Set output buffering options
- Include content from external files.

-> The JSP specification defines three directives

(i) Page: It's used to provide information about page, such as scripting language that is used, content type or buffer size.

(ii) Include: It's used to include the content of external files.

(iii) Taglib - It's used to import custom actions defined in tag libraries

Page directives:

-> Page directive sets Page properties used during page translation.

- JSP page can have any number of directives.
- import directive can occur only occurance.
- imbeded in <%@ and %> delimiters.

Types of page directives:

- Language:- (default java) defines server side scripting language (exp: java)
- Extends: Declares the class which the servlet compiled from JSP needs to extend.
- Import: Declares the package and classes that need to be imported for using in the java code (comma separated list).
- Session: (default true) Boolean which says if the session implict variable is allowed or not.
- Buffer: defines buffer size of the jsp in Kbs (if set none no buffering is done).

autoFlush: When true the buffer is flushed when max buffer size is reached (if set to false an exception is thrown when buffer exceeds the limit)

isThreadSafe: (default true) If false compiled servlet implements SingleThreadModel interface.

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

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

contentType: (default text/html) defines MIME (Multipurpose Internet Mail extensions) type for the output response

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

pageEncoding: defines the character encoding for the jsp page.

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

Include directive:

-> Include directives are used to insert template text and Jsp code during the translation phase.

-> The content of the include file specified by the directive is included in the including JSP page.

Exp: <%@ include file = "included.jsp" %>

JSP Actions:

-> JSP actions are processed during the request processing phase.

- as opposed to Jsp directives which are poocessed during translation.

-> Standard actions should be supported by J2EE complaint web servers.

-> Custom actions can be created using tag libraries.

JSP ACtions types :-

① Include:

-> Include action used for including resources in a jsp page.

- Include directives includes resources in a JSP page at translation time.
- Include actions includes response of a resource into the process of the JSP page.

- same as including resources using RequestDispatcher interface.
- Changes in the included resourse reflected while accessing the page.
- Normally used for the including dynamic resources.

Exp:

<jsp:include page = "includedPage.jsp" />

It includes the output of includedPage.jsp into the page where this is included.

② Forward :-

-> Forward action forwards the response to other web specification resources.

- same as forwarding to resources using RequestDispatcher interface.

-> Forwarded only when content is not commited to other web application resources.

- otherwise an IllegalStateException is thrown
- can be avoided by setting a high buffer size for the forwarding jsp page.

Exp:- <jsp:forward page = "Forwarded.html" />

it forwards the request to Forwarded.html

③ Param:-

-> Param action is used in conjuction with Include & forward actions to include additional request parameters to the included or forwarded resource.

Exp:-

```
<jsp:forward page = "Param2.jsp" >
    <jsp:param name = "FirstName" value =
        "Ram" >
</jsp:forward>
```

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

④ UseBean:

-> useBean action 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" />
```

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

Exp":

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

⑤ get/setProperty":

→ getProperty action is used to in conjuction with useBean to getProperty value of the bean defined by the useBean action.

Exp":

```
<jsp:getProperty name = "myBean"
        property = "firstName" />
```

- name correspondent to the 'id' value in the useBean.
- property refers to the name of the bean property

-> ⑥ SetProperty used to set bean properties.

Exp:

<jsp:setProperty name = "myBean" property = "first-Name" value = "Ram" />

- sets the name property of myBean to Ram

Exp 2: <jsp:setProperty name = "myName" property = "firstName" param = "fname" />

- sets the name property of myBean to the request parameter fname.

Exp 3: <jsp:setProperty name = "myBean" property = "*" >

- sets property to the corresponding value in request

⑦ PlugIn:

-> It enables the JSP container to render appropriate HTML (based on the browser type) to:

- initiate the download of the Java plugin
- execution of the specified applet or bean

-> PlugIn standard action allows the applet to be embeded in a browser neutral fashion

Exp:-

```
<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>
```

JavaScript

-> JavaScript is an object-based language

-> JavaScript is an example of a scripting language that is embeded in HTML document.

-> Javascript is a light weight and cross-platform.

-> Javascript is not compiled but translated. Javascript translator is responsible to translate javascript code.

Uses of Javascript:-

-> Javascript is used to create interactive websites. It mainly used for
  -> client side validation
  -> dynamic dropdown menus