

WEB TECHNOLOGIES

[R15A0520]

LECTURE NOTES

B.TECH III YEAR – II SEM(R15)
(2018-19)



DEPARTMENT OF
COMPUTER SCIENCE AND ENGINEERING

MALLA REDDY COLLEGE OF ENGINEERING &
TECHNOLOGY

(Autonomous Institution – UGC, Govt. of India)

Recognized under 2(f) and 12 (B) of UGC ACT 1956

(Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified)
Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad – 500100, Telangana State, India

(R15A0520) WEB TECHNOLOGIES

Objectives:

- Giving the students the insights of the Internet programming and how to design and implement complete applications over the web.
- It covers the notions of Web servers and Web Application Servers, Design Methodologies with concentration on Object-Oriented concepts, Client-Side
- Programming, Server-Side Programming, Active Server Pages, Database Connectivity to web applications, Adding Dynamic content to web applications,
- Programming Common Gateway Interfaces, Programming the User Interface for the web applications.

UNIT I:

Web Basics and Overview: Introduction to Internet, World Wide Web, Web Browsers, URL, MIME, HTTP, Web Programmers Tool box.

HTML Common tags: List, Tables, images, forms, frames, Basics of CSS and types of CSS.

Client-Side Programming (Java Script): Introduction to Java Script, declaring variables, functions, Event handlers (onclick, onsubmit, etc.,) and Form Validation.

UNIT II:

Server-Side Programming (PHP): Declaring Variables, Data types, Operators, Control structures, Functions, Reading data from web form controls like text buttons, radio buttons, list, etc., Handling File Uploads, Handling Sessions & Cookies.

Introduction to XML: Document type definition, XML Schemas, Document Object model, Presenting XML , Introduction to XHTML, Using XML Processors: DOM and SAX.

UNIT III:

Web Servers and Servlets: Tomcat web server, Installing the Java Software Development Kit, Tomcat Server & Testing Tomcat.

Introduction to Servlets: Lifecycle of a Servlet, JSDK, Deploying Servlet, The Servlet API, The javax. Servlet Package, Reading Servlet 150 parameters, Reading Initialization parameters. The javax.servlet HTTP package, Handling Http Request & Responses, Using Cookies-Session Tracking.

UNIT IV:

Introduction to JSP: The Problem with Servlet. The Anatomy of a JSP Page, JSP Processing. JSP Application Design with MVC Setting Up and JSP Environment, JSP Declarations, Directives, Expressions, Code Snippets, implement objects, Requests, Using Cookies and Session for Session Tracking.

UNIT V:

Database Access: Database Programming using JDBC, JDBC drivers, Studying Javax.sql.* package, Connecting to database in PHP, Execute Simple Queries, Accessing a Database from a Servlet and JSP page.

Java Beans: Introduction to Beans, Deploying java Beans in a JSP page.

TEXT BOOKS:

1. Web Technologies, Uttam K Roy, Oxford University Press
2. The Complete Reference PHP – Steven Holzner, Tata McGraw-Hill

REFERENCE BOOKS:

1. Programming world wide web-Sebesta, Pearson Education ,2007.
2. Core SERVLETS ANDJAVASERVER PAGES VOLUME 1: CORE TECHNOLOGIES By Marty Hall and Larry Brown Pearson
3. Internet and World Wide Web – How to program by Dietel and Nieto PHI/Pearson Education Asia.
4. Jakarta Struts Cookbook, Bill Siggelkow, S P D O'Reilly for chap 8.
5. March's beginning JAVA JDK 5, Murach, SPD
6. An Introduction to Web Design and Programming –Wang-Thomson

OUTCOMES:

- Analyze a web page and identify its elements and attributes.
- Create web pages using XHTML and Cascading Styles sheets.
- Installation and usage of Server software's.
- Database Connectivity to web applications
- Build web applications using Servlet and JSP

INDEX

UNIT NO	TOPIC	PAGE NO
I	Web Basics and Overview	01 - 06
	HTML common tags	07 – 16
	Cascading Style Sheets	17 – 19
	Introduction to Java Script	20 - 32
II	Server-Side Programming (PHP)	33 - 59
	Introduction to XML	60 - 86
III	Web Servers and Servlets	87 - 88
	Introduction to Servlets	89 - 112
IV	Introduction to JSP	113 - 116
	JSP ELEMENTS	116 - 126
V	Database Access	127 - 133
	Java Beans	134 - 136

UNIT - I

Web Basics and Overview: Introduction to Internet, World Wide Web, Web Browsers, URL, MIME, HTTP, Web Programmers Tool box.

HTML Common tags: List, Tables, images, forms, frames, Basics of CSS and types of CSS.

Client-Side Programming (Java Script): Introduction to Java Script, declaring variables, functions, Event handlers (onclick, onsubmit, etc.,) and Form Validation.

Introduction to Internet:- A global computer network providing a variety of information and communication facilities, consisting of interconnected networks using standardized communication protocols. "the guide is also available on the Internet"

The Internet is the global system of interconnected computer networks that use the Internet protocol suite (TCP/IP) to link devices worldwide. It is a network of networks that consists of private, public, academic, business, and government networks of local to global scope, linked by a broad array of electronic, wireless, and optical networking technologies. The Internet carries a vast range of information resources and services.

History of Internet

This marvelous tool has quite a history that holds its roots in the cold war scenario. A need was realized to connect the top universities of the United States so that they can share all the research data without having too much of a time lag. This attempt was a result of Advanced Research Projects Agency (ARPA) which was formed at the end of 1950s just after the Russians had climbed the space era with the launch of Sputnik. After the ARPA got success in 1969, it didn't take the experts long to understand that how much potential can this interconnection tool have. In 1971 Ray Tomlinson made a system to send electronic mail. This was a big step in the making as this opened gateways for remote computer accessing i.e. telnet.

During all this time, rigorous paper work was being done in all the elite research institutions. From giving every computer an address to setting out the rules, everything was getting penned down. 1973 saw the preparations for the vital TCP/IP and Ethernet services. At the end of 1970s, Usenet groups had surfaced up. By the time the 80s had started, IBM came up with its PC based on Intel 8088 processor which was widely used by students and universities for it solved the purpose of easy computing. By 1982, the Defense Agencies made the TCP/IP compulsory and the term "internet" was coined. The domain name services arrived in the year 1984 which is also the time around which various internet based marked their debut. A worm, or a rust the computers, attacked in 1988 and disabled over 10% of the computer systems all over the world. While most of the researchers regarded it as an opportunity to enhance computing as it was still in its juvenile phase, quite a number of computer companies became interested in dissecting the cores of the malware which resulted to the formation Computer Emergency Rescue Team (CERT). Soon after the world got over with the computer worm, World Wide Web came into existence. Discovered by Tim Berners-Lee, World Wide Web was seen as a service to connect documents in websites using hyperlinks.

World Wide Web

The World Wide Web (abbreviated WWW or the Web) is an information space where documents and other web resources are identified by Uniform Resource Locators (URLs), interlinked by hypertext links, and can be accessed via the Internet. English scientist Tim Berners-Lee invented the World Wide Web in 1989. He wrote the first web browser computer program in 1990 while employed at CERN in Switzerland. The Web browser was released outside CERN in 1991, first to other research institutions starting in January 1991 and to the general public on the Internet in August 1991.

The World Wide Web has been central to the development of the Information Age and is the primary tool billions of people use to interact on the Internet. Web pages are primarily text documents formatted and annotated with Hypertext Markup Language (HTML). In addition to formatted text, web pages may contain images, video, audio, and software components that are rendered in the user's web browser as coherent pages of multimedia content.

Embedded hyperlinks permit users to navigate between web pages. Multiple web pages with a common theme, a common domain name, or both, make up a website. Website content can largely be provided by the publisher, or interactively where users contribute content or the content depends upon the users or their actions. Websites may be mostly informative, primarily for entertainment, or largely for commercial, governmental, or non-governmental organizational purposes



WWW is another example of client/server computing. Each time a link is followed, the client is requesting a document (or graphic or sound file) from a server (also called a Web server) that's part of the World Wide Web that "serves" up the document. The server uses a protocol called HTTP or Hyper Text Transport Protocol. The standard for creating hypertext documents for the WWW is Hyper Text Markup Language or HTML. HTML essentially codes plain text documents so they can be viewed on the Web.

Browsers:

WWW Clients, or "Browser": The program you use to access the WWW is known as a browser because it "browses" the WWW and requests these hypertext documents. Browsers can be graphical, allows to see and hear the graphics and audio;

text-only browsers (i.e., those with no sound or graphics capability) are also available. All of these programs understand http and other Internet protocols such as FTP, gopher, mail, and news, making the WWW a kind of "one stop shopping" for Internet users.

Year	List of Web browsers
1991	World Wide Web (Nexus)
1992	Viola WWW, Erwise, MidasWWW, MacWWW (Samba)
1993	Mosaic, Cello,[2] Lynx 2.0, Arena, AMosaic 1.0
1994	IBM WebExplorer, Netscape Navigator, SlipKnot 1.0, MacWeb, IBrowse, Agora (Argo), Minuet
1995	Internet Explorer 1, Internet Explorer 2, Netscape Navigator 2.0, OmniWeb, UdiWWW, Grail
1996	Arachne 1.0, Internet Explorer 3.0, Netscape Navigator 3.0, Opera 2.0, PowerBrowser 1.5,[4] Cyberdog, Amaya 0.9,[5] AWeb, Voyager
1997	Internet Explorer 4.0, Netscape Navigator 4.0, Netscape Communicator 4.0, Opera 3.0,[6] Amaya 1.0[5]
1998	iCab, Mozilla
1999	Amaya 2.0,[5] Mozilla M3, Internet Explorer 5.0
2000	Konqueror, Netscape 6, Opera 4,[7] Opera 5,[8] K-Meleon 0.2, Amaya 3.0,[5] Amaya 4.0[5]
2001	Internet Explorer 6, Galeon 1.0, Opera 6,[9] Amaya 5.0[5]
2002	Netscape 7, Mozilla 1.0, Phoenix 0.1, Links 2.0, Amaya 6.0,[5] Amaya 7.0[5]
2003	Opera 7,[10] Apple Safari 1.0, Epiphany 1.0, Amaya 8.0[5]
2004	Firefox 1.0, Netscape Browser, OmniWeb 5.0
2005	Opera 8,[11] Apple Safari 2.0, Netscape Browser 8.0, Epiphany 1.8, Amaya 9.0,[5] AOL Explorer 1.0, Maxthon 1.0, Shiira 1.0
2006	Mozilla Firefox 2.0, Internet Explorer 7, Opera 9,[12], SeaMonkey 1.0, K-Meleon 1.0, Galeon 2.0, Camino 1.0, Avant11, iCab 3
2007	Apple Safari 3.0, Maxthon 2.0, Netscape Navigator 9, NetSurf 1.0, Flock 1.0, Conkeror
2008	Google Chrome 1, Mozilla Firefox 3, Opera 9.5,[13], Apple Safari 3.1, Konqueror 4, Amaya 10.0,[5] Flock 2, Amaya 11.0[5]
2009	Google Chrome 2–3, Mozilla Firefox 3.5, Internet Explorer 8, Opera 10,[14], Apple Safari 4, SeaMonkey 2, Camino 2,surf, Pale Moon 3.0[15]
2010	Google Chrome 4–8, Mozilla Firefox 3.6, Opera 10.50,[16], Opera 11, Apple Safari 5, K-Meleon 1.5.4,
2011	Google Chrome 9–16, Mozilla Firefox 4-9, Internet Explorer 9, Opera 11.50, Apple Safari 5.1, Maxthon 3.0, SeaMonkey 2.1–2.6
2012	Google Chrome 17–23, Mozilla Firefox 10–17, Internet Explorer 10, Opera 12, Apple Safari 6, Maxthon 4.0, SeaMonkey 2.7-2.14
2013	Google Chrome 24–31, Mozilla Firefox 18–26, Internet Explorer 11, Opera 15–18, Apple Safari 7, SeaMonkey 2.15-2.23
2014	Google Chrome 32–39, Mozilla Firefox 27–34, Opera 19–26, Apple Safari 8
2015	Google Chrome 40–47, Microsoft Edge, Mozilla Firefox 35–43, Opera 27–34, Vivaldi

2016	Google Chrome 48–55, Mozilla Firefox 44–50, Microsoft Edge 14, Opera 35–42, Apple Safari 10, SeaMonkey 2.24–2.30, Pale Moon 26.0.0[17], Pale Moon 27.0.0[18]
2017	Google Chrome 56–60, Microsoft Edge 15, Mozilla Firefox 51–55.0.2, Opera 43–45, Opera Neon

Uniform Resource Locators, or URLs: A Uniform Resource Locator, or URL is the address of a document found on the WWW. Browser interprets the information in the URL in order to connect to the proper Internet server and to retrieve your desired document. Each time a click on a hyperlink in a WWW document instructs browser to find the URL that's embedded within the hyperlink.

The elements in a URL: **Protocol://server's address/filename**

Hypertext protocol: <http://www.aucegypt.edu>

File Transfer Protocol: <ftp://ftp.dartmouth.edu>

Telnet Protocol: <telnet://pac.carl.org>

News Protocol: <news:alt.rock-n-roll.stones>

What are Domains? Domains divide World Wide Web sites into categories based on the nature of their owner, and they form part of a site's address, or uniform resource locator (URL). Common top-level domains are:

.com—commercial enterprises	.mil—military site
org—organization site (non-profits, etc.)	int—organizations established by international treaty
.net—network	.biz—commercial and personal
.edu—educational site (universities, schools, etc.)	.info—commercial and personal
.gov—government organizations	.name—personal sites

Additional three-letter, four-letter, and longer top-level domains are frequently added. Each country linked to the Web has a two-letter top-level domain, for example .fr is France, .ie is Ireland.

MIME (Multi-Purpose Internet Mail Extensions): MIME is an extension of the original Internet e-mail protocol that lets people use the protocol to exchange different kinds of data files on the Internet: audio, video, images, application programs, and other kinds, as well as the ASCII text handled in the original protocol, the Simple Mail Transport Protocol (SMTP). In 1991, Nathan Borenstein of Bellcore proposed to the IETF that SMTP be extended so that Internet (but mainly Web) clients and servers could recognize and handle other kinds of data than ASCII text. As a result, new file types were added to "mail" as a supported Internet Protocol file type.

Servers insert the MIME header at the beginning of any Web transmission. Clients use this header to select an appropriate "player" application for the type of data the header indicates. Some of these players are built into the Web client or browser (for example, all browsers come with GIF and JPEG image players as well as the ability to handle HTML files); other players may need to be downloaded.

New MIME data types are registered with the Internet Assigned Numbers Authority (IANA).

MIME is specified in detail in Internet Request for Comments 1521 and 1522, which amend the original mail protocol specification, RFC 821 (the Simple Mail Transport Protocol) and the ASCII messaging header, RFC 822.

Hypertext Transport Protocol:

HTTP means HyperText Transfer Protocol. HTTP is the underlying protocol used by the World Wide Web and this protocol defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands.

For example, when you enter a URL in your browser, this actually sends an HTTP command to the Web server directing it to fetch and transmit the requested Web page. The other main standard that controls how the World Wide Web works is HTML, which covers how Web pages are formatted and displayed.

HTTP is called a stateless protocol because each command is executed independently, without any knowledge of the commands that came before it. This is the main reason that it is difficult to implement Web sites that react intelligently to user input.

HTTPS: A similar abbreviation, HTTPS means Hyper Text Transfer Protocol Secure. Basically, it is the secure version of HTTP. Communications between the browser and website are encrypted by Transport Layer Security (TLS), or its predecessor, Secure Sockets Layer (SSL).

The Web Programmer's Toolbox:

- **HTML** - a *markup* language
 - To describe the general form and layout of documents
 - HTML is **not** a programming language - it cannot be used describe **computations**.
 - An HTML document is a mix of **content** and **controls**
 - Controls are **tags** and their **attributes**
 - Tags often delimit content and specify something about how the content should be arranged in the document
For example, <p>Write a paragraph here </p> is an *element*.
 - Attributes provide additional information about the content of a tag
For example,
- Plug ins
 - Integrated into tools like word processors, effectively converting them to WYSIWYG HTML editors
- Filters
 - Convert documents in other formats to HTML

- Advantages of both filters and plug-ins:
 - Existing documents produced with other tools can be converted to HTML documents
 - Use a tool you already know to produce HTML
- Disadvantages of both filters and plug-ins:
 - HTML output of both is not perfect - must be fine tuned
 - HTML may be non-standard
 - You have two versions of the document, which are difficult to synchronize
- XML
 - A meta-markup language (a language for defining markup language)
 - Used to create a new markup language for a particular purpose or area
 - Because the tags are designed for a specific area, they can be meaningful
- JavaScript
 - A client-side HTML-embedded scripting language
 - Provides a way to access elements of HTML documents and dynamically change them
- Flash
 - A system for building and displaying text, graphics, sound, interactivity, and animation (movies)
 - Two parts:
 1. Authoring environment
 2. Player

Supports both motion and shape animation

PHP

A server-side scripting language

Great for form processing and database access through the Web

Ajax

Asynchronous JavaScript + XML

- No new technologies or languages

Much faster for Web applications that have extensive user/server interactions

Uses asynchronous requests to the server

Requests and receives small parts of documents, resulting in much faster responses

Java Web Software

Servlets – server-side Java classes

JavaServer Pages (JSP) – a Java-based approach to server-side scripting

JavaServer Faces – adds an event-driven interface model on JSP

ASP.NET

Does what JSP and JSF do, but in the .NET environment

Allows .NET languages to be used as server-side scripting language

Ruby

A pure object-oriented interpreted scripting language

Every data value is an object, and all operations are via method calls

Both classes and objects are dynamic

Rails

A development framework for Web-based applications

Particularly useful for Web applications that access databases

Written in Ruby and uses Ruby as its primary user language

HTML Common tags:-

HTML is the building block for web pages. HTML is a format that tells a computer how to display a web page. The documents themselves are plain text files with special "tags" or codes that a web browser uses to interpret and display information on your computer screen.

- HTML stands for Hyper Text Markup Language
- An HTML file is a text file containing small markup tags
- The markup tags tell the Web browser how to display the page
- An HTML file must have an htm or html file extension.

HTML Tags:- HTML tags are used to mark-up HTML elements .HTML tags are surrounded by the two characters < and >. The surrounding characters are called angle brackets. HTML tags normally come in pairs like **and** The first tag in a pair is the start tag, the second tag is the end tag . The text between the start and end tags is the element content . HTML tags are not case sensitive, **means the same as** .

The most important tags in HTML are tags that define headings, paragraphs and line breaks.

Tag	Description
<!DOCTYPE...>	This tag defines the document type and HTML version.
<html>	This tag encloses the complete HTML document and mainly comprises of document header which is represented by <head>...</head> and document body which is represented by <body>...</body> tags.
<head>	This tag represents the document's header which can keep other HTML tags like <title>, <link> etc.
<title>	The <title> tag is used inside the <head> tag to mention the document title.
<body>	This tag represents the document's body which keeps other HTML tags like <h1>, <div>, <p> etc.
<p>	This tag represents a paragraph.
<h1> to <h6>	Defines header 1 to header 6
 	Inserts a single line break
<hr>	Defines a horizontal rule
<!-->	Defines a comment

Headings:-

Headings are defined with the <h1> to <h6> tags. <h1> defines the largest heading while <h6> defines the smallest.

<h1>This is a heading</h1>

<h2>This is a heading</h2>

<h3>This is a heading</h3>

<h4>This is a heading</h4>

<h5>This is a heading</h5>

<h6>This is a heading</h6>

Paragraphs:-

Paragraphs are defined with the <p> tag. Think of a paragraph as a block of text. You can use the align attribute with a paragraph tag as well.

```
<p align="left">This is a paragraph</p>
<p align="center">this is another paragraph</p>
```

Note: You must indicate paragraphs with <p> elements. A browser ignores any indentations or blank lines in the source text. Without <p> elements, the document becomes one large paragraph. HTML automatically adds an extra blank line before and after a paragraph.


Line Breaks:-

The
 tag is used when you want to start a new line, but don't want to start a new paragraph. The
 tag forces a line break wherever you place it. It is similar to single spacing in a document.

This Code	output
<p>This is a para graph with line breaks</p>	This is a para graph with line breaks

Horizontal Rule The element is used for horizontal rules that act as dividers between sections like this:

The horizontal rule does not have a closing tag. It takes attributes such as align and width

Code	Output
<hr width="50%" align="center">	

Sample html program

```
<!DOCTYPE html>
<html>
  <head>
    <title>This is document title
    </title>
  </head>
  <body>
    <h1>This is a heading</h1>
    <p>Document content goes here.....</p>
  </body>
</html>
```



- Type the above program in notepad and save with some file name eg:sample.html
- Open the file with browser and the webpage looks like this

Lists:-HTML offers web authors three ways for specifying lists of information.

All lists must contain one or more list elements. Lists are of three types

- 1)Un ordered list
- 2)Ordered List
- 3)Definition list

HTML Unordered Lists:An unordered list is a collection of related items that have no special order or sequence. This list is created by using HTML tag. Each item in the list is marked with a bullet.

Example

```
<!DOCTYPE html>
<html>
  <head>
    <title>HTML Unordered List</title>
  </head>
  <body>
    <ul>
      <li>Beetroot</li>
      <li>Ginger</li> <li>Potato</li>
      <li>Radish</li>
    </ul>
  </body>
</html>
```



- Beetroot
- Ginger
- Potato
- Radish

HTML Ordered Lists:- items are numbered list instead of bulleted, This list is created by using tag.

```
<!DOCTYPE html>
<html>
  <head>
    <title>HTML Ordered List</title>
  </head>
  <body>
    <ol>
      <li>Beetroot</li>
      <li>Ginger</li>
      <li>Potato</li>
      <li>Radish</li>
    </ol>
  </body>
</html>
```



1. Beetroot
2. Ginger
3. Potato
4. Radish

HTML Definition Lists:- HTML and XHTML supports a list style which is called definition lists where entries are listed like in a dictionary or encyclopedia. The definition list is the ideal way to present a glossary, list of terms, or other name/value list. Definition List makes use of following three tags.

- 1). <dl> - Defines the start of the list
- 2). <dt> - A term
- 3). <dd> - Term definition
- 4). </dl> - Defines the end of the list

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<title>HTML Definition List</title>
```

```
</head>
```

```
<body>
```

```
<dl>
```

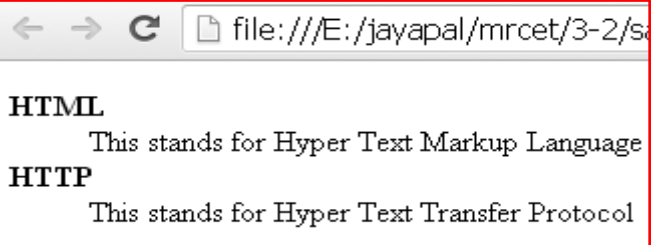
```
<dt><b>HTML</b></dt> <dd>This stands for Hyper Text Markup Language</dd>
```

```
<dt><b>HTTP</b></dt> <dd>This stands for Hyper Text Transfer Protocol</dd>
```

```
</dl>
```

```
</body>
```

```
</html>
```



HTML tables:

The HTML tables allow web authors to arrange data like text, images, links, other tables, etc. into rows and columns of cells. The HTML tables are created using the <table> tag in which the <tr> tag is used to create table rows and <td> tag is used to create data cells.

Example:

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<title>HTML Tables</title>
```

```
</head>
```

```
<body>
```

```
<table border="1">
```

```
<tr>
```

```
<td>Row 1, Column 1</td> <td>Row 1, Column 2</td>
```

```
</tr>
```

```
<tr> <td>Row 2, Column 1</td> <td>Row 2, Column 2</td>
```

```
</tr>
```

```
</table>
```

```
</body>
```

```
</html>
```

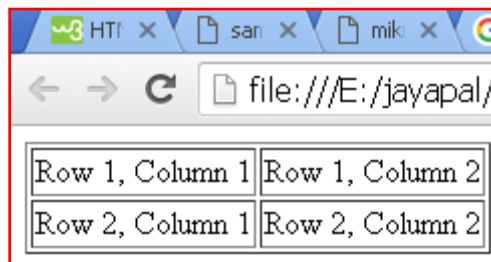
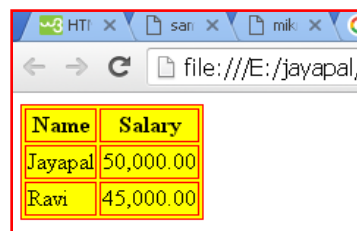


Table Heading: Table heading can be defined using `<th>` tag. This tag will be put to replace `<td>` tag, which is used to represent actual data cell. Normally you will put your top row as table heading as shown below, otherwise you can use `<th>` element in any row.

Tables Backgrounds: set table background using one of the following two ways:

- 1) bgcolor attribute - You can set background color for whole table or just for one cell.
- 2) background attribute - You can set background image for whole table or just for one cell. You can also set border color also using bordercolor attribute.

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Tables</title> </head>
<body>
  <table border="1" bordercolor="red" bgcolor="yellow">
    <tr> <th>Name</th>
    <th>Salary</th> </tr>
    <tr>
      <td>Jayapal </td> <td>50,000.00</td>
    </tr>
    <tr> <td>Ravi</td> <td>45,000.00</td>
    </tr>
  </table>
</body>
</html>
```



Images are very important to beautify as well as to depict many complex concepts in simple way on your web page.

Insert Image:

insert any image in the web page by using `` tag.

``

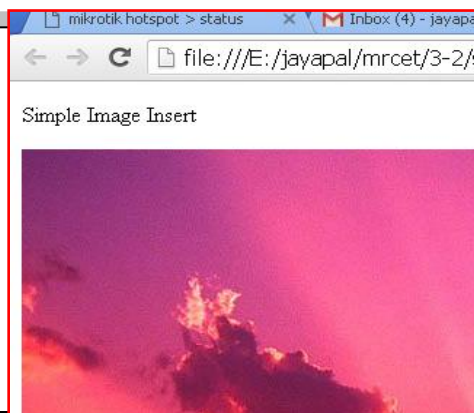
Attribute Values

Value	Description
left	Align the image to the left
right	Align the image to the right
middle	Align the image in the middle
top	Align the image at the top
bottom	Align the image at the bottom

`
<html>
 <head>
 <title>Using Image in Webpage</title>
 </head>
 <body> <p>Simple Image Insert</p>

 </body>
</html>

```

**HTML FORMS:**

HTML Forms are required to collect some data from the site visitor. For example, during user registration you would like to collect information such as name, email address, credit card, etc. A form will take input from the site visitor and then will post it to a back-end application such as CGI, ASP Script or PHP script etc. The back-end application will perform required processing on the passed data based on defined business logic inside the application. There are various form elements available like text fields, text area fields, drop-down menus, radio buttons, checkboxes, etc.

```
<form action="Script URL" method="GET|POST"> form elements like input, text area etc. </form>
```

**Form Attributes**

Apart from common attributes, following is a list of the most frequently used form attributes:

Attribute	Description
action	Backend script ready to process your passed data.
method	Method to be used to upload data. The most frequently used are GET and POST methods.
target	Specify the target window or frame where the result of the script will be displayed. It takes values like _blank, _self, _parent etc.
enctype	<p>You can use the enctype attribute to specify how the browser encodes the data before it sends it to the server. Possible values are:</p> <p>application/x-www-form-urlencoded - This is the standard method most forms use in simple scenarios.</p> <p>multipart/form-data - This is used when you want to upload binary data in the form of files like image, word file etc.</p>



## HTML Form Controls :

``

you can use to collect data using HTML

- Text Input Controls
- Checkboxes Controls
- Radio Box Controls
- Select Box Controls
- File Select boxes
- Hidden Controls
- Clickable Buttons
- Submit and Reset Button

### Text Input Controls:-

There are three types of text input used on forms:

- 1) **Single-line text input controls** - This control is used for items that require only one line of user input, such as search boxes or names. They are created using HTML `<input>` tag.

**`<input type="text">`** defines a one-line input field for **text input**:

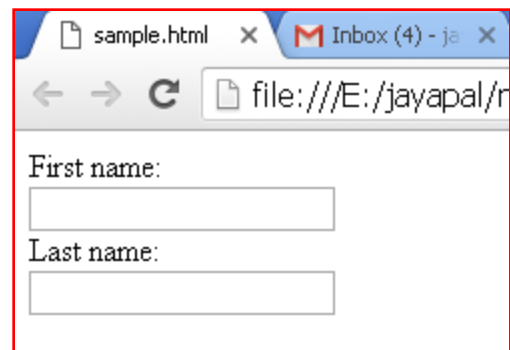
#### Example:

```
<form>
First name:

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

Last name:

<input type="text" name="lastname">
</form>
```



- 2) **Password input controls** - This is also a single-line text input but it masks the character as soon as a user enters it. They are also created using HTML `<input>` tag.

#### Input Type Password

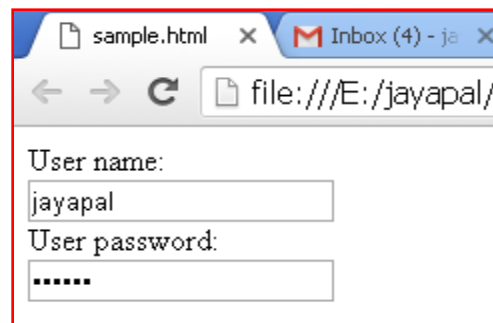
**`<input type="password">`** defines a **password field**:

```
<form>
User name:

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

User password:

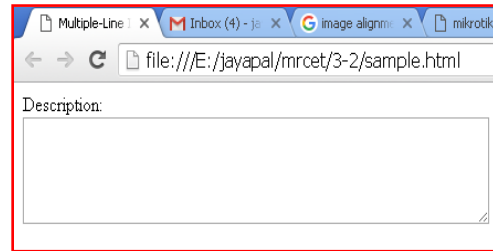
<input type="password" name="psw">
</form>
```



**3)Multi-line text input controls** - This is used when the user is required to give details that may be longer than a single sentence. Multi-line input controls are created using HTML `<textarea>` tag.

```
<!DOCTYPE html>
<html>
 <head>
 <title>Multiple-Line Input Control</title>
 </head>
 <body>
 <form> Description:

 <textarea rows="5" cols="50" name="description"> Enter description here... </textarea>
 </form>
 </body>
</html>
```



### Checkboxes Controls:-

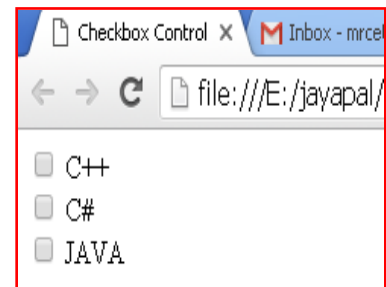
Checkboxes are used when more than one option is required to be selected. They are also created using HTML `<input>` tag but type attribute is set to checkbox.

Here is an example HTML code for a form with two checkboxes:

```
<!DOCTYPE html>
<html> <head> <title>Checkbox Control</title> </head>
<body>
 <form>
 <input type="checkbox" name="C++" value="on"> C++

 <input type="checkbox" name="C#" value="on"> C#

 <input type="checkbox" name="JAVA" value="on"> JAVA
 </form>
</body> </html>
```



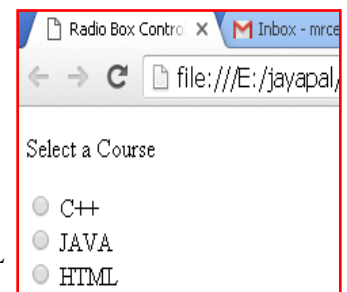
### Radio Button Control:-

Radio buttons are used when out of many options, just one option is required to be selected. They are also created using HTML `<input>` tag but type attribute is set to radio.

```
<!DOCTYPE html>
<html> <head> <title>Radio Box Control</title> </head>
 <body> <p>Select a Course</p>
 <form>
 <input type="radio" name="subject" value="C++"> C++

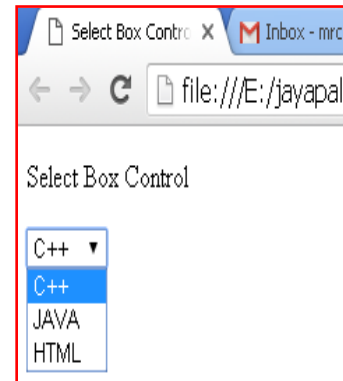
 <input type="radio" name="subject" value="JAVA"> JA VA

 <input type="radio" name="subject" value="HTML"> HTML
 </form>
 </body> </html>
```



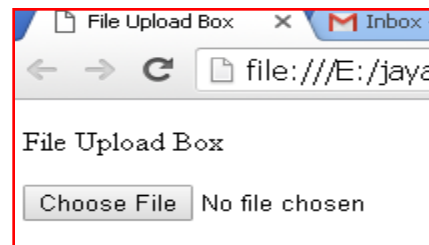
**Select Box Controls :-** A select box, also called drop down box which provides option to list down various options in the form of drop down list, from where a user can select one or more options.

```
<!DOCTYPE html>
<html>
<head>
 <title>Select Box Control</title>
</head>
<body>
 <form>
 <select name="dropdown">
 <option value="C++" selected>C++</option>
 <option value="JA VA">JA VA</option>
 <option value="HTML">HTML</option>
 </select>
 </form>
</body>
</html>
```



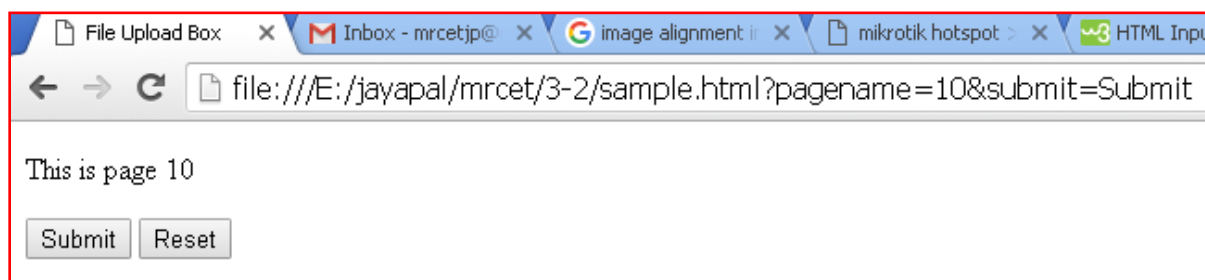
**File Select boxes:-** If you want to allow a user to upload a file to your web site, you will need to use a file upload box, also known as a file select box. This is also created using the `<input>` element but type attribute is set to **file**.

```
<!DOCTYPE html>
<html>
<head>
 <title>File Upload Box</title>
</head>
<body>
 <p>File Upload Box</p>
 <form>
 <input type="file" name="fileupload" accept="image/*" />
 </form>
</body>
</html>
```



**Hidden Controls:-** Hidden form controls are used to hide data inside the page which later on can be pushed to the server. This control hides inside the code and does not appear on the actual page. For example, following hidden form is being used to keep current page number. When a user will click next page then the value of hidden control will be sent to the web server and there it will decide which page will be displayed next based on the passed current page.

```
<html> <head> <title>File Upload Box</title> </head>
 <body>
 <form>
 <p>This is page 10</p>
 <input type="hidden" name="pagename" value="10" />
 <input type="submit" name="submit" value="Submit" />
 <input type="reset" name="reset" value="Reset" />
 </form>
 </body> </html>
```



### Button Controls:-

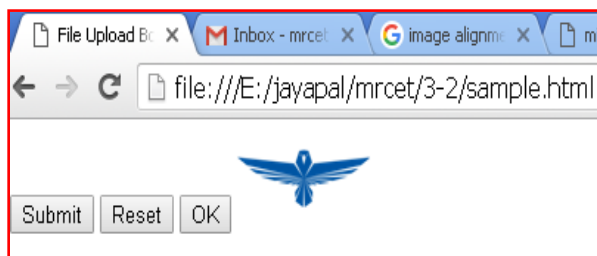
There are various ways in HTML to create clickable buttons. You can also create a clickable button using `<input>` tag by setting its type attribute to **button**. The type attribute can take the following values:

Type	Description
submit	This creates a button that automatically submits a form.
reset	This creates a button that automatically resets form controls to their initial values.
button	This creates a button that is used to trigger a client-side script when the user clicks that button.
image	This creates a clickable button but we can use an image as background of the button.

```

<!DOCTYPE html>
<html>
<head>
 <title>File Upload Box</title>
</head>
<body>
 <form>
 <input type="submit" name="submit" value="Submit" />
 <input type="reset" name="reset" value="Reset" />
 <input type="button" name="ok" value="OK" />
 <input type="image" name="imagebutton" src="test1.png" />
 </form>
</body> </html>

```



**HTML frames:** These are used to divide your browser window into multiple sections where each section can load a separate HTML document. A collection of frames in the browser window is known as a frameset. The window is divided into frames in a similar way the tables are organized: into rows and columns.

To use frames on a page we use `<frameset>` tag instead of `<body>` tag. The `<frameset>` tag defines, how to divide the window into frames. The **rows** attribute of `<frameset>` tag defines horizontal frames and **cols** attribute defines vertical frames. Each frame is indicated by `<frame>` tag and it defines which HTML document shall open into the frame.

**Note:** HTML **<frame>** Tag. **Not Supported in HTML5.**

```
<frameset cols="25%,50%,25%">
 <frame src="frame_a.htm">
 <frame src="frame_b.htm">
 <frame src="frame_c.htm">
</frameset>
```

```
<!DOCTYPE html>
<html>
 <head>
 <title>Page Title</title>
 </head>
 <body>
 <iframe src="sample1.html" height="400" width="400" frameborder="1">
 <h1>This is a Heading</h1>
 <p>This is a paragraph.</p>
 </iframe>
 </body>
</html>
```



## CSS stands for Cascading Style Sheets

CSS describes **how HTML elements are to be displayed on screen, paper, or in other media.**

CSS **saves a lot of work.** It can control the layout of multiple web pages all at once.

CSS can be added to HTML elements in 3 ways:

- **Inline** - by using the style attribute in HTML elements
- **Internal** - by using a <style> element in the <head> section
- **External** - by using an external CSS file

### Inline CSS

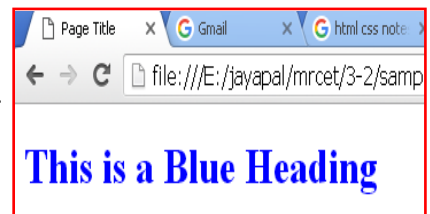
An inline CSS is used to apply a unique style to a single HTML element.

An inline CSS uses the style attribute of an HTML element.

This example sets the text color of the <h1> element to blue:

```
<h1 style="color:blue;">This is a Blue Heading</h1>
```

```
<html>
 <head>
 <title>Page Title</title>
 </head>
 <body>
 <h1 style="color:blue;">This is a Blue Heading</h1>
 </body>
</html>
```



**Internal CSS:** An internal CSS is used to define a style for a single HTML page. An internal CSS is defined in the <head> section of an HTML page, within a <style> element:

```
<html>
 <head>
 <style>
 body {background-color: powderblue;}
 h1 {color: blue;}
 p {color: red;}
 </style>
 </head>
 <body>
 <h1>This is a heading</h1>
 <p>This is a paragraph.</p>
 </body>
</html>
```

### External CSS:-

An external style sheet is used to define the style for many HTML pages. **With an external style sheet, you can change the look of an entire web site, by changing one file!** To use an external style sheet, add a link to it in the <head> section of the HTML page:

```
<html>
 <head>
 <link rel="stylesheet" href="styles.css">
 </head>
 <body>
 <h1>This is a heading</h1>
 <p>This is a paragraph.</p>
 </body>
</html>
```

An external style sheet can be written in any text editor. The file must not contain any HTML code, and must be saved with a **.css extension**.

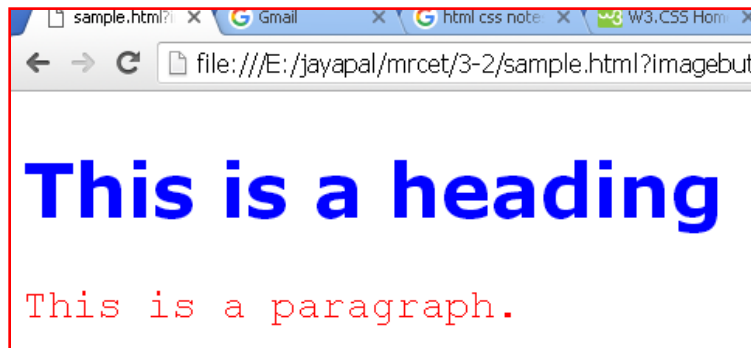
Here is how the "styles.css" looks:

```
body { background-color: powderblue; }
h1 { color: blue; }
p { color: red; }
```



**CSS Fonts:** The CSS **color** property defines the text color to be used.  
 The CSS **font-family** property defines the font to be used.  
 The CSS **font-size** property defines the text size to be used.

```
<html>
<head>
<style>
h1 {
 color: blue;
 font-family: verdana;
 font-size: 300%;
}
p {
 color: red;
 font-family: courier;
 font-size: 160%;
}
</style>
</head>
<body>
<h1>This is a heading</h1>
<p>This is a paragraph.</p>
</body>
</html>
```



**CSS Border:** The CSS **border** property defines a border around an HTML element.

**CSS Padding:** The CSS **padding** property defines a padding (space) between the text and the border.

**CSS Margin:** The CSS **margin** property defines a margin (space) outside the border.

```
<html> <head>
<style>
h1 {
 color: blue;
 font-family: verdana;
 font-size: 300%; }
p {
 color: red; font-size: 160%; border: 2px solid powderblue; padding: 30px; margin: 50px; }
</style>
</head>
<body>
<h1>This is a heading</h1>
<p>This is a paragraph.</p>
</body>
</html>
```



## JavaScript:

### What is JavaScript?

Java Script is one popular scripting language over internet. Scripting means a small sneak (piece). It is always independent on other languages.

JavaScript is most commonly used as a client side scripting language. This means that JavaScript code is written into an HTML page. When a user requests an HTML page with JavaScript in it, the script is sent to the browser and it's up to the browser to do something with it.

### Difference between JavaScript and Java

JavaScript	Java
Cannot live outside a Web page	Can build stand-alone applications or live in a Web page as an <i>applet</i> .
Doesn't need a compiler	Requires a compiler
Knows all about your page	Applets are dimly aware of your Web page.
Untyped	Strongly typed
Somewhat object-oriented	Object-oriented

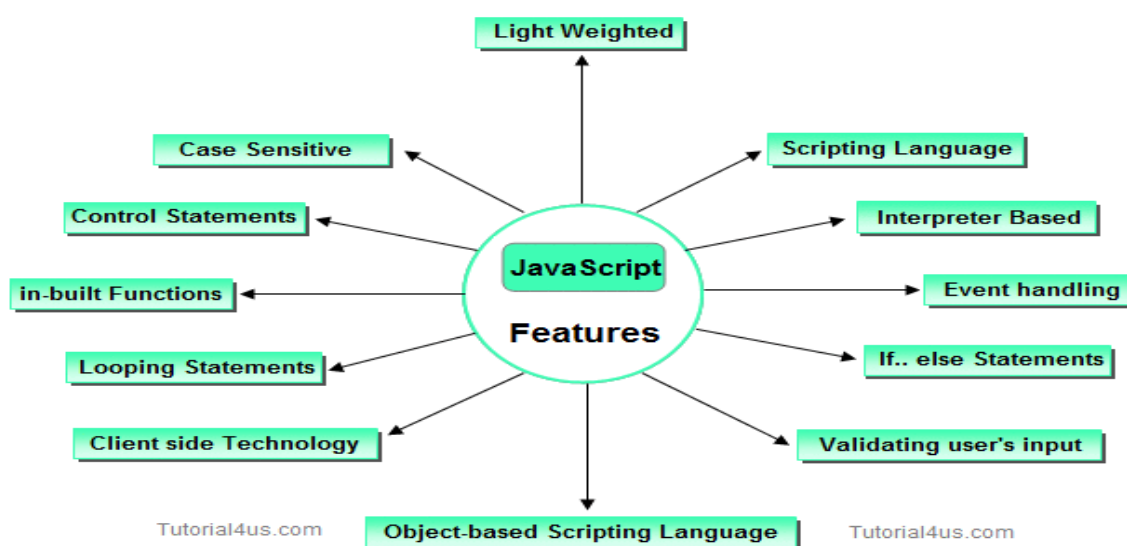
There are no relationship between in java & java script. Java Script is a scripting language that always dependent in HTML language. It used to css commands. It is mainly used to creating DHTML pages & validating the data. This is called client side validations.

### Why we Use JavaScript?

Using HTML we can only design a web page but you can not run any logic on web browser like addition of two numbers, check any condition, looping statements (for, while), decision making statement (if-else) at client side. All these are not possible using HTML So for perform all these task at client side you need to use JavaScript.

### Features of JavaScript

JavaScript is a client side technology, it is mainly used for gives client side validation, but it have lot of features which are given below;





→ **Java script is object based oriented language.**

Inheritance is does not support in JavaScript, so it is called object based oriented language.

→ JavaScript was developed by Netscape (company name) & initially called **live script**.

Later Microsoft developed & adds some features live script then it is called “**Jscript**”.

Jscript is nothing but **Java script**. We cannot create own classes in java script.

→ Java script is designed to **add interactivity to HTML pages**. It is usually embedded directly into html pages.

→ Java script is mainly useful to improve designs of WebPages, **validate form** data at client side, detects (find) visitor’s browsers, create and use to cookies, and much more.

→ Java script is also called **light weight programming language**, because Java script is return with very simple syntax. Java script is containing executable code.

→ Java script is also called **interpreted language**, because script code can be executed without preliminary compilation.

→ It Handling **dates, time, onSubmit, onLoad, onClick, onMouseOver & etc**.

→ JavaScript is **case sensitive**.

→ Most of the javascript control statements syntax is same as syntax of controlstatements in C language.

→ An important part of JavaScript is the ability to create new functions within scripts.

Declare a function in JavaScript using **function** keyword.

**Creating a java script:** - html script tag is used to script code inside the html page.

```
<script> </script>
```

The script is containing **2 attributes**. They are

**1) Language attribute:** -

It represents name of scripting language such as JavaScript, VbScript.

```
<script language=“JavaScript”>
```

**2) Type attribute:** - It indicates MIME (multi purpose internet mail extension) type of scripting code. It sets to an alpha-numeric MIME type of code.

```
<script type=“text / JavaScript”>
```

**Location of script or placing the script:** - Script code can be placed in both head & body section of html page.

**Script in head section**

```
<html>
<head>
<script type="text / JavaScript">
 Script code here
</script>
</head>
<body>
</body>
</html>
```

**Script in body section**

```
<html>
<head>
</head>
<body>
 <script type="text / JavaScript">
 Script code here
 </script>
</body>
</html>
```

**Scripting in both head & body section:** - we can create unlimited number of scripts inside the same page. So we can locate multiple scripts in both head & body section of page.

**Ex:** - <html>  
 <head>  
 <script type="text / JavaScript">  
 Script code here  
 </script>  
 </head>  
 <body>  
 <script type="text / JavaScript">  
 Script code here  
 </script>  
 </body>  
</html>

**Program:** -

```
<html>
<head>
<script language="JavaScript">
document.write("hai my name is Kalpana")
</script>
</head>
<body text="red">
<marquee>
<script language="JavaScript">
document.write("hai my name is Sunil Kumar Reddy")
</script> </marquee>
</body>
</html>
```

**O/P:** - hai my name is Kalpana

hai my name is Sunil Kumar Reddy

**document.write** is the proper name of object.

→ There are 2 ways of executing script code

- 1) direct execute
- 2) to execute script code dynamically

**Reacts to events:** - JavaScript can be set to execute when something happens. When the page is finished loading in browser window (or) when the user clicks on html element dynamically.

**Ex: -**

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<script language="JavaScript">
function myf()
{
document.write("Hai Kalpana")
}
</script>
</HEAD>
<BODY>
to execute script code:
<input type="button" value="click me" onclick="myf()">
To execute script code:
<input type="button" value="touch me" onmouseover="myf()">
</BODY>
</HTML>
```

**O/P: - to execute script code:**

click me

**To execute script code:**

touch me

**Creating external script:** - some times you might want to run same script on several pages without having to write the script on each page. To simplify this, write external script & save .js extension. To use external script specify .js file in src attribute of script tag.

**Note: - external script can not contain script tag.**

**save: - external.js**

```
document.write("this is external script code 1 "+"
");
document.write("this is external script code 2 "+"
");
document.write("this is external script code 3 "+"
");
document.write("this is external script code 4 ");
```

```
<HTML> <BODY>
<script language="JavaScript">
document.write("this is document code 1 "+"
");
document.write("this is document code 2 "+"
");
</script>
<script src="external.js">
</script>
</BODY>
</HTML>
```

**O/P: -**

this is document code 1  
this is document code 2  
this is external script code 1  
this is external script code 2  
this is external script code 3  
this is external script code 4

**JavaScript syntax rules:** - JavaScript is case sensitive language. In this upper case lower case letters are differentiated (not same).

**Ex:** - a=20;  
A=20;

Those the variable name 'a' is different from the variable named 'A'.

**Ex:** - myf( ) // correct  
myF( ) // incorrect

→ ; is optional in general JavaScript.

**Ex:** - a=20 // valid  
b=30 // valid  
A=10; b=40; // valid

However it is required when you put multiple statements in the same line.

→ JavaScript ignore white space. In java script white space, tag space & empty lines are not preserved.  
→ To display special symbols we use \.

**Comment lines:** - comments lines are not executable.

// single line comment  
/\* this is multi line comment \*/

**Declaring variable:** - variable is a memory location where data can be stored. In java script variables with any type of data are declared by using the keyword 'var'. All keywords are small letters only.

var a; a=20;  
var str; str= "Sunil";  
var c; c='a';  
var d; d=30.7;

But the keyword is not mandatory when declare of the variable.

c; → not valid. In this solution var keyword must be declared.  
→ During the script, we can change value of variable as well as type of value of variable.

**Ex:** -  
a=20;  
a=30.7;

**JavaScript functions:** - in java script functions are created with the keyword 'function' as shown below

**Syntax:** - function funname( )  
{  
-----  
}

Generally we can place script containing function head section of web page. There are 2 ways to call the function.

- 1) direct call function
- 2) Events handlers to call the function dynamically.

1→ We can pass data to function as argument but that data will be available inside the function.

**Ex: -**

```

<HTML>
<HEAD>
<TITLE> Function direct call</TITLE>
<script language="JavaScript">
function add(x,y)
{
 z=x+y
 return z
}
</script>

```

```

</HEAD>
<BODY>
<script>
var r=add(30,60)
document.write("addition is :"+r);
</script>
</BODY>
</HTML>

```

**O/P: - addition is :90**

**2→** to add dynamical effects, java script provide a list of events that call function dynamically. Hare each event is one attribute that always specified in html tags.

```

attrname="attrval"
eventName="funname()"

```

**Ex: -**

```

<HTML>
<HEAD>
<TITLE> Function dynamically</TITLE>
<script language="JavaScript">
function add()
{
 x=20
 y=30
 z=x+y
 document.write("addition is :"+z);
}
</script>

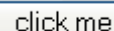
```

```

</HEAD>
<BODY> to call function:
<input type="button" value="click hare"
onclick="add()">
</script>
</BODY>
</HTML>

```

**O/P: -** to call function:  
addition is :90



**EVENT HANDLERS:** Events are not case sensitive.

**Java script events: -**

<u>Attribute</u>	<u>The event occurs when...</u>
onclick	mouse click an object
ondblclick	mouse double clicks
onmouseover	a mouse cursor on touch here
onmousedown	a mouse button is pressed
onmousemove	the mouse is moved
onmouseout	the mouse is moved out an element
onmouseup	a mouse button is released
onkeydown	a keyboard key is pressed
onkeypress	a keyboard key is pressed or held down
onkeyup	a keyboard key is released
onfocus	an elements get focus
onblur	an element loses focus
onchange	the content of a field change
onselect	text is selected
onload	a page or an image is finished loading
onunload	the user exist the page

<b>onerror</b>	<b>an error occurs when loading a document or an image</b>
<b>onabort</b>	<b>loading an image is interrupted</b>
<b>onresize</b>	<b>a window or frame is resized</b>
<b>onreset</b>	<b>the reset button is pressed</b>
<b>onsubmit</b>	<b>the submit button is clicked</b>

**Ex: -**

```

<HTML>
<HEAD>
<TITLE> Mouse Events </TITLE>
<script language="JavaScript">
function add()
{
a=55
b=45
c=a+b
document.write("addition is :"+c)
}
</script>
</HEAD>
<BODY>
<b onclick="add()">
to call function click here :

<b onmouseover="add()">
to call function touch here :

<b ondblclick="add()">
to call function double click here :


```

```


<b onmousemove="add()">
to call function cursor move here :

<b onmouseup="add()">
to call function cursor up here :

<b onmouseout="add()">
to call function cursor out here :

</BODY>
</HTML>

```

**O/P: -**

```

to call function click here :
to call function touch here :
to call function double click here :
 addition is :100
to call function cursor move here :
to call function cursor up here :
to call function cursor out here :

```

**Program: -**

```

<HTML>
<HEAD>
<TITLE> display student name </TITLE>
<script language="JavaScript">
function disp()
{
// access from data
var name=window.document.student.sname.value
// (or) var name=window.document.getElementById("snameid").value
//checking name
if(name=""||!isNaN(name)||!isNaN(name.charAt(0)))
 window.alert("sname you entered is invalid")
else
 document.write("sname you have entered is : "+name);
}

```

```

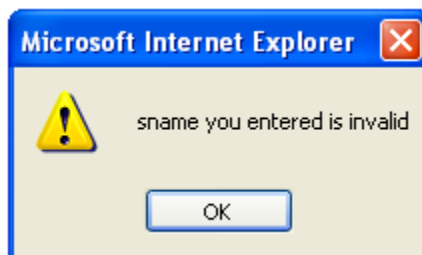
</script>
</HEAD>
<BODY>
<form name="student">
Enter Student name:
<input type="text" name="sname" id="snameid" value="enter" onblur="disp()">
</form>
</BODY>
</HTML>

```

O/P: -

Enter Student name:

Enter Student name:   
sname you have entered is : true



**Popup boxes:** - popup (arises) box is a small window that always shown before opening the page. The purpose of popup box is to write message, accept some thing from user. Java script provides 3 types of popup boxes. They are **1) alert 2) Confirm. 3) Prompt.**

### 1) alert popup box :-

Alert box is a very frequently useful to send or write cautionary messages to end use alert box is created by alert method of window object as shown below.

**Syntax: - window – alert (“message”);**

When alert popup, the user has to click ok before continue browsing.

**Ex: -**

```

<html>
<head>
<title> alert box </title>
<script language="JavaScript">
function add()
{
a=20
b=40
c=a+b

```

```

window.alert("This is for addition of 2
no's")
document.write("Result is: "+c)
}
</script>
</head>
<body onload="add()">
</body>
</html>

```

O/P: -



Result is: 60

**2) confirm popup box:-**

This is useful to verify or accept some thing from user. It is created by confirm method of window object as shown below.

**Syntax:-** `window.confirm ("message?");`

When the confirm box pop's up, user must click either ok or cancel buttons to proceed. If user clicks ok button it returns the boolean value true. If user clicks cancel button, it returns the boolean value false.

**Ex: -**

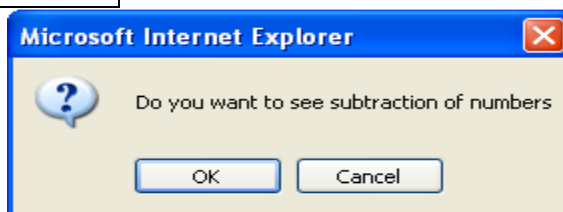
```
<HTML>
<HEAD>
<TITLE> Confirm </TITLE>
<script>
function sub()
{
a=50
b=45
c=a-b
x=window.confirm("Do you want to see
subtraction of numbers")
if(x==true)
{
```

```
document.write("result is :"+c)
}
else
{
document.write("you clicked cancel button")
}
}
</script>
</HEAD>
<BODY onload="sub()">
to see the o/p in pop up box:
</BODY>
</HTML>
```

**O/P: -**

to see the o/p in pop up box:

result is :5



**3) Prompt popup box:-** It is useful to accept data from keyboard at runtime. Prompt box is created by prompt method of window object.

**Syntax:-** `window.prompt ("message", "default text");`

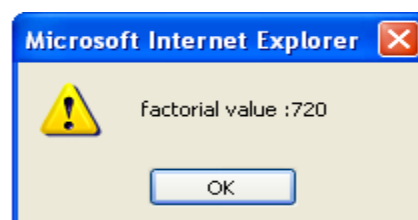
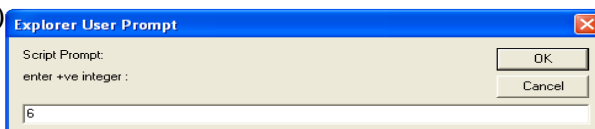
When prompt dialog box arises user will have to click either ok button or cancel button after entering input data to proceed. If user click ok button it will return input value. If user click cancel button the value "null" will be returned.

**Ex: -**

```
<HTML>
<HEAD>
<TITLE> Prompt </TITLE>
<script>
function fact()
{
var b=window.prompt("enter +ve integer
:", "enter here")
var c=parseInt(b)
a=1
for(i=c; i>=1; i--)
```

```
{
a=a*i
}
window.alert("factorial value :"+a)
}
</script>
</HEAD>
<BODY onload="fact()">
</BODY>
</HTML>
```

**O/P: -**





**FORM VALIDATION:**

When we create forms, providing form validation is useful to ensure that your customers enter valid and complete data. For example, you may want to ensure that someone inserts a valid e-mail address into a text box, or perhaps you want to ensure that someone fills in certain fields.

We can provide custom validation for your forms in two ways: server-side validation and client-side validation.

**SERVER-SIDE VALIDATION**

In the server-side validation, information is being sent to the server and validated using one of server-side languages. If the validation fails, the response is then sent back to the client, page that contains the web form is refreshed and a feedback is shown. This method is secure because it will work even if JavaScript is turned off in the browser and it can't be easily bypassed by malicious users. On the other hand, users will have to fill in the information without getting a response until they submit the form. This results in a slow response from the server.

The exception is validation using Ajax. Ajax calls to the server can validate as you type and provide immediate feedback. Validation in this context refers to validating rules such as username availability.

**Server side validation** is performed by a web server, after input has been sent to the server.

**CLIENT-SIDE VALIDATION**

Server-side validation is enough to have a successful and secure form validation. For better user experience, however, you might consider using client-side validation. This type of validation is done on the client using script languages such as JavaScript. By using script languages user's input can be validated as they type. This means a more responsive, visually rich validation.

With client-side validation, form never gets submitted if validation fails. Validation is being handled in JavaScript methods that you create (or within frameworks/plugins) and users get immediate feedback if validation fails.

Main drawback of client-side validation is that it relies on JavaScript. If users turn JavaScript off, they can easily bypass the validation. This is why validation should always be implemented on both the client and server. By combining server-side and client-side methods we can get the best of the two: fast response, more secure validation and better user experience.

**Client side validation** is performed by a web browser, before input is sent to a web server.

Validation can be defined by many different methods, and deployed in many different ways.

**Simple Example:**

```
<html>
<head>
<title>Form Validation</title>
<script type="text/javascript">
```

```
<!--
// Form validation code will come here.
function validate()
{
var n = document.myForm.Name.value;
if(n == "" || (!isNaN(parseInt(n))) || n.length < 3 || n.length >= 8)
{
alert("Please enter valid name and minimum length 3 characters and maximum length 8
characters !");
document.myForm.Name.focus();
return false;
}

var emailID = document.myForm.EMail.value;
if(emailID == "")
{
alert("Please provide your Email!");
document.myForm.EMail.focus() ;
return false;
}

atpos = emailID.indexOf("@");
dotpos = emailID.lastIndexOf(".");
if (atpos < 1 || (dotpos - atpos < 2))
{
alert("Please enter correct email ID")
document.myForm.EMail.focus() ;
return false;
}

var z = document.myForm.Zip.value;
if(z == "" ||isNaN(z) || z.length != 6)
{
alert("Please provide a zip in the format #####.");
document.myForm.Zip.focus() ;
return false;
}

var c = document.myForm.Country.value;
if(c == "-1")
{
```

```
alert("Please provide your country!");
return false;
}
return(true);
}

//-->
</script>
</head>
<body bgcolor="bisque">
<h1><p align="center"> Application Form Validation Using JavaScript</p></h1>
<form action="reg.html" name="myForm" onsubmit="return(validate());">
<table cellspacing="5" cellpadding="5" align="center" border="5" width="438">
<tr>
<td align="right">Name</td>
<td><input type="text" name="Name" size="50" /></td>
</tr>
<tr>
<td align="right">EMail</td>
<td><input type="text" name="EMail" size="50" /></td>
</tr>
<tr>
<td align="right">Zip Code</td>
<td><input type="text" name="Zip" size="50" /></td>
</tr>
<tr>
<td align="right">Country</td>
<td>
<select name="Country">
<option value="-1" selected>[choose yours]</option>
<option value="1">INDIA</option>
<option value="2">UK</option>
<option value="3">USA</option>
</select>
</td>
</tr>
<tr>
<td align="right"></td>
<td><input type="submit" value="Submit" /></td>
</tr>
 </table> </form> </body> </html>
```

## Output:

This page says: Please enter valid name and minimum length 3 characters and maximum length 8 characters!

Application Form Validation Using JavaScript

Name	mr
EMail	
Zip Code	
Country	[choose yours] ▼
	Submit

This page says: Please provide your Email!

Application Form Validation Using JavaScript

Name	mrcet
EMail	
Zip Code	
Country	[choose yours] ▼
	Submit

This page says: Please enter correct email ID

Application Form Validation Using JavaScript

Name	mrcet
EMail	mrcet@gmail
Zip Code	
Country	[choose yours] ▼
	Submit

This page says: Please provide a zip in the format #####

Application Form Validation Using JavaScript

Name	mrcet
EMail	mrcet@gmail.com
Zip Code	
Country	[choose yours] ▼
	Submit

This page says: Please provide your country!

Application Form Validation Using JavaScript

Name	mrcetcs
EMail	mrcet@gmail.com
Zip Code	50040
Country	[choose yours] ▼
	Submit

Application Form Validation Using JavaScript

Name	mrcetcs
EMail	mrcet@gmail.com
Zip Code	50040
Country	INDIA ▼
	Submit

## UNIT – II

### TOPICS:

#### Introduction to PHP

- Declaring Variables
- Data Types
- Operators
- Control Structures
- Functions
- Reading data from WEB form controls like text boxes, radio buttons, lists etc..
- Handling File Uploads
- Handling Sessions and Cookies

#### Introduction to XML

- Basic XML document
- Presenting XML
- Document Type Definition(DTD)
- XML Schemas
- Document Object Model(DOM)
- Introduction to XHTML
- Using XML Processors: DOM and SAX

## PHP INTRODUCTION

PHP started out as a small open source project that evolved as more and more people found out how useful it was. **Rasmus Lerdorf** unleashed the first version of PHP way back in **1994**.

- PHP is a recursive acronym for "**PHP: Hypertext Preprocessor**".
- PHP is a **server side scripting language** that is embedded in HTML. PHP scripts are executed on the server
- It is used to manage dynamic content, databases, session tracking, even build entire e-commerce sites.
- PHP supports many databases (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, Generic ODBC, Microsoft SQL Server , etc.)
- PHP is an open source software.
- PHP is pleasingly zippy in its execution, especially when compiled as an Apache module on the Unix side. The MySQL server, once started, executes even very complex queries with huge result sets in record-setting time.
- PHP supports a large number of major protocols such as POP3, IMAP, and LDAP.
- PHP is forgiving: PHP language tries to be as forgiving as possible.
- **PHP Syntax is C-Like.**

#### Common uses of PHP:

PHP performs system functions, i.e. from files on a system it can create, open, read, write, and close them. **The other uses of PHP are:**

- PHP can handle forms, i.e. gather data from files, save data to a file, thru email you can send data, return data to the user.
- You add, delete, and modify elements within your database thru PHP.
- Access cookies variables and set cookies.
- Using PHP, you can restrict users to access some pages of your website.
- It can encrypt data.

**Characteristics of PHP:**

- Simplicity
- Efficiency
- Security
- Flexibility
- Familiarity

All PHP code must be included inside one of the three special markup tags are recognized by the PHP Parser.

```
<?php PHP code goes here ?>
<? PHP code goes here ?>
<script language="php"> PHP code goes here </script>
```

Most common tag is the **<?php...?>**

**SYNTAX OVERVIEW:**

**Canonical PHP tags** *The most universally effective PHP tag style is:*

```
<?php...?>
```

**Short-open (SGML-style) tags** *Short or short-open tags look like this:*

```
<?...?>
```

**HTML script tags** *HTML script tags look like this:*

```
<script language="PHP">...</script>
```

**PHP - VARIABLE TYPES**

The main way to store information in the middle of a PHP program is by using a **variable**. Here are the most important things to know about variables in PHP.

- A variable is used to store information.
- All variables in PHP are denoted with a leading dollar sign (\$).
- The value of a variable is the value of its most recent assignment.
- Variables are assigned with the = **operator**, with the variable on the left-hand side and the expression to be evaluated on the right.
- Variables can, but do not need, to be declared before assignment.
- Variables used before they are assigned have default values.
- PHP does a good job of **automatically converting types from one to another** when necessary.
- PHP variables are Perl-like.

**Syntax:** \$var\_name = value;

**Eg:** creating a variable containing a string, and a variable containing a number:

```
<?php
$txt="HelloWorld!";
$x=16;
?>
```

**PHP is a Loosely Typed Language:**

- ✓ In PHP, a variable does not need to be declared before adding a value to it.
- ✓ You do not have to tell PHP which data type the variable is
- ✓ PHP automatically converts the variable to the correct data type, depending on its value.

**Naming Rules for Variables**

- ✓ A variable name must start with a letter or an underscore "\_"
- ✓ A variable name can only contain alpha-numeric characters and underscores (a-z, A-Z, 0-9, and \_)
- ✓ A variable name should not contain spaces. If a variable name is more than one word, it should be separated with an underscore (\$my\_string), or with capitalization/Camel notation (\$myString)

**PHP Variables Scope**

In PHP, variables can be declared anywhere in the script. The scope of a variable is the part of the script where the variable can be referenced / used. PHP has three different variable scopes:

- **local**
- **global**
- **static**

**Global and Local Scope**

A variable declared **outside** a function has a GLOBAL SCOPE and can only be accessed outside a function:

**Example**

```
<?php
$x = 5; // global scope
function myTest() {
 // using x inside this function will generate an error
 echo "<p>Variable x inside function is: $x</p>";
}
myTest();
echo "<p>Variable x outside function is: $x</p>";
?>
```

A variable declared **within** a function has a LOCAL SCOPE and can only be accessed within that function:

**Example**

```
<?php
function myTest() {
 $x = 5; // local scope
 echo "<p>Variable x inside function is: $x</p>";
}
myTest(); // using x outside the function will generate an error
echo "<p>Variable x outside function is: $x</p>";
?>
```

You can have local variables with the same name in different functions, because local variables are only recognized by the function in which they are declared.

**PHP The global Keyword**

The global keyword is used to access a global variable from within a function. To do this, use the global keyword before the variables (inside the function):

**Example**

```
<?php
$x = 5; $y = 10;
function myTest() {
 global $x, $y;
 $y = $x + $y;
}
myTest();
echo $y; // outputs 15
?>
```

PHP also stores all global variables in an array called **\$GLOBALS**[*index*]. The *index* holds the name of the variable. This array is also accessible from within functions and can be used to update global variables directly. The example above can be rewritten like this:

**Example**

```
<?php
$x = 5;
$y = 10;
function myTest() {
 $GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y'];
}
myTest();
echo $y; // outputs 15
?>
```

**PHP The static Keyword**

Normally, when a function is completed / executed, all of its variables are deleted. However, sometimes we want a local variable NOT to be deleted. We need it for a further job.

To do this, use the **static** keyword when you first declare the variable:

**Example**

```
<?php
function myTest() {
 static $x = 0;
 echo $x;
 $x++;
}
myTest();
myTest();
myTest(); ?> Output: 0 1 2
```

Then, each time the function is called, that variable will still have the information it contained from the last time the function was called.

**Note:** The variable is still local to the function.



## Variable Naming

Rules for naming a variable is-

- Variable names must begin with a letter or underscore character.
- A variable name can consist of numbers, letters, underscores but you cannot use characters like +, -, %, (, ), . &, etc

**There is no size limit for variables.**

## PHP - Data Types:

PHP has a total of **eight data types** which we use to construct our variables:

- **Integers:** are whole numbers, without a decimal point, like 4195.
  - **Doubles:** are floating-point numbers, like 3.14159 or 49.1.
  - **Booleans:** have only two possible values either true or false.
  - **Strings:** are sequences of characters, like 'PHP supports string operations.'
  - **Arrays:** are named and indexed collections of other values.
  - **Objects:** are instances of programmer-defined classes.
  - **NULL:** is a special type that only has one value: NULL.
  - **Resources:** are special variables that hold references to resources external to PHP (such as database connections).
- Scalar types
- Compound types
- Special types

The first four are simple types, and the next two (arrays and objects) are compound - the compound types can package up other arbitrary values of arbitrary type, whereas the simple types cannot.

## PHP Integers

Integers are **primitive data types**. They are **whole numbers**, without a decimal point, like 4195. They are the simplest type. They correspond to simple whole numbers, both positive and negative {..., -2, -1, 0, 1, 2, ...}.

Integer can be in decimal (base 10), octal (base 8), and hexadecimal (base 16) format. Decimal format is the default, octal integers are specified with a leading 0, and hexadecimal have a leading 0x.

**Ex:** \$v = 12345;

\$var1 = -12345 + 12345;

### notation.php

```
<?php
```

```
$var1 = 31; $var2 = 031; $var3 = 0x31;
```

```
echo "$var1\n$var2\n$var3"; ?>
```

### Output:

```
31
25
49
```

The default notation is the **decimal**. The script shows these three numbers in decimal. In Java and C, if an integer value is bigger than the maximum value allowed, integer overflow happens. PHP works differently. In PHP, the integer becomes a float number. Floating point numbers have greater boundaries. In 32bit system, an integer value size is four bytes. The maximum integer value is 2147483647.

**boundary.php**

```
<?php
$var = PHP_INT_MAX;
echo var_dump($var);
$var++;
echo var_dump($var);
?>
```

We assign a maximum integer value to the \$var variable. We increase the variable by one. And we compare the contents.

**Output:**

```
int(2147483647)
float(2147483648)
```

As we have mentioned previously, internally, the number becomes a floating point value.

**var\_dump():** The PHP var\_dump() function returns the data type and value.

**PHP Doubles or Floating point numbers**

Floating point numbers represent real numbers in computing. Real numbers measure continuous quantities like weight, height or speed. Floating point numbers in PHP can be larger than integers and they can have a decimal point. The size of a float is platform dependent.

We can use various syntaxes to create floating point values.

```
<?php
$a = 1.245;
$b = 1.2e3;
$c = 2E-10;
$d = 1264275425335735;
var_dump($a);
var_dump($b);
var_dump($c);
var_dump($d);
?>
```

The **\$d** variable is assigned a large number, so it is automatically converted to float type.

**Output:**

```
float(1.245)
float(1200)
float(2.0E-10)
float(1264275425340000)
```

This is the output of beside script

**PHP Boolean**

A Boolean represents two possible states: TRUE or FALSE.

```
$x = true; $y = false;
```

Booleans are often used in conditional testing.

```
<?php
$male = False;
$r = rand(0, 1);
$male = $r ? True: False;
if ($male) {
 echo "We will use name John\n";
} else {
 echo "We will use name Victoria\n";
}
?>
```

The script uses a **random integer** generator to simulate our case. `$r = rand(0, 1);`  
The **rand( )** function returns a random number from the given integer boundaries **0 or 1**.  
**\$male = \$r? True: False;**

We use the ternary operator to set a \$male variable. The variable is based on the random \$r value. If \$r equals to **1**, the \$male variable is set to **True**. If \$r equals to **0**, the \$male variable is set to **False**.

## PHP Strings

String is a data type representing textual data in computer programs. Probably the single most important data type in programming.

```
<?php
$a = "PHP ";
$b = 'PERL';
echo $a . $b; ?>
```

**Output: PHP PERL**

**We can use single quotes and double quotes to create string literals.**

The script outputs two strings to the console. The `\n` is a special sequence, a new line.

**The escape-sequence replacements are –**

- `\n` is replaced by the newline character
- `\r` is replaced by the carriage-return character
- `\t` is replaced by the tab character
- `\$` is replaced by the dollar sign itself (\$)
- `\"` is replaced by a single double-quote (")
- `\\` is replaced by a single backslash (\)

## The Concatenation Operator

There is only one string operator in PHP.

The concatenation operator ( `.` ) is used to put two string values together. To concatenate two string variables together, use the concatenation operator:

```
<?php
$txt1="Hello Kalpana!";
$txt2="What a nice day!";
echo $txt1 . " " . $txt2;
?>
```

**O/P: Hello Kalpana! What a nice day!**

## Search for a Specific Text within a String

The **PHP strpos()** function searches for a specific text within a string. If a **match is found**, the function **returns the character position of the first match**. If **no match is found**, it will return **FALSE**. The example below searches for the text "world" in the string "Hello world!":

### Example

```
<?php
echo strpos("Hello world!", "world");
?>
```

**output: 6**

**Tip:** The first character position in a string is 0 (not 1).

## Replace Text within a String

The PHP **str\_replace()** function replaces some characters with some other characters in a string. The example below replaces the text "world" with "Dolly":

**Example**

```
<?php
echo str_replace("world", "Kalpana", "Hello world!");
?>
```

**Output: Hello Kalpana!****The strlen() function:**

The **strlen()** function is used to return the length of a string. Let's find the length of a string:

Eg: <?php

```
echo strlen("Hello world!"); ?> The output of the code above will be: 12
```

**PHP Array**

Array is a complex data type which handles a collection of elements. Each of the elements can be accessed by an index. An array stores multiple values in one single variable. In the following example \$cars is an array. The PHP var\_dump() function returns the data type and value:

**Example**

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
print_r($cars);
var_dump($cars);
?>
```

The **array** keyword is used to create a collection of elements. In our case we have names. The print\_r function prints human readable information about a variable to the console.

**O/P:** Array ( [0] => Volvo [1] => BMW [2] => Toyota )  
array(3) { [0]=> string(5) "Volvo" [1]=> string(3) "BMW" [2]=> string(6) "Toyota" }

**PHP Object**

An object is a data type which stores data and information on how to process that data. In PHP, an object must be explicitly declared. First we must declare a class of object. For this, we use the class keyword. A class is a structure that can contain properties and methods:

**Example**

```
<?php
class Car {
 function Car() {
 $this->model = "VW";
 }
}
$herbie = new Car(); // create an object
echo $herbie->model; // show object properties
?>
```

**Output: VW****PHP NULL**

NULL is a special data type that only has **one value: NULL**. To give a variable the NULL value, simply assign it like this –

**Ex: \$my\_var = NULL;**

The special constant NULL is capitalized by convention, but actually it is case insensitive; you could just as well have typed –

**\$my\_var = null;**

A variable that has been assigned NULL has the following properties –

- It evaluates to FALSE in a Boolean context.
- It returns FALSE when tested with **IsSet()** function.

**Tip:** If a variable is created without a value, it is automatically assigned a value of NULL. Variables can also be emptied by setting the value to NULL:

### Example1

```
<?php
$x = "Hello world!";
$x = null;
var_dump($x);
?>
```

## PHP Resource

The special resource type is not an actual data type. It is the storing of a reference to functions and resources external to PHP. A common **example** of using the resource data type is a **database call**. Resources are handlers to opened files, database connections or image canvas areas. We will not talk about the resource type here, since it is an advanced topic.

### constant() function

As indicated by the name, this function will return the value of the constant. This is useful when you want to retrieve value of a constant, but you do not know its name, i.e. It is stored in a variable or returned by a function. **constant() example**

```
<?php
define("MINSIZE", 50);
echo MINSIZE;
echo constant("MINSIZE"); // same thing as the previous line
?>
```

**Output: 50 50**

Only scalar data (boolean, integer, float and string) can be contained in constants.

## PHP - Operators:

### What is Operator?

Simple answer can be given using expression  $4 + 5$  is equal to 9. **Here 4 and 5 are called operands and + is called operator.** PHP language supports following type of operators.

<b>Arithmetic Operators</b>	<b>Assignment Operators</b>
<b>Increment/Decrement operators</b>	<b>Conditional (or ternary) Operators</b>
<b>Comparison Operators</b>	<b>String Operators</b>
<b>Logical (or Relational) Operators</b>	<b>Array Operators</b>

### Arithmetic Operators:

There are following arithmetic operators supported by PHP language:

Assume variable **A** holds **10** and variable **B** holds **20** then:

Operator	Description	Example
+	Adds two operands	$\$A + \$B$ will give 30
-	Subtracts second operand from the first	$\$A - \$B$ will give -10
*	Multiply both operands	$\$A * \$B$ will give 200
/	Divide numerator by denominator	$\$B / \$A$ will give 2
%	Modulus Operator and remainder of after an integer division	$\$B \% \$A$ will give 0
**	Exponentiation ( $\$x$ to the $\$y$ 'th power)	$\$A ** \$B$

**Increment/Decrement operators**

Operator	Description	Example
++	Increment operator, increases integer value by one	\$A++ - 11 / ++\$A
--	Decrement operator, decreases integer value by one	\$A-- will give 9 / --\$A

**Comparison Operators:**

There are following comparison operators supported by PHP language Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
==	Checks if the value of two operands are equal or not	(\$A==\$B) is not true.
===	Identical(Returns true if \$A is equal to \$B, and they are of the same type)	\$A === \$B
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(\$A != \$B) is true.
<>	Returns true if \$x is not equal to \$y	\$A <> \$B
!==	Not identical (Returns true if \$A is not equal to \$B, or they are not of the same type)	\$A !== \$B
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(\$A > \$B) is not true.
<	Checks if the value of left operand is less (A < B) is true. Than the value of right operand, if yes then condition becomes true.	
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then returns true.	(\$A >= \$B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(\$A <= \$B) is true.

**Logical Operators:**

There are following logical operators supported by PHP language Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
<b>and (or) &amp;&amp;</b>	Called Logical AND operator. If both the operands are true then then condition becomes true.	(\$A and \$B) is true. (\$A && \$B) is true.
<b>or (or)   </b>	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.	(\$A or \$B) is true. (\$A    \$B) is true.
<b>!</b>	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(\$A && \$B) is false.

**Assignment Operators:**

There are following assignment operators supported by PHP language:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	\$C = \$A + \$B

<b>+=</b>	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	\$C += \$A is equivalent to \$C = \$C + \$A
<b>-=</b>	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	\$C -= \$A is equivalent to \$C = \$C - \$A
<b>*=</b>	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	\$C *= \$A is equivalent to \$C = \$C * \$A
<b>/=</b>	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	\$C /= \$A is equivalent to \$C = \$C / \$A
<b>%=</b>	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	\$C %= \$A is equivalent to \$C = \$C % \$A

### Conditional Operator

There is one more operator called conditional operator. This first evaluates an expression for a true or false value and then execute one of the two given statements depending upon the result of the evaluation.

**The conditional operator has this syntax:**

Operator	Description	Example
<b>? :</b>	Conditional Expression	If Condition is true ? Then value X : Otherwise value Y

### PHP String Operators

PHP has two operators that are specially designed for strings.

Operator	Description	Example
<b>.</b>	Concatenation	\$txt1 . \$txt2 (Concatenation of \$txt1 and \$txt2)
<b>.=</b>	Concatenation assignment	\$txt1 .= \$txt2 (Appends \$txt2 to \$txt1)

### PHP Array Operators

The PHP array operators are used to compare arrays.

Operator	Description	Example
<b>+</b>	Union	\$x + \$y (Union of \$x and \$y)
<b>==</b>	Equality	\$x == \$y (Returns true if \$x and \$y have the same key/value pairs)
<b>===</b>	Identity	\$x === \$y (Returns true if \$x and \$y have the same key/value pairs in the same order and of the same types)
<b>!= or &lt;&gt;</b>	Inequality	\$x != \$y or \$x <> \$y Returns true if \$x is not equal to \$y
<b>!==</b>	Non-identity	\$x !== \$y (Returns true if \$x is not identical to \$y)

### Precedence of PHP Operators

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator –

**For example**  $x = 7 + 3 * 2$ ; Here x is assigned 13, not 20 because operator \* has higher precedence than + so it first get multiplied with  $3*2$  and then adds into 7. **Ans:13**

Here operators with the highest precedence appear at the top of the table; those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Unary	! ++ --	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Logical AND	&&	Left to right
Logical OR		Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=	Right to left

Ex: <!DOCTYPE html>

```
<html>
<body>
<?php
$x = 10;
$y = 6;
echo $x + $y;
?>
</body>
</html>
O/P: 16
```

<!DOCTYPE html>

```
<html>
<body>
<?php
$x = 100;
$y = 50;
var_dump($x > $y); //
returns true because $x is
greater than $y
?>
O/P: bool(true)
```

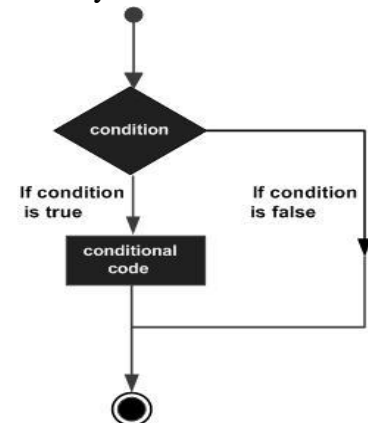
<!DOCTYPE html>

```
<html>
<body>
<?php
$x = 100;
$y = 50;
if ($x == 100 xor $y == 80) {
 echo "Hello world!";
}
?>
O/P: Hello world!
```

## PHP - Decision Making

The if, elseif ...else and switch statements are used to take decision based on the different condition. You can use conditional statements in your code to make your decisions. PHP supports following three decision making statements –

- **if...else statement** – use this statement if you want to execute a set of code when a condition is true and another if the condition is not true
- **elseif statement** – is used with the if...else statement to execute a set of code if **one** of the several condition is true
- **switch statement** – is used if you want to select one of many blocks of code to be executed, use the Switch statement. The switch statement is used to avoid long blocks of if..elseif..else code.



### Syntax

if (condition)

*code to be executed if condition is true;*

else

*code to be executed if condition is false;*

### Example

The following example will output "Have a nice weekend!" if the current day is Friday, Otherwise, it will output "Have a nice day!":

EX: <html>

```
<body>
<?php
$d=date("D");
if ($d=="Fri")
 echo "Have a nice weekend!";
else
 echo "Have a nice day!";
?>
</body> </html>
```



**The If...Else Statement**

If you want to execute some code if a condition is true and another code if a condition is false, use the if...else statement.

**The elseif Statement**

If you want to execute some code if one of the several conditions is true use the elseif statement

<b>Syntax</b>  if ( <i>condition</i> ) <i>code to be executed if condition is true;</i> elseif ( <i>condition</i> ) <i>code to be executed if condition is true;</i> else <i>code to be executed if condition is false;</i>	<b>EX:</b> <html> <body> <?php \$d=date("D"); if (\$d=="Fri") echo "Have a nice weekend!"; elseif (\$d=="Sun") echo "Have a nice Sunday!"; else echo "Have a nice day!"; ?> </body> </html>
<b>Example</b> The following example will output "Have a nice weekend!" if the current day is Friday, and "Have a nice Sunday!" if the current day is Sunday. Otherwise, it will output "Have a nice day!"	

**The Switch Statement**

If you want to select one of many blocks of code to be executed, use the Switch statement. The switch statement is used to avoid long blocks of if..elseif..else code.

<b>Syntax</b>  switch ( <i>expression</i> ) { case <i>label1</i> : <i>code to be executed if expression = label1;</i> break; case <i>label2</i> : <i>code to be executed if expression = label2;</i> break; default: <i>code to be executed if expression is different from both label1 and label2;</i> }	<b>Example</b> The <i>switch</i> statement works in an unusual way. First it evaluates given expression then seeks a lable to match the resulting value. If a matching value is found then the code associated with the matching label will be executed or if none of the lable matches then statement will execute any specified default code.
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**PHP - Loop Types**

Loops in PHP are used to execute the same block of code a specified number of times. PHP supports following four loop types.

- **for** – loops through a block of code a specified number of times.
- **while** – loops through a block of code if and as long as a specified condition is true.
- **do...while** – loops through a block of code once, and then repeats the loop as long as a special condition is true.

- **foreach** – loops through a block of code for each element in an array.

We will discuss about **continue** and **break** keywords used to control the loops execution.

### The for loop statement

The for statement is used when you know how many times you want to execute a statement or a block of statements.

### Syntax

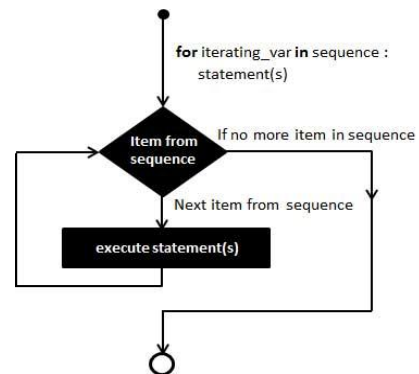
```
for (initialization; condition; increment)
{
 code to be executed;
}
```

The initializer is used to set the start value for the counter of the number of loop iterations. A variable may be declared here for this purpose and it is traditional to name it \$i.

### Example

The following example makes five iterations and changes the assigned value of two variables on each pass of the loop –

```
<html> <body>
 <?php
 $a = 0;
 $b = 0;
 for($i=0; $i<5; $i++)
 {
 $a += 10;
 $b += 5;
 }
 echo ("At the end of the loop a=$a and b=$b");
 ?> </body> </html>
```



This will produce the following result –

At the end of the loop a=50 and b=25

### The while loop statement

The while statement will execute a block of code if and as long as a test expression is true. If the test expression is true then the code block will be executed. After the code has executed the test expression will again be evaluated and the loop will continue until the test expression is found to be false.

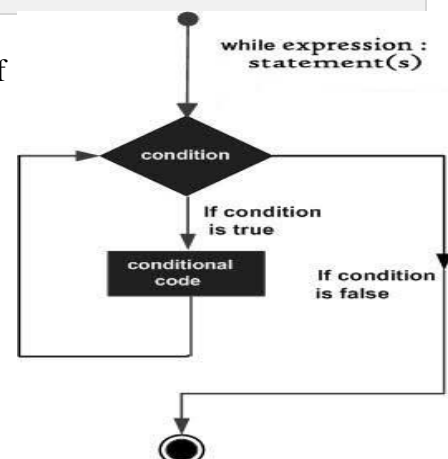
### Syntax

```
while (condition)
{
 code to be executed;
}
```

### Example

This example decrements a variable value on each iteration of the loop and the counter increments until it reaches 10 when the evaluation is false and the loop ends.

```
<html>
<body>
 <?php
 $i = 0;
```



```

$num = 50;
while($i < 10)
{
 $num--;
 $i++;
}
echo ("Loop stopped at i = $i and num = $num");
?> </body> </html>

```

**This will produce the following result –**

Loop stopped at i = 10 and num = 40

### The do...while loop statement

The do...while statement will execute a block of code at least once. It then will repeat the loop as long as a condition is true.

### Syntax

```

do
{
 code to be executed;
}while (condition);

```

### Example

The following example will increment the value of i at least once, and it will continue incrementing the variable i as long as it has a value of less than 10 –

```

<html> }while($i < 10);
<body> echo ("Loop stopped at i = $i");
<?php ?>
 $i = 0; $num = 0; </body>
 do{ </html>
 $i++;

```

**O/P:** Loop stopped at i = 10

### The foreach loop statement

The foreach statement is used to **loop through arrays**. For each pass the value of the current array element is assigned to \$value and the array pointer is moved by one and in the next pass next element will be processed.

### Syntax

```

foreach (array as value)
{
 code to be executed;
}

```

### Example

Try out beside example to list out the values of an array.

```

<html>
<body>
<?php
 $array = array(1, 2, 3, 4, 5);
 foreach($array as $value)
 {
 echo "Value is $value
";
 }
?>
</body> </html>

```

**This will produce the following result –**

Value is 1  
Value is 2  
Value is 3  
Value is 4  
Value is 5

### The break statement

The PHP **break** keyword is used to terminate the execution of a loop prematurely. The **break** statement is situated inside the statement block. It gives you full control and whenever you want to exit from the loop you can come out. After coming out of a loop immediate statement to the loop will be executed.

#### Example

In the following example condition test becomes true when the counter value reaches 3 and loop terminates.

```
<?php
$i = 0;
while($i < 10) {
 $i++;
 if($i == 3)break; }
echo ("Loop stopped at i = $i");
?> O/P: Loop stopped at i=3
```

### The continue statement

The PHP **continue** keyword is used to halt the current iteration of a loop but it does not terminate the loop. Just like the **break** statement the **continue** statement is situated inside the statement block containing the code that the loop executes, preceded by a conditional test. For the pass encountering **continue** statement, rest of the loop code is skipped and next pass starts.

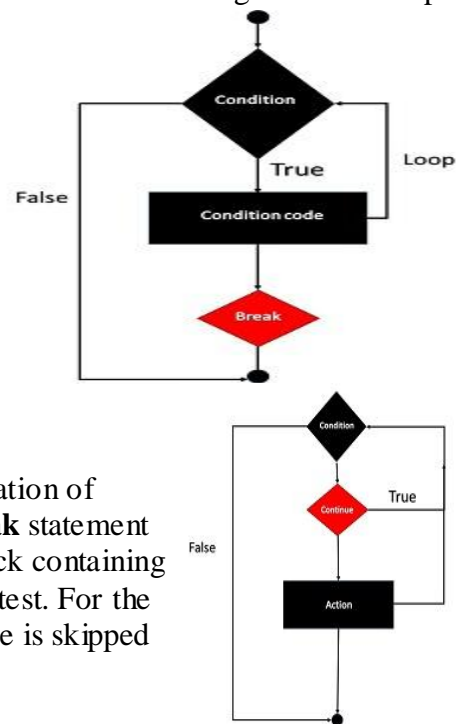
#### Example

In the following example loop prints the value of array but for which condition becomes true it just skip the code and next value is printed.

```
<html>
<body>
<?php
$nos = array(1, 2, 3, 4, 5);
foreach($nos as $value)
{
 if($value == 3)
 continue;
 echo "Value is $value
";
}
?>
</body>
</html>
```

This will produce the following result –

```
Value is 1
Value is 2
Value is 4
Value is 5
```



## PHP – Functions

PHP functions are similar to other programming languages. A function is a piece of code which takes one more input in the form of parameter and does some processing and returns a value. You already have seen many functions like **fopen()** and **fread()** etc. They are built-in functions but PHP gives you option to create your own functions as well.

There are two parts which should be clear to you –

- **Creating a PHP Function**
- **Calling a PHP Function**

In fact you hardly need to create your own PHP function because there are already more than 1000 of built-in library functions created for different area and you just need to call them according to your requirement.

### Creating PHP Function

It's very easy to create your own PHP function. Suppose you want to create a PHP function which will simply write a simple message on your browser when you will call it. Following example creates a function called writeMessage() and then calls it just after creating it.

```
<html> <head>
 <title>Writing PHP Function</title>
</head>
<body>
 <?php
 /* Defining a PHP Function */
 function writeMessage()
 {
```

```
 echo "Have a nice time Kalpana!";
 } /* Calling a PHP Function */
 writeMessage();
 ?>
</body>
</html>
```

**Output:** Have a nice time Kalpana!

### PHP Functions with Parameters

PHP gives you option to pass your parameters inside a function. You can pass as many as parameters you're like. These parameters work like variables inside your function. Following example takes two integer parameters and add them together and then print them.

```
<html>
<head> <title>Writing PHP Function with Parameters</title> </head>
<body>
<?php
 function addFunction($num1, $num2)
 {
 $sum = $num1 + $num2;
 echo "Sum of the two numbers is : $sum";
 }
 addFunction(10, 20);
?> </body> </html>
```

**Output:** Sum of the two numbers is : 30

### Passing Arguments by Reference

It is possible to pass arguments to functions by reference. This means that a reference to the variable is manipulated by the function rather than a copy of the variable's value. Any changes made to an argument in these cases will change the value of the original variable. You can pass an argument by reference by adding an ampersand to the variable name in either the function call or the function definition.

```
<html>
 <head>
 <title>Passing Argument by Reference</title>
 </head>
 <body>
 <?php
 function addFive($num)
 {
```

```
$num += 5;
}
function addSix(&$num)
{
 $num += 6;
}
$orignum = 10;
addFive($orignum);
echo "Original Value is $orignum
";
addSix($orignum);
echo "Original Value is $orignum
";
?>
</body>
</html>
```

**Output:** Original Value is 10  
Original Value is 16

### PHP Functions returning value

A function can return a value using the **return** statement in conjunction with a value or object. **return** stops the execution of the function and sends the value back to the calling code. You can return more than one value from a function using **return array(1,2,3,4)**.

```
<html> <head> <title>Writing PHP Function which returns value</title> </head>
<body>
<?php
 function addFunction($num1, $num2)
 {
 $sum = $num1 + $num2;
 return $sum;
 }
 $return_value = addFunction(10, 20);
 echo "Returned value from the function : $return_value";
?> </body> </html>
```

**Output:** Returned value from the function : 30

### Setting Default Values for Function Parameters

You can set a parameter to have a default value if the function's caller doesn't pass it. Following function prints NULL in case use does not pass any value to this function.

```
<html> <head> <title>Writing PHP Function which returns value</title> </head>
<body>
<?php
 function printMe($param = NULL)
 {
 print $param;
 }
 printMe("This is test");
 printMe();
?>
</body> </html>
```

**Output:** This is test

### Dynamic Function Calls

It is possible to assign function names as strings to variables and then treat these variables exactly as you would the function name itself.

<pre>&lt;html&gt; &lt;head&gt; &lt;title&gt;Dynamic Function Calls&lt;/title&gt; &lt;/head&gt; &lt;body&gt; &lt;?php     function sayHello()     {         echo "Hello&lt;br /&gt;";     }     \$function_holder = "sayHello";     \$function_holder(); ?&gt; &lt;/body&gt; &lt;/html&gt;</pre>	<pre>&lt;html&gt; &lt;head&gt; &lt;title&gt;Dynamic Function Calls&lt;/title&gt; &lt;/head&gt; &lt;body&gt; &lt;?php     function add(\$x,\$y)     {         echo "addition=" . (\$x+\$y);     }     \$function_holder = "add";     \$function_holder(20,30); ?&gt; &lt;/body&gt; &lt;/html&gt;</pre>
<b>Output:</b> Hello	<b>Output:</b> addition=50

### PHP Default Argument Value

The following example shows how to use a default parameter. If we call the function setHeight() without arguments it takes the default value as argument:

#### Example

```
<?php
function setHeight($minheight = 50) {
 echo "The height is : $minheight \t";
}
setHeight(350);
setHeight(); // will use the default value of 50
setHeight(135);
setHeight(80);
?>
```

O/P: 350      50      135      80

## PHP - Web Concepts and Reading data from WEB

### Identifying Browser & Platform

PHP creates some useful **environment variables** that can be seen in the **phpinfo.php** page that was used to setup the PHP environment. One of the environment variables set by PHP is **HTTP\_USER\_AGENT** which identifies the user's browser and operating system.

### Using HTML Form with name validation in PHP: test.php

```
<?php
if($_POST["name"] || $_POST["age"])
{
 if (preg_match("/^[A-Za-z'-]$/",$_POST['name']))
 {
 die ("invalid name and name should be alpha"); }
 echo "Welcome ". $_POST['name']. "
";
```

```
 echo "You are ". $_POST['age']. " years old.";
 exit();
} ?>
<html>
<body>
<form action="<?php $_PHP_SELF ?>" method="POST">
 Name: <input type="text" name="name" />
 Age: <input type="text" name="age" />
 <input type="submit" />
</form> </body> </html>
```

- The PHP default variable **\$\_PHP\_SELF** is used for the PHP script name and when you click "submit" button then same PHP script will be called and will produce following result
- The method = "POST" is used to post user data to the server script.

### PHP Forms and User Input:

The PHP **\$\_GET** and **\$\_POST** variables are used to retrieve information from forms, like user input.

### PHP - GET & POST Methods

There are two ways the browser client can send information to the web server.

- The GET Method
- The POST Method

Before the browser sends the information, it encodes it using a scheme called **URL encoding or URL Parameters**. In this scheme, name/value pairs are joined with equal signs and different pairs are separated by the ampersand.

**name1=value1&name2=value2&name3=value3**

### The GET Method

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the **?** character.

**http://www.sampletest.com/index.htm?name1=value1&name2=value2**

- The GET method produces a long string that appears in your server logs, in the **browser's Location: box**.
- The GET method is restricted to send upto **1024 characters only**.
- Never use GET method if you have **password or other sensitive information** to be sent to the server.
- GET **can't be used to send binary data**, like images or word documents, to the server.
- The PHP provides **\$\_GET** associative array to access all the sent information using GET method.

```
<?php
if($_GET["name"] || $_GET["age"])
{
 echo "Welcome ". $_GET['name']. "
";
 echo "You are ". $_GET['age']. " years old.";
 exit();
}
?> <html> <body>
```



```
<form action="<?php $_PHP_SELF ?>" method="GET">
 Name: <input type="text" name="name" />
 Age: <input type="text" name="age" />
 <input type="submit" />
</form> </body> </html>
```

### The POST Method

The POST method transfers information **via HTTP headers or HTTP Parameters**. The information is encoded as described in case of GET method and put into a header called QUERY\_STRING.

- The POST method does not have any restriction on data size to be sent.
- The POST method can be used to send ASCII as well as binary data.
- The data sent by POST method goes through HTTP header so security depends on HTTP protocol. By using Secure HTTP you can make sure that your information is secure.
- The PHP provides **\$\_POST** associative array to access all the sent information using POST method.

### PHP Form Handling

**Example:** The example below contains an HTML form with two input fields and a submit button:

```
<html>
<body>
<form action="welcome.php" method="post">
Name: <input type="text" name="fname" />
Age: <input type="text" name="age" />
<input type="submit" />
</form>
</body>
</html>
```

When a user fills out the form above and click on the submit button, the form data is sent to a PHP file, called "**welcome.php**": its looks like this:

```
<html> <body>
Welcome <?php echo $_POST["fname"]; ?>!
You are <?php echo $_POST["age"]; ?> years old.
</body>
</html> Output: Welcome Kalpana! You are 29 years old.
```

**\$\_GET** is an array of variables passed to the current script via the **URL parameters**.

**\$\_POST** is an array of variables passed to the current script via the **HTTP POST method**.

### PHP - File Uploading

A PHP script can be used with a HTML form to allow users to upload files to the server. Initially files are uploaded into a temporary directory and then relocated to a target destination by a PHP script.

Information in the **phpinfo.php** page describes the temporary directory that is used for file uploads as **upload\_tmp\_dir** and the maximum permitted size of files that can be uploaded is stated as **upload\_max\_filesize**. These parameters are set into PHP configuration file **php.ini**

**The process of uploading a file follows these steps –**

- The user opens the page containing a HTML form featuring a text files, a browse button and a submit button.
- The user clicks the browse button and selects a file to upload from the local PC.
- The full path to the selected file appears in the text filed then the user clicks the submit button.
- The selected file is sent to the temporary directory on the server.
- The PHP script that was specified as the form handler in the form's action attribute checks that the file has arrived and then copies the file into an intended directory.
- The PHP script confirms the success to the user.

As usual when writing files it is necessary for both temporary and final locations to have permissions set that enable file writing. If either is set to be read-only then process will fail. An uploaded file could be a text file or image file or any document.

**Creating an upload form**

The following HTML code below creates an up loader form. This form is having method attribute set to **post** and **enctype attribute** is set to **multipart/form-data**

There is one global PHP variable called **\$\_FILES**. This variable is an associate double dimension array and keeps all the information related to uploaded file. So if the value assigned to the input's name attribute in uploading form was **file**, then PHP would create following five variables –

- **\$\_FILES['file']['tmp\_name']** the uploaded file in the temporary dir on the web server.
- **\$\_FILES['file']['name']** – the actual name of the uploaded file.
- **\$\_FILES['file']['size']** – the size in bytes of the uploaded file.
- **\$\_FILES['file']['type']** – the MIME type of the uploaded file.
- **\$\_FILES['file']['error']** – the error code associated with this file upload.

**Example: Below example should allow upload images and gives back result as uploaded file information.**

```
<?php
if(isset($_FILES['image']))
{
 $file_name =
$_FILES['image']['name'];
 $file_size=$_FILES['image']['size'];
 $file_type=$_FILES['image']['type'];
}
?>
<html> <body>
 <form action="" method="POST"
enctype="multipart/form-data">
 <input type="file" name="image" />
 <input type="submit"/>
```

```

 Sent file: <?php echo
$_FILES['image']['name']; ?>
 File size: <?php echo
$_FILES['image']['size']; ?>
 File type: <?php echo
$_FILES['image']['type'] ?>

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

Choose File No file chosen

Submit

- Sent file: images.jpg
- File size: 8926
- File type: image/jpeg



## PHP - Cookies

Cookies are text files stored on the client computer and they are kept of use tracking purpose. PHP transparently supports HTTP cookies.

There are three steps involved in identifying returning users –

- Server script sends a set of cookies to the browser. For example name, age, or identification number etc.
- Browser stores this information on local machine for future use.
- When next time browser sends any request to web server then it sends those cookies information to the server and server uses that information to identify the user.

### Setting Cookies with PHP

PHP provided **setcookie()** function to set a cookie. This function requires upto six arguments and should be called before `<html>` tag. For each cookie this function has to be called separately.

```
setcookie(name, value, expire, path, domain, security);
```

Here is the detail of all the arguments –

- **Name** – This sets the name of the cookie and is stored in an environment variable called `HTTP_COOKIE_VARS`. This variable is used while accessing cookies.
- **Value** – This sets the value of the named variable and is the content that you actually want to store.
- **Expiry** – This specify a future time in seconds since 00:00:00 GMT on 1st Jan 1970. After this time cookie will become inaccessible. If this parameter is not set then cookie will automatically expire when the Web Browser is closed.
- **Path** – This specifies the directories for which the cookie is valid. A single forward slash character permits the cookie to be valid for all directories.
- **Domain** – This can be used to specify the domain name in very large domains and must contain at least two periods to be valid. All cookies are only valid for the host and domain which created them.
- **Security** – This can be set to 1 to specify that the cookie should only be sent by secure transmission using HTTPS otherwise set to 0 which mean cookie can be sent by regular HTTP.

Following example will create two cookies **name** and **age** these cookies will be expired after one hour.

```
<?php
 setcookie("name", "KALPANA", time()+3600, "/", "", 0);
 setcookie("age", "36", time()+3600, "/", "", 0);
?>
<html>
<head> <title>Setting Cookies with PHP</title> </head>
<body>
 <?php echo "Set Cookies"?>
</body>
</html>
```

### Accessing Cookies with PHP

PHP provides many ways to access cookies. Simplest way is to use either `$_COOKIE` or `$HTTP_COOKIE_VARS` variables. Following example will access all the cookies set in above example.

```
<html>
 <head> <title>Accessing Cookies with PHP</title> </head>
 <body>
 <?php
 echo $_COOKIE["name"]. "
";
 /* is equivalent to */
 echo $HTTP_COOKIE_VARS["name"]. "
";
 echo $_COOKIE["age"] . "
";
 /* is equivalent to */
 echo $HTTP_COOKIE_VARS["name"] . "
";
 ?>
 </body> </html>
```

You can use **isset()** function to check if a cookie is set or not.

```
<html> <head> <title>Accessing Cookies with PHP</title> </head>
 <body>
 <?php
 if(isset($_COOKIE["name"]))
 echo "Welcome " . $_COOKIE["name"] . "
";
 else
 echo "Sorry... Not recognized" . "
";
 ?>
 </body> </html>
```

### Deleting Cookie with PHP

Officially, to delete a cookie you should call `setcookie()` with the name argument only but this does not always work well, however, and should not be relied on.

It is safest to set the cookie with a date that has already expired –

```
<?php
 setcookie("name", "", time()- 60, "/", "", 0);
 setcookie("age", "", time()- 60, "/", "", 0);
?>
<html>
<head> <title>Deleting Cookies with PHP</title> </head>
<body>
 <?php echo "Deleted Cookies" ?>
</body> </html>
```

### PHP - Session

- When you work with an application, you open it, do some changes, and then you close it. This is much like a Session.
- Session ID is stored as a cookie on the client box or passed along through URL's.
- The values are actually stored at the server and are accessed via the session id from your cookie. On the client side the session ID expires when connection is broken.
- Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc). By default, session variables last until the user closes the browser.
- Session variable values are stored in the 'superglobal' associative array '\$\_SESSION'

### Start a PHP Session

A session is started with the `session_start()` function.

Session variables are set with the PHP global variable: `$_SESSION`.

#### demo\_session1.php

```
<?php
// Start the session
session_start();
?>
<html>
<body>
<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>
</body> </html>
```

#### Modify a PHP Session Variable

To change a session variable, just overwrite it:

```
<?php
session_start();
?>
<html>
<?php
$_SESSION["favcolor"] = "yellow";
print_r($_SESSION); ?> </html>
```

### Get PHP Session Variable Values

Next, we create another page called "**demo\_session2.php**". From this page, we will access the session information we set on the first page ("**demo\_session1.php**").

Notice that session variables are not passed individually to each new page, instead they are retrieved from the session we open at the beginning of each page (`session_start()`).

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>
<?php
// Echo session variables that were set on previous page
echo "Favorite color is " . $_SESSION["favcolor"] . "
";
echo "Favorite animal is " . $_SESSION["favanimal"] . " .";
?>
</body> </html>
```

### Destroy a PHP Session

To remove all global session variables and destroy the session, use `session_unset()` and `session_destroy()`:

#### Example

```
<?php
session_start();
?>
<html>
<body>
<?php
session_unset(); // remove all session variables
session_destroy(); // destroy the session
?> </body> </html>
```

### Difference between a session and a cookie

- The main **difference between a session** and a **cookie** is that **session** data is stored on the server, whereas **cookies** store data in the visitor's browser.
- **Sessions** are more secure than **cookies** as it is stored in server. **Cookie** can be turn off from browser.

	Sessions	Cookies
1	Sessions are <b>server-side</b> files that contain user information	Cookies are <b>client-side</b> files that contain user information
2	Session Max life time is 1440 Seconds(24 Minutes) as defined in php.ini file in <b>php.ini</b> on line 1604 you can find ; http://php.net/session.gc-maxlifetime session.gc_maxlifetime = 1440 You can edit this value if you need custom session life.	We have to set cookie max life time manually with php code with setcookie function. setcookie('email', 'test@example.com', time()+3600); /* expire in 1 hour */ Expire time : I hour after current time
3	In php <b>\$_SESSION</b> super global variable is used to manage session.	In php <b>\$_COOKIE</b> super global variable is used to manage cookie.
4	Before using <b>\$_SESSION</b> , you have to write session_start(); In that way session will start	You don't need to start Cookie as It is stored in your local machine.
5	You can store as much data as you like within in sessions.(default is 128MB.) memory_limit= 128M php.ini line 479 ;http://php.net/memory-limit	Official MAX Cookie size is 4KB
6	Session is dependent on COOKIE. Because when you start session with session_start() then SESSIONID named key will be set in COOKIE with Unique Identifier Value for your system.	
7	<b>session_destroy()</b> ; is used to " <b>Destroys all data registered to a session</b> ", and if you want to unset some key's of SESSION then use unset() function. unset(\$_SESSION["key1"], \$_SESSION["key2"])	There is no function named unsetcookie() time()-3600);//expire before 1 hour In that way you unset cookie(Set cookie in previous time)
8	Session ends when user closes his browser.	Cookie ends depends on the life time you set for it.
9	A <b>session</b> is a group of information on the server that is associated with the cookie information.	<b>Cookies</b> are used to identify sessions.

**Write a Program to create simple Login and Logout example using sessions.**

#### login.php

```
<html>
<head>
<title>Login Form</title>
</head>
<body>
<h2>Login Form</h2>
<form method="post" action="checklogin.php">
User Id: <input type="text" name="uid">

```

```
Password: <input type="password" name="pw">

<input type="submit" value="Login">
</form>
</body>
</html>
```

**checklogin.php**

```
<?php
$uid = $_POST['uid'];
$pw = $_POST['pw'];
if($uid == 'arun' and $pw == 'arun123')
{
 session_start();
 $_SESSION['sid']=session_id();
 header("location:securepage.php");
}
?>
```

**securepage.php**

```
<?php
 session_start();
 if($_SESSION['sid']==session_id())
 {
echo "Welcome to you
";
echo "Logout";
 }
 else
 {
 header("location:login.php");
 }
?>
```

**logout.php**

```
<?php
 echo "Logged out successfully";
 session_start();
 session_destroy();
 setcookie(PHPSESSID,session_id(),time()-1);
?>
```

**XML** - XML stands for **Extensible Mark-up Language**, developed by W3C in 1996. It is a text-based mark-up language derived from Standard Generalized Mark-up Language (SGML). XML 1.0 was officially adopted as a W3C recommendation in 1998. XML was designed to carry data, not to display data. XML is designed to be self-descriptive. XML is a subset of SGML that can define your own tags. A Meta Language and tags describe the content. XML Supports CSS, XSL, DOM. XML does not qualify to be a programming language as it does not performs any computation or algorithms. It is usually stored in a simple text file and is processed by special software that is capable of interpreting XML.

### The Difference between XML and HTML

1. HTML is about displaying information, where asXML is about carrying information. In other words, XML was created to structure, store, and transport information. HTML was designed to display the data.
2. Using XML, we can create own tags where as in HTML it is not possible instead it offers several built in tags.
3. XML is platform independent neutral and language independent.
4. XML tags and attribute names are case-sensitive where as in HTML it is not.
5. XML attribute values must be single or double quoted where as in HTML it is not compulsory.
6. XML elements must be properly nested.
7. All XML elements must have a closing tag.

### Well Formed XML Documents

**A "Well Formed" XML document must have the following correct XML syntax:**

- XML documents must have a root element
- XML elements must have a closing tag(start tag must have matching end tag).
- XML tags are case sensitive
- XML elements must be properly nested Ex:<one><two>Hello</two></one>
- XML attribute values must be quoted

XML with correct syntax is "Well Formed" XML. XML validated against a DTD is "Valid" XML.

### What is Markup?

XML is a markup language that defines set of rules for encoding documents in a format that is both human-readable and machine-readable.

### Example for XML Document

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?> <!--xml declaration-->
<note>
<to>MRCET</to>
<from>MRGI</from>
<heading>KALPANA</heading>
<body>Hello, world! </body>
</note>
```

- Xml document begins with XML declaration statement: <? xml version="1.0" encoding="ISO-8859-1"?> .
- The next line describes the **root element** of the document: <note>.



- This element is "the parent" of all other elements.
- The next 4 lines describe 4 **child elements** of the root: to, from, heading, and body. And finally the last line defines the end of the root element : **< /note>**.
- The XML declaration has no closing tag i.e. **</?xml>**
- The **default standalone value** is set to **no**. Setting it to **yes** tells the processor there are no external declarations (DTD) required for parsing the document. The file name extension used for xml program is .xml.

### Valid XML document

If an XML document is well-formed and has an associated Document Type Declaration (DTD), then it is said to be a valid XML document. We will study more about DTD in the chapter XML - DTDs.

### XML DTD

Document Type Definition purpose is to define the structure of an XML document. It defines the structure with a list of defined elements in the xml document. Using DTD we can specify the various elements types, attributes and their relationship with one another. Basically DTD is used to specify the set of rules for structuring data in any XML file.

### Why use a DTD?

XML provides an application independent way of sharing data. With a DTD, independent groups 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.

### DTD - XML building blocks

Various building blocks of XML are-

**1. Elements:** The basic entity is **element**. The elements are used for defining the tags. The elements typically consist of opening and closing tag. Mostly only one element is used to define a single tag.

**Syntax1:** `<!ELEMENT element-name (element-content)>`

**Syntax 2:** `<!ELEMENT element-name (#CDATA)>`

#CDATA means the element contains character data that is not supposed to be parsed by a parser. or

**Syntax 3:** `<!ELEMENT element-name (#PCDATA)>`

#PCDATA means that the element contains data that IS going to be parsed by a parser. or

**Syntax 4:** `<!ELEMENT element-name (ANY)>`

The keyword ANY declares an element with any content.

### Example:

`<!ELEMENT note (#PCDATA)>`

### Elements with children (sequences)

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

```
<!ELEMENT parent-name (child-element-name)> EX: <!ELEMENT student (id)>
 <!ELEMENT id (#PCDATA)> or
<!ELEMENT element-name (child-element-name, child-element-name,.....)>
```

**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 document will be:

```

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

```

## 2. Tags

Tags are used to markup elements. A starting tag like `<element_name>` mark up the beginning of an element, and an ending tag like `</element_name>` mark up the end of an element.

### Examples:

A body element: `<body>body text in between</body>`.

A message element: `<message>some message in between</message>`

**3. Attribute:** The attributes are generally used to specify the values of the element. These are specified within the double quotes. Ex: `<flag type="true">`

## 4. Entities

Entities as variables used to define common text. Entity references are references to entities. Most of you will know the HTML entity reference: "&nbsp;" that is used to insert an extra space in an HTML document. Entities are expanded when a document is parsed by an XML parser.

### The following entities are predefined in XML:

&lt; (<), &gt; (>), &amp; (&), &quot; (") and &apos; (').

**5. CDATA:** It stands for 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.

**6. PCDATA:** It stands for Parsed Character Data(i.e., text). Any parsed character data should not contain the markup characters. The markup characters are < or > or &. If we want to use these characters then make use of &lt; , &gt; or &amp;. 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**. Tags inside the text will be treated as markup and entities will be expanded.

```

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

```

Where PCDATA refers parsed character data. In the above xml document the elements to, from, heading, body carries some text, so that, these elements are declared to carry text in DTD file.

This definition file is stored with **.dtd** extension.

DTD identifier is an identifier for the document type definition, which may be the path to a file on the system or URL to a file on the internet. If the DTD is pointing to external path, it is called External Subset.

The square brackets [ ] enclose an optional list of entity declarations called Internal Subset.

**Types of DTD:**

1. Internal DTD
2. External DTD

**1. Internal DTD**

A DTD is referred to as an internal DTD if elements are declared within the XML files. To refer it as internal DTD, standalone attribute in XML declaration must be set to yes. This means, the declaration works independent of external source.

**Syntax:**

The syntax of internal DTD is as shown:

```
<!DOCTYPE root-element [element-declarations]>
```

Where root-element is the name of root element and element-declarations is where you declare the elements.

**Example:**

Following is a simple example of internal DTD:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE address [
 <!ELEMENT address (name,company,phone)>
 <!ELEMENT name (#PCDATA)>
 <!ELEMENT company (#PCDATA)>
 <!ELEMENT phone (#PCDATA)>
]>
<address>
 <name>Kalpana</name>
 <company>MRCET</company>
 <phone>(040) 123-4567</phone>
</address>
```

**Let us go through the above code:**

Start Declaration- Begin the XML declaration with following statement `<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>`

DTD- Immediately after the XML header, the document type declaration follows, commonly referred to as the DOCTYPE:

```
<!DOCTYPE address [
```

The DOCTYPE declaration has an exclamation mark (!) at the start of the element name. The DOCTYPE informs the parser that a DTD is associated with this XML document.

**DTD Body-** The DOCTYPE declaration is followed by body of the DTD, where you declare elements, attributes, entities, and notations:

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone_no (#PCDATA)>
```

Several elements are declared here that make up the vocabulary of the `<name>` document. `<!ELEMENT name (#PCDATA)>` defines the element name to be of type `"#PCDATA"`. Here `#PCDATA` means parse-able text data. End Declaration - Finally, the declaration section of the DTD is closed using a closing bracket and a closing angle bracket (`]>`). This effectively ends the definition, and thereafter, the XML document follows immediately.

**Rules**

- ✓ The document type declaration must appear at the start of the document (preceded only by the XML header) — it is not permitted anywhere else within the document.
- ✓ Similar to the DOCTYPE declaration, the element declarations must start with an exclamation mark.
- ✓ The Name in the document type declaration must match the element type of the root element.

**External DTD**

In external DTD elements are declared outside the XML file. They are accessed by specifying the system attributes which may be either the legal .dtd file or a valid URL. To refer it as external DTD, standalone attribute in the XML declaration must be set as no. This means, declaration includes information from the external source.

**Syntax** Following is the syntax for external DTD:

```
<!DOCTYPE root-element SYSTEM "file-name">
```

where file-name is the file with **.dtd** extension.

**Example** The following example shows external DTD usage:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE address SYSTEM "address.dtd">
<address>
 <name>Kalpana</name>
 <company>MRCET</company>
 <phone>(040) 123-4567</phone>
</address>
```

The content of the DTD file **address.dtd** are as shown:

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
```

**Types**

You can refer to an external DTD by using either system identifiers or public identifiers.

**SYSTEM IDENTIFIERS**

A system identifier enables you to specify the location of an external file containing DTD declarations. Syntax is as follows:

```
<!DOCTYPE name SYSTEM "address.dtd" [...]>
```

As you can see, it contains keyword SYSTEM and a URI reference pointing to the location of the document.

**PUBLIC IDENTIFIERS**

Public identifiers provide a mechanism to locate DTD resources and are written as below:

```
<!DOCTYPE name PUBLIC "-//Beginning XML/DTD Address Example//EN">
```

As you can see, it begins with keyword PUBLIC, followed by a specialized identifier. Public identifiers are used to identify an entry in a catalog. Public identifiers can follow any format; however, a commonly used format is called Formal Public Identifiers, or FPIs.

## XML Schemas

- XML Schema is commonly known as XML Schema Definition (XSD). It is used to describe and validate the structure and the content of XML data. XML schema defines the elements, attributes and data types. Schema element supports Namespaces. It is similar to a database schema that describes the data in a database. XSD extension is “.xsd”.
- This can be used as an alternative to XML DTD. The XML schema became the W3C recommendation in 2001.
- XML schema defines elements, attributes, element having child elements, order of child elements. It also defines fixed and default values of elements and attributes.
- XML schema also allows the developer to use **data types**.

**Syntax :** You need to declare a schema in your XML document as follows:

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

### Example : contact.xsd

The following example shows how to use schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="contact">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="name" type="xs:string" />
 <xs:element name="company" type="xs:string" />
 <xs:element name="phone" type="xs:int" />
 </xs:sequence>
 </xs:complexType>
</xs:element>
</xs:schema>
```

The basic idea behind XML Schemas is that they describe the legitimate format that an XML document can take.

### XML Document: myschema.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<contact xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="contact.xsd">
<name>KALPANA</name>
<company>04024056789</company>
<phone>9876543210</phone>
</contact>
```

### Limitations of DTD:

- There is no built-in data type in DTDs.
- No new data type can be created in DTDs.
- The use of cardinality (no. of occurrences) in DTDs is limited.
- Namespaces are not supported.
- DTDs provide very limited support for modularity and reuse.
- We cannot put any restrictions on text content.
- Defaults for elements cannot be specified.
- DTDs are written in a non-XML format and are difficult to validate.

**Strengths of Schema:**

- XML schemas provide much greater specificity than DTDs.
- They supports large number of built-in-data types.
- They are namespace-aware.
- They are extensible to future additions.
- They support the uniqueness.
- It is easier to define data facets (restrictions on data).

**SCHEMA STRUCTURE****The Schema Element**

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

**Element definitions**

As we saw in the chapter XML - Elements, elements are the building blocks of XML document. An element can be defined within an XSD as follows:

```
<xs:element name="x" type="y"/>
```

**Data types:**

These can be used to specify the type of data stored in an Element.

- String (xs:string)
- Date (xs:date or xs:time)
- Numeric (xs:integer or xs:decimal)
- Boolean (xs:boolean)

**EX: Sample.xsd**

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/XMLSchema">
 <xs:element name="sname" type="xs:string"/>
 /* <xs:element name="dob" type="xs:date"/>
 <xs:element name="dobtime" type="xs:time"/>
 <xs:element name="marks" type="xs:integer"/>
 <xs:element name="avg" type="xs:decimal"/>
 <xs:element name="flag" type="xs:boolean"/> */
</xs:schema>
```

**Sample.xml:**

```
<?xml version="1.0" encoding="UTF-8"?>
<sname xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="sample.xsd">
 Kalpana /*yyyy-mm-dd 23:14:34 600 92.5 true/false */
</sname>
```

**Definition Types**

You can define XML schema elements in following ways:

**Simple Type** - Simple type element is used only in the context of the text. Some of predefined simple types are: xs:integer, xs:boolean, xs:string, xs:date. For example:

```
<xs:element name="phone_number" type="xs:int" />
<phone>9876543210</phone>
```

### Default and Fixed Values for Simple Elements

In the following example the default value is "red":

```
<xs:element name="color" type="xs:string" default="red"/>
```

In the following example the fixed value is "red":

```
<xs:element name="color" type="xs:string" fixed="red"/>
```

**Complex Type** - A complex type is a container for other element definitions. This allows you to specify which child elements an element can contain and to provide some structure within your XML documents. For example:

```
<xs:element name="Address">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="name" type="xs:string" />
 <xs:element name="company" type="xs:string" />
 <xs:element name="phone" type="xs:int" />
 </xs:sequence>
 </xs:complexType>
</xs:element>
```

In the above example, Address element consists of child elements. This is a container for other `<xs:element>` definitions, that allows to build a simple hierarchy of elements in the XML document.

**Global Types** - With global type, you can define a single type in your document, which can be used by **all other references**. For example, suppose you want to generalize the person and company for different addresses of the company. In such case, you can define a general type as below:

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

Now let us use this type in our example as below:

```
<xs:element name="Address1">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="address" type="AddressType" />
 <xs:element name="phone1" type="xs:int" />
 </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="Address2">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="address" type="AddressType" />
 <xs:element name="phone2" type="xs:int" />
 </xs:sequence>
 </xs:complexType>
</xs:element>
```

Instead of having to define the name and the company twice (once for Address1 and once for Address2), we now have a single definition. This makes maintenance simpler, i.e., if you decide to add "Postcode" elements to the address, you need to add them at just one place.

### Attributes

Simple elements cannot have attributes. If an element has attributes, it is considered to be of a complex type. But the attribute itself is always declared as a simple type. Attributes in XSD provide extra information within an element. Attributes have name and type property as shown below:

```
<xs:attribute name="x" type="y"/>
```

**Ex:** <lastname lang="EN">Smith</lastname>

```
<xs:attribute name="lang" type="xs:string"/>
```

### Default and Fixed Values for Attributes

```
<xs:attribute name="lang" type="xs:string" default="EN"/>
```

```
<xs:attribute name="lang" type="xs:string" fixed="EN"/>
```

### Optional and Required Attributes

Attributes are optional by default. To specify that the attribute is required, use the "use" attribute:

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

### Restrictions on Content

When an XML element or attribute has a data type defined, it puts restrictions on the element's or attribute's content. If an XML element is of type "xs:date" and contains a string like "Hello World", the element will not validate.

### Restrictions on Values:

The value of **age** cannot be lower than 0 or greater than 120:

```
<xs:element name="age">
```

```
<xs:simpleType>
```

```
<xs:restriction base="xs:integer">
```

```
<xs:minInclusive value="0"/>
```

```
<xs:maxInclusive value="120"/>
```

```
</xs:restriction>
```

```
</xs:simpleType> </xs:element>
```

### Restrictions on a Set of Values

The example below defines an element called "car" with a restriction. The only acceptable values are: Audi, Golf, BMW:

```
<xs:element name="car">
```

```
<xs:simpleType>
```

```
<xs:restriction base="xs:string">
```

```
<xs:enumeration value="Audi"/>
```

```
<xs:enumeration value="Golf"/>
```

```
<xs:enumeration value="BMW"/>
```

```
</xs:restriction>
```

```
</xs:simpleType>
```

```
</xs:element>
```



**Restrictions on Length**

To limit the length of a value in an element, we would use the length, maxLength, and minLength constraints. **The value must be exactly eight characters:**

```
<xs:element name="password">
 <xs:simpleType>
 <xs:restriction base="xs:string">
 <xs:length value="8"/> [<xs:minLength value="5"/> <xs:maxLength value="8"/>]
 </xs:restriction>
 </xs:simpleType>
</xs:element>
```

**XSD Indicators**

We can control HOW elements are to be used in documents with indicators.

**Indicators:** There are seven indicators

**Order indicators:**

- All
- Choice
- Sequence

**Occurrence indicators:**

- maxOccurs
- minOccurs

**Group indicators:**

- Group name
- attributeGroup name

**→ Order Indicators**

Order indicators are used to define the order of the elements.

**All Indicator**

The <all> indicator specifies that the child elements can appear in any order, and that each child element must occur only once:

```
<xs:element name="person">
 <xs:complexType>
 <xs:all>
 <xs:element name="firstname" type="xs:string"/>
 <xs:element name="lastname" type="xs:string"/>
 </xs:all>
 </xs:complexType>
</xs:element>
```

**Note:** When using the <all> indicator you can set the <minOccurs> indicator to 0 or 1 and the <maxOccurs> indicator can only be set to 1 (the <minOccurs> and <maxOccurs> are described later).

**Choice Indicator**

The <choice> indicator specifies that either one child element or another can occur:

```
<xs:element name="person">
 <xs:complexType>
 <xs:choice>
 <xs:element name="employee" type="employee"/>
 <xs:element name="member" type="member"/>
 </xs:choice>
 </xs:complexType>
</xs:element>
```

## Sequence Indicator

The <sequence> indicator specifies that the child elements must appear in a specific order:

```
<xs:element name="person">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="firstname" type="xs:string"/>
 <xs:element name="lastname" type="xs:string"/>
 </xs:sequence>
 </xs:complexType>
</xs:element>
```

## → Occurrence Indicators

Occurrence indicators are used to define how often an element can occur.

**Note:** For all "Order" and "Group" indicators (any, all, choice, sequence, group name, and group reference) **the default value for maxOccurs and minOccurs is 1.**

### maxOccurs Indicator

The <maxOccurs> indicator specifies the maximum number of times an element can occur:

```
<xs:element name="person">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="full_name" type="xs:string"/>
 <xs:element name="child_name" type="xs:string" maxOccurs="10"/>
 </xs:sequence>
 </xs:complexType>
</xs:element>
```

### minOccurs Indicator

The <minOccurs> indicator specifies the minimum number of times an element can occur:

```
<xs:element name="person">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="full_name" type="xs:string"/>
 <xs:element name="child_name" type="xs:string" maxOccurs="10" minOccurs="0"/>
 </xs:sequence>
 </xs:complexType>
</xs:element>
```

**Tip:** To allow an element to appear an unlimited number of times, use the

**maxOccurs="unbounded"** statement:

**EX:** An XML file called "Myfamily.xml":

```
<?xml version="1.0" encoding="UTF-8"?>
<persons xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="family.xsd">
 <person>
 <full_name>KALPANA</full_name>
 <child_name>mrcet</child_name>
 </person>
 <person>
 <full_name>Tove Refsnes</full_name>
 <child_name>Hege</child_name>
 <child_name>Stale</child_name>
```

```

 <child_name>Jim</child_name>
 <child_name>Borge</child_name>
 </person>
 <person>
 <full_name>Stale Refsnes</full_name>
 </person>
</persons>

```

The XML file above contains a root element named "persons". Inside this root element we have defined three "person" elements. Each "person" element must contain a "full\_name" element and it can contain up to five "child\_name" elements.

Here is the schema file "**family.xsd**":

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema
 elementFormDefault="qualified">
 <xs:element name="persons">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="person" maxOccurs="unbounded">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="full_name" type="xs:string"/>
 <xs:element name="child_name" type="xs:string" minOccurs="0" maxOccurs="5"/>
 </xs:sequence>
 </xs:complexType>
 </xs:element>
 </xs:sequence>
 </xs:complexType>
 </xs:element>
</xs:schema>

```

→ **Group Indicators:** Group indicators are used to define related sets of elements.

### Element Groups

Element groups are defined with the group declaration, like this:

```

<xs:group name="groupname">
...
</xs:group>

```

You must define an all, choice, or sequence element inside the group declaration. The following example defines a group named "persongroup", that defines a group of elements that must occur in an exact sequence:

```

<xs:group name="persongroup">
 <xs:sequence>
 <xs:element name="firstname" type="xs:string"/>
 <xs:element name="lastname" type="xs:string"/>
 <xs:element name="birthday" type="xs:date"/>
 </xs:sequence>
</xs:group>

```

After you have defined a group, **you can reference it in another definition**, like this:

```
<xs:element name="person" type="personinfo"/>
<xs:complexType name="personinfo">
 <xs:sequence>
 <xs:group ref="persongroup"/>
 <xs:element name="country" type="xs:string"/>
 </xs:sequence>
</xs:complexType>
```

### Attribute Groups

Attribute groups are defined with the attributeGroup declaration, like this:

```
<xs:attributeGroup name="groupname">
...
</xs:attributeGroup>
```

The following example defines an attribute group named "personattrgroup":

```
<xs:attributeGroup name="personattrgroup">
 <xs:attribute name="firstname" type="xs:string"/>
 <xs:attribute name="lastname" type="xs:string"/>
 <xs:attribute name="birthday" type="xs:date"/>
</xs:attributeGroup>
```

After you have defined an attribute group, you can reference it in another definition, like this:

```
<xs:element name="person">
 <xs:complexType>
 <xs:attributeGroup ref="personattrgroup"/> </xs:complexType> </xs:element>
```

### Example Program: "shiporder.xml"

```
<?xml version="1.0" encoding="UTF-8"?>
<shiporder orderid="889923"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="shiporder.xsd">
 <orderperson>John Smith</orderperson>
 <shipto>
 <name>Ola Nordmann</name>
 <address>Langgt 23</address>
 <city>4000 Stavanger</city>
 <country>Norway</country>
 </shipto>
 <item>
 <title>Empire Burlesque</title>
 <note>Special Edition</note>
 <quantity>1</quantity>
 <price>10.90</price>
 </item>
 <item>
 <title>Hide your heart</title> <quantity>1</quantity>
 <price>9.90</price> </item>
</shiporder>
```

**Create an XML Schema "shiporder.xsd":**

```

<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="shiporder">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="orderperson" type="xs:string"/>
 <xs:element name="shipto">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="name" type="xs:string"/>
 <xs:element name="address" type="xs:string"/>
 <xs:element name="city" type="xs:string"/>
 <xs:element name="country" type="xs:string"/>
 </xs:sequence>
 </xs:complexType>
 </xs:element>
 <xs:element name="item" maxOccurs="unbounded">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="title" type="xs:string"/>
 <xs:element name="note" type="xs:string" minOccurs="0"/>
 <xs:element name="quantity" type="xs:positiveInteger"/>
 <xs:element name="price" type="xs:decimal"/>
 </xs:sequence>
 </xs:complexType>
 </xs:element>
 </xs:sequence>
 <xs:attribute name="orderid" type="xs:string" use="required"/>
 </xs:complexType>
</xs:element>
</xs:schema>

```

**XML DTD vs XML Schema**

The schema has more advantages over DTD. A DTD can have two types of data in it, namely the CDATA and the PCDATA. The CDATA is not parsed by the parser whereas the PCDATA is parsed. In a schema you can have primitive data types and custom data types like you have used in programming.

**Schema vs. DTD**

- ☐ XML Schemas are extensible to future additions
- ☐ XML Schemas are richer and more powerful than DTDs
- ☐ XMLSchemas are written in XML
- ☐ XMLSchemas support data types
- ☐ XMLSchemas support namespaces

**XML Parsers**

An XML parser converts an XML document into an XML DOM object - which can then be manipulated with a JavaScript.

**Two types of XML parsers:****➤ Validating Parser**

- It requires document type declaration
- It generates error if document does not
  - Conform with DTD and
  - Meet XML validity constraints

**➤ Non-validating Parser**

- It checks well-formedness for xml document
- It can ignore external DTD

**What is XML Parser?**

XML Parser provides way how to access or modify 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.

**Types of parsers:**

- **Dom Parser** - Parses the document by loading the complete contents of the document and creating its complete hierarchical tree in memory.
- **SAX Parser** - Parses the document on event based triggers. Does not load the complete document into the memory.
- **JDOM Parser** - Parses the document in similar fashion to DOM parser but in more easier way.
- **StAX Parser** - Parses the document in similar fashion to SAX parser but in more efficient way.
- **XPath Parser** - Parses the XML based on expression and is used extensively in conjunction with XSLT.
- **DOM4J Parser** - A java library to parse XML, XPath and XSLT using Java Collections Framework, provides support for DOM, SAX and JAXP.

**DOM-Document Object Model**

The Document Object Model protocol converts an XML document into a collection of objects in your program. XML documents have a hierarchy of informational units called nodes; this hierarchy allows a developer to navigate through the tree looking for specific information. Because it is based on a hierarchy of information, the DOM is said to be tree based. DOM is a way of describing those nodes and the relationships between them.

You can then manipulate the object model in any way that makes sense. This mechanism is also known as the "random access" protocol, because you can visit any part of the data at any time. You can then modify the data, remove it, or insert new data.

The XML DOM, on the other hand, also provides an API that allows a developer to add, edit, move, or remove nodes in the tree at any point in order to create an application. A DOM parser creates a tree structure in memory from the input document and then waits for requests from client. A DOM parser always serves the client application with the **entire document no matter how much is actually needed** by the client. With DOM parser, method calls in client application have to be explicit and forms a kind of chained method calls.

Document Object Model is for defining the standard for accessing and manipulating XML documents. **XML DOM** is used for

- Loading the xml document
- Accessing the xml document
- Deleting the elements of xml document
- Changing the elements of xml document

According to the DOM, everything in an XML document is a node. It considers

- The entire document is a document node
- Every XML element is an element node
- The text in the XML elements are text nodes
- Every attribute is an attribute node
- Comments are comment nodes

**The W3C DOM specification is divided into three major parts:**

**DOM Core-** This portion defines the basic set of interfaces and objects for any structured documents.

**XML DOM-** This part specifies the standard set of objects and interfaces for XML documents only.

**HTML DOM-** This part specifies the objects and interfaces for HTML documents only.

#### DOM Levels

- Level 1 Core: W3C Recommendation, October 1998
  - ✓ It has feature for primitive navigation and manipulation of XML trees
  - ✓ other Level 1 features are: All HTML features
- Level 2 Core: W3C Recommendation, November 2000
  - ✓ It adds Namespace support and minor new features
  - ✓ other Level 2 features are: Events, Views, Style, Traversal and Range
- Level 3 Core: W3C Working Draft, April 2002
  - ✓ It supports: Schemas, XPath, XSL, XSLT

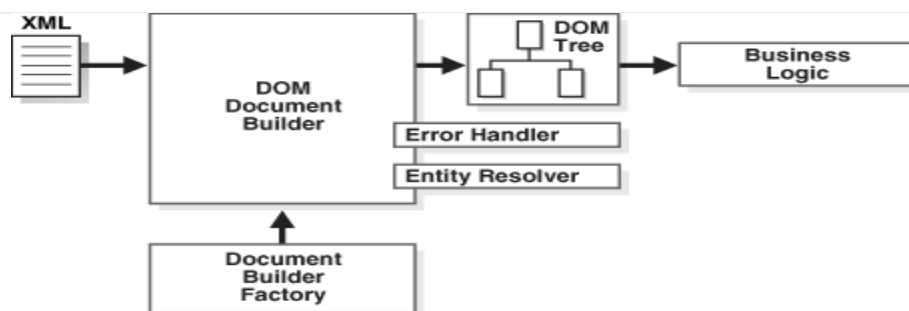
We can access and parse the XML document in two ways:

- **Parsing using DOM (tree based)**
- **Parsing using SAX (Event based)**

Parsing the XML doc. using DOM methods and properties are called as **tree based approach** whereas using SAX (Simple Api for Xml) methods and properties are called as **event based approach**.

#### Steps to Using DOM Parser

Let's note down some broad steps involved in using a DOM parser for parsing any XML file in java.



**DOM based XML Parsing:(tree based)**

JAXP is a tool, stands for Java Api for Xml Processing, used for accessing and manipulating xml document in a tree based manner.

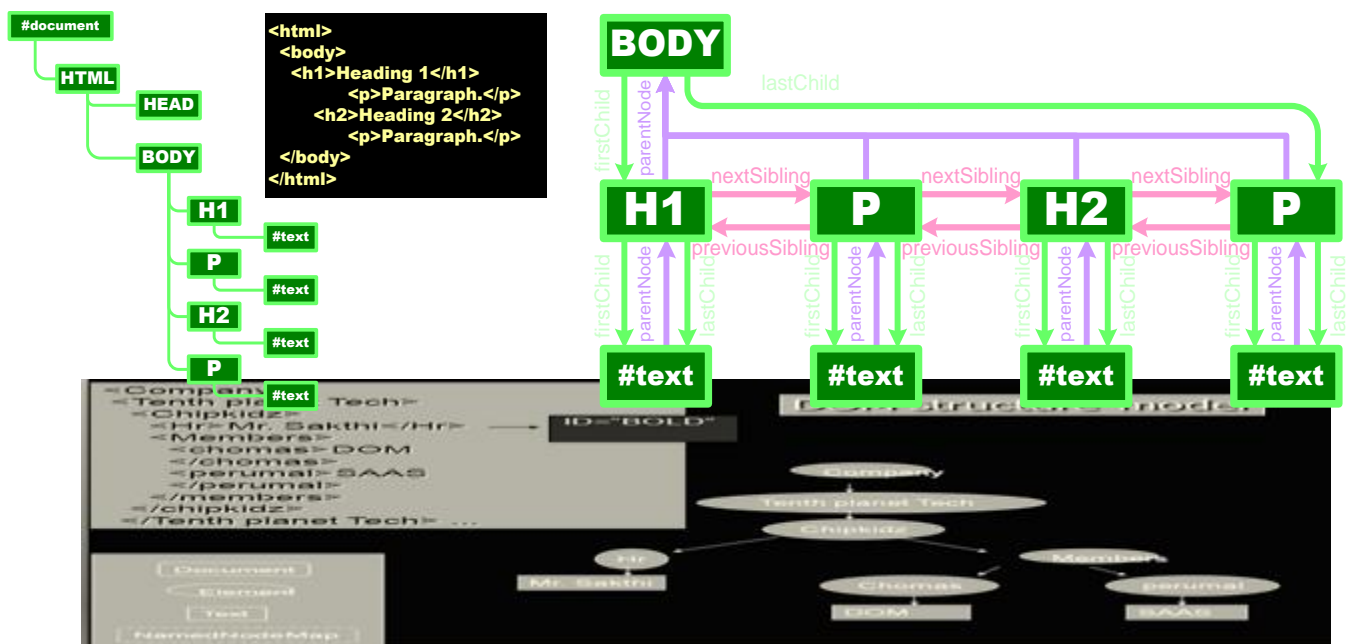
The following DOM javaClasses are necessary to process the XML document:

- DocumentBuilderFactory class creates the instance of DocumentBuilder.
- DocumentBuilder produces a Document (a DOM) that conforms to the DOM specification.

The following methods and properties are necessary to process the XML document:

Property	Meaning
nodeName	Finding the name of the node
nodeValue	Obtaining value of the node
parentNode	To get parnet node
childNodes	Obtain child nodes
Attributes	For getting the attributes values

Method	Meaning
getElementByTagName(name)	To access the element by specifying its name
appendChild(node)	To insert a child node
removeChild(node)	To remove existing child node

**DOM Document Object**

- ✓ There are 12 types of nodes in a DOM Document object

1. Document node	7. EntityReference node
2. Element node	8. Entity node
3. Text node	9. Comment node
4. Attribute node	10. DocumentType node
5. Processing instruction node	11. DocumentFragment node
6. CDATA Section node	12. Notation node



**Examples for Document method**

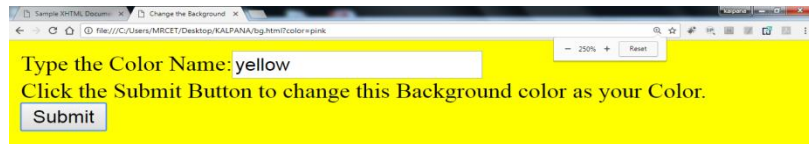
```

<html>
 <head>
 <title>Change the Background</title>
 </head>
 <body>
 <script language = "JavaScript">
 function background()
 {
 var color = document.bg.color.value;
 document.body.style.backgroundColor=color;
 }
 </script>
 <form name="bg">
 Type the Color Name:<input type="text" name="color" size="20">

 Click the Submit Button to change this Background color as your Color.

 <input type="button" value="Submit" onClick='background()'/>
 </form>
 </body>
</html>

```

**DOM NODE Methods**

Method Name	Description
<b>appendChild</b>	Appends a child node.
<b>cloneNode</b>	Duplicates the node.
<b>getAttribute</b>	Returns the node's attributes.
<b>getChildNodes</b>	Returns the node's child nodes.
<b>getNodeName</b>	Returns the node's name.
<b>getNodeType</b>	Returns the node's type (e.g., element, attribute, text, etc.).
<b>getNodeValue</b>	Returns the node's value.
<b>getParentNode</b>	Returns the node's parent.
<b>hasChildNodes</b>	Returns <b>true</b> if the node has child nodes.
<b>removeChild</b>	Removes a child node from the node.
<b>replaceChild</b>	Replaces a child node with another node.
<b>setNodeValue</b>	Sets the node's value.
<b>insertBefore</b>	Appends a child node in front of a child node.

**DOM Advantages & Disadvantages****ADVANTAGES**

- Robust API for the DOM tree
- Relatively simple to modify the data structure and extract data
- It is good when random access to widely separated parts of a document is required
- It supports both read and write operations

**Disadvantages**

- Stores the entire document in memory, It is memory inefficient
- As Dom was written for any language, method naming conventions don't follow standard java programming conventions

**DOM or SAX****DOM**

- Suitable for small documents
- Easily modify document
- Memory intensive; Load the complete XML document

**SAX**

- Suitable for large documents; saves significant amounts of memory
- Only traverse document once, start to end
- Event driven
- Limited standard functions.

**Loading an XML file:one.html**

```
<html> <body>
<script type="text/javascript">
try
{
xmlDocument=new ActiveXObject("Microsoft.XMLDOM");
}
catch(e)
{
try {
xmlDocument=document.implementation.createDocument("", "", null);
}
catch(e) {alert(e.message)}
}
try
{
xmlDocument.async=false;
xmlDocument.load("faculty.xml");
document.write("XML document student is loaded");
}
catch(e) {alert(e.message)}
</script>
</body> </html>
```

**faculty.xml:**

```
<?xml version="1.0"?>
< faculty >
 <eno>30</eno>
 <personal_inf>
 <name>Kalpana</name>
 <address>Hyd</address>
 <phone>9959967192</phone>
 </personal_inf>
 <dept>CSE</dept>
 <col>MRCET</col>
 <group>MRGI</group>
</faculty>
```

**OUTPUT:** XML document student is loaded

**ActiveXObject:** It creates empty xml document object.

**Use separate function for Loading an XML document: two.html**

```
<html> <head>
<script type="text/javascript">
Function My_function(doc_file)
{
try
{
xmlDocument=new ActiveXObject("Microsoft.XMLDOM");
}
catch(e)
{
try
{
xmlDocument=document.implementation.createDocument("", "", null);
}
catch(e){alert(e.message)}
}
try
{
xmlDocument.async=false;
xmlDocument.load("faculty.xml");
return(xmlDocument);
}
catch(e){alert(e.message)}
return(null);
}
</script>
</head>
<body>
<script type="text/javascript">
xmlDoc=My_function("faculty.xml");
document.write("XML document student is loaded");
</script>
</body> </html>
```

**OUTPUT:** XML document student is loaded

**Use of properties and methods: three.html**

```
<html> <head>
<script type="text/javascript" src="my_function_file.js"></script>
</head> <body>
<script type="text/javascript">
xmlDocument=My_function("faculty.xml");
document.write("XML document faculty is loaded and content of this file is:");
document.write("
");
document.write("ENO:" +
xmlDocument.getElementsByTagName("eno")[0].childNodes[0].nodeValue);
document.write("
");
document.write("Name:" +
xmlDocument.getElementsByTagName("name")[0].childNodes[0].nodeValue);
```

```
document.write("
");
document.write("ADDRESS:"'+
xmlDocument.getElementsByTagName("address")[0].childNodes[0].nodeValue);
document.write("
");
document.write("PHONE:"'+
xmlDocument.getElementsByTagName("phone")[0].childNodes[0].nodeValue);
document.write("
");
document.write("DEPARTMENT:"'+
xmlDocument.getElementsByTagName("dept")[0].childNodes[0].nodeValue);
document.write("
");
document.write("COLLEGE:"'+
xmlDocument.getElementsByTagName("col")[0].childNodes[0].nodeValue);
document.write("
");
document.write("GROUP:"'+
xmlDocument.getElementsByTagName("group")[0].childNodes[0].nodeValue);
</script>
</body>
</html>
```

**OUTPUT:**

XML document faculty is loaded and content of this file is

```
ENO: 30
NAME: Kalpana
ADDRESS: Hyd
PHONE: 9959967192
DEPARTMENT: CSE
COLLEGE: MRCET
GROUP: MRGI
```

**We can access any XML element using the index value: four.html**

```
<html> <head>
<script type="text/javascript" src="my_function_file.js"></script>
</head> <body>
<script type="text/javascript">
xmlDoc=My_function("faculty1.xml");
value=xmlDoc.getElementsByTagName("name");
document.write("value[0].childNodes[0].nodeValue");
</script></body></html>
```

**OUTPUT:** Kalpana

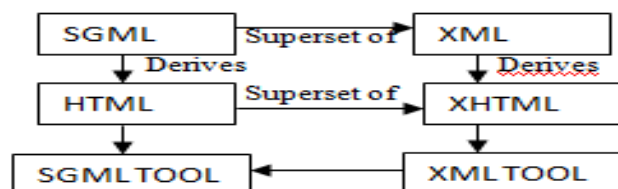
**XHTML: eXtensible Hypertext Markup Language**

**Hypertext** is simply a piece of text that works as a link. **Markup language** is a language of writing layout information within documents. The XHTML recommended by **W3C**. Basically an XHTML document is a plain text file and it is very much similar to HTML. It contains rich text, means text with tags. The extension to this program should be either **html** or **htm**. These programs can be opened in some web browsers and the corresponding web page can be viewed.

## HTML Vs XHTML

HTML	XHTML
1. The HTML tags are case insensitive. EX: <BoDy>-----</body>	1. The XHTML tags are case sensitive. EX: <body>-----</body>
2. We can omit the closing tags sometimes.	2. For every tag there must be a closing tag. EX: <h1>-----</h1> or <h1-----/>
3. The attribute values not always necessary to quote.	3. The attribute values are must be quoted.
4. In HTML there are some implicit attribute values.	4. In XHTML the attribute values must be specified explicitly.
5. In HTML even if we do not follow the nesting rules strictly it does not cause much difference.	5. In XHTML the nesting rules must be strictly followed. These nesting rules are- - A form element cannot contain another form element. -an anchor element does not contain another form element -List element cannot be nested in the list element -If there are two nested elements then the inner element must be enclosed first before closing the outer element -Text element cannot be directly nested in form element

The relationship between SGML, XML, HTML and XHTML is as given below



**Standard structure:** DOCTYPE, html, head and body

The doctype is specified by the DTD. The XHTML syntax rules are specified by the file xhtml11.dtd file. There are 3 types of DTDs.

1. **XHTML 1.0 Strict:** clean markup code
2. **XHTML 1.0 Transitional:** Use some html features in the existing XHTML document.
3. **XHTML 1.0 Frameset:** Use of Frames in an XHTML document.

**EX:**

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml11.dtd">

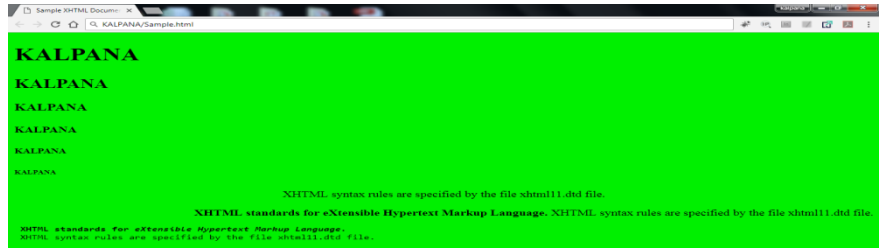
<html xmlns="http://www.w3c.org/1999/xhtml">
<head>
<title>Sample XHTML Document</title>
</head>
<body bgcolor="#FF0000">
<basefont face="arial" size="14" color="white">

```

```

<h1>MRCET</h1>
<h2>MRCET</h2>
<h3>MRCET</h3>
<h4> KALPANA </h4>
<h5> KALPANA </h5>
<h6>KALPANA</h6>
<p><center> XHTML syntax rules are specified by the file xhtml11.dtd file. </center></p>
<div align="right"> XHTML standards for eXtensible Hypertext Markup Language.
XHTML syntax rules are specified by the file xhtml11.dtd file.</div>
<pre> XHTML standards for <i>eXtensible Hypertext Markup Language.</i>
XHTML syntax rules are specified by the file xhtml11.dtd file.</pre>
</basefont>
</body>
</html>

```



## DOM in JAVA

### DOM interfaces

The DOM defines several Java interfaces. Here are the most common interfaces:

- **Node** - The base datatype of the DOM.
- **Element** - The vast majority of the objects you'll deal with are Elements.
- **Attr** Represents an attribute of an element.
- **Text** The actual content of an Element or Attr.
- **Document** Represents the entire XML document. A Document object is often referred to as a DOM tree.

### Common DOM methods

When you are working with the DOM, there are several methods you'll use often:

- **Document.getDocumentElement()** - Returns the root element of the document.
- **Node.getFirstChild()** - Returns the first child of a given Node.
- **Node.getLastChild()** - Returns the last child of a given Node.
- **Node.getNextSibling()** - These methods return the next sibling of a given Node.
- **Node.getPreviousSibling()** - These methods return the previous sibling of a given Node.
- **Node.getAttribute(attrName)** - For a given Node, returns the attribute with the requested name.

### Steps to Using DOM

Following are the steps used while parsing a document using DOM Parser.

- Import XML-related packages.
- Create a DocumentBuilder
- Create a Document from a file or stream
- Extract the root element
- Examine attributes
- Examine sub-elements

**DOM**

```
import java.io.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.*;
public class parsing_DOMDemo
{
 public static void main(String[] args)
 {
 try
 {
 System.out.println("enter the name of XML document");
 BufferedReader input=new BufferedReader(new InputStreamReader(System.in));
 String file_name=input.readLine();
 File fp=new File(file_name);
 if(fp.exists())
 {
 try
 {
 DocumentBuilderFactory Factory_obj= DocumentBuilderFactory.newInstance();
 DocumentBuilder builder=Factory_obj.newDocumentBuilder();
 InputSource ip_src=new InputSource(file_name);
 Document doc=builder.parse(ip_src);
 System.out.println("file_name+"is well-formed.");
 }
 catch (Exception e)
 {
 System.out.println(file_name+"is not well-formed.");
 System.exit(1);
 }
 }
 else
 {
 System.out.println("file not found:"+file_name);
 }
 }
 catch(IOException ex)
 {
 ex.printStackTrace();
 }
 }
}
```

**SAX:**

**SAX (the Simple API for XML)** is an event-based parser for xml documents. Unlike a DOM parser, a SAX parser creates no parse tree. SAX is a streaming interface for XML, which means that applications using SAX receive event notifications about the XML document being processed an element, and attribute, at a time in sequential order starting at the top of the document, and ending with the closing of the ROOT element.

- Reads an XML document from top to bottom, recognizing the tokens that make up a well-formed XML document

- Tokens are processed in the same order that they appear in the document
- Reports the application program the nature of tokens that the parser has encountered as they occur
- The application program provides an "event" handler that must be registered with the parser
- As the tokens are identified, callback methods in the handler are invoked with the relevant information

**When to use?**

You should use a SAX parser when:

- You can process the XML document in a linear fashion from the top down
- The document is not deeply nested
- You are processing a very large XML document whose DOM tree would consume too much memory. Typical DOM implementations use ten bytes of memory to represent one byte of XML
- The problem to be solved involves only part of the XML document
- Data is available as soon as it is seen by the parser, so SAX works well for an XML document that arrives over a stream

**Disadvantages of SAX**

- We have no random access to an XML document since it is processed in a forward-only manner
- If you need to keep track of data the parser has seen or change the order of items, you must write the code and store the data on your own
- The data is broken into pieces and clients never have all the information as a whole unless they create their own data structure

**The kinds of events are:**

- The start of the document is encountered
- The end of the document is encountered
- The start tag of an element is encountered
- The end tag of an element is encountered
- Character data is encountered
- A processing instruction is encountered

Scanning the XML file from start to end, each event invokes a corresponding callback method that the programmer writes.

**SAX packages**

**javax.xml.parsers:** Describing the main classes needed for parsing

**org.xml.sax:** Describing few interfaces for parsing

**SAX classes**

- **SAXParser** Defines the API that wraps an XMLReader implementation class
- **SAXParserFactory** Defines a factory API that enables applications to configure and obtain a SAX based parser to parse XML documents
- **ContentHandler** Receive notification of the logical content of a document.



- **DTDHandler** Receive notification of basic DTD-related events.
- **EntityResolver** Basic interface for resolving entities.
- **ErrorHandler** Basic interface for SAX error handlers.
- **DefaultHandler** Default base class for SAX event handlers.

### SAX parser methods

**StartDocument() and endDocument()** – methods called at the start and end of an XML document.

**StartElement() and endElement()** – methods called at the start and end of a document element.

**Characters()** – method called with the text contents in between the start and end tags of an XML document element.

### ContentHandler Interface

This interface specifies the callback methods that the SAX parser uses to notify an application program of the components of the XML document that it has seen.

- **void startDocument()** - Called at the beginning of a document.
- **void endDocument()** - Called at the end of a document.
- **void startElement(String uri, String localName, String qName, Attributes atts)** - Called at the beginning of an element.
- **void endElement(String uri, String localName, String qName)** - Called at the end of an element.
- **void characters(char[] ch, int start, int length)** - Called when character data is encountered.
- **void ignorableWhitespace(char[] ch, int start, int length)** - Called when a DTD is present and ignorable whitespace is encountered.
- **void processingInstruction(String target, String data)** - Called when a processing instruction is recognized.
- **void setDocumentLocator(Locator locator)** - Provides a Locator that can be used to identify positions in the document.
- **void skippedEntity(String name)** - Called when an unresolved entity is encountered.
- **void startPrefixMapping(String prefix, String uri)** - Called when a new namespace mapping is defined.
- **void endPrefixMapping(String prefix)** - Called when a namespace definition ends its scope.

### Attributes Interface

This interface specifies methods for processing the attributes connected to an element.

- **int getLength()** - Returns number of attributes, etc.

### SAX simple API for XML

```
import java.io.*;
import org.xml.sax;
import org.xml.sax.helpers;
public class parsing_SAXDemo
{
public static void main(String[] args) throws IOException
{
```

```

try {
System.out.println("enter the name of XML document");
BufferedReader input=new BufferedReader(new InputStreamReader(System.in));
String file_name=input.readLine();
File fp=new File(file_name);
if(fp.exists())
{
try
{
XMLReader reader=XMLReaderFactory.createXMLReader();
reader.parse(file_name);
System.out.println("file_name+"is well- formed.");
}
catch (Exception e)
{
System.out.println(file_name+"is not well- formed.");
System.exit(1);
}
}
else
{
System.out.println("file not found:"+file_name);
}
}
catch(IOException ex){ex.printStackTrace();}
} }

```

### Differences between DOM and SAX

DOM	SAX
Stores the entire XML document into memory before processing	Parses node by node
Occupies more memory	Doesn't store the XML in memory
We can insert or delete nodes	We can't insert or delete a node
DOM is a tree model parser	SAX is an event based parser
Document Object Model (DOM) API	SAX is a Simple API for XML
Preserves comments	Doesn't preserve comments
DOM is slower than SAX, heavy weight.	SAX generally runs a little faster than DOM, light weight.
Traverse in any direction.	Top to bottom traversing is done in this approach
Random Access	Serial Access
<b>Packages required to import</b> import javax.xml.parsers.*; import javax.xml.parsers.DocumentBuilder; import javax.xml.parsers.DocumentBuilderFactory;	<b>Packages required to import</b> import java.xml.parsers.*; import org.xml.sax.*; import org.xml.sax.helpers;

## UNIT III

### Web Servers and Servlets

#### Web Servers:

Web servers are computers that deliver (*serves up*) Web pages. Every Web server has an IP address and possibly a domain name. For example, if you enter the URL *http://www.mrcet.com/index.html* in your browser, this sends a request to the Web server whose domain name is *mrcet.com*. The server then fetches the page named *index.html* and sends it to your browser.

Any computer can be turned into a Web server by installing server software and connecting the machine to the Internet. There are many Web server software applications, including public domain software and commercial packages.

#### Install TOMCAT web server and APACHE.

While installation, we assign port number 8080 to APACHE. Make sure that these ports are available i.e., no other process is using this port.

#### DESCRIPTION:

##### *Set the JAVA\_HOME Variable*

You must set the JAVA\_HOME environment variable to tell Tomcat where to find Java. Failing to properly set this variable prevents Tomcat from handling JSP pages. This variable should list the base JDK installation directory, not the bin subdirectory. On Windows XP, you could also go to the Start menu, select Control Panel, choose System, click on the Advanced tab, press the Environment Variables button at the bottom, and enter the JAVA\_HOME variable and value directly as:

**Name:** JAVA\_HOME

**Value:** C:\jdk

##### *Set the CLASSPATH*

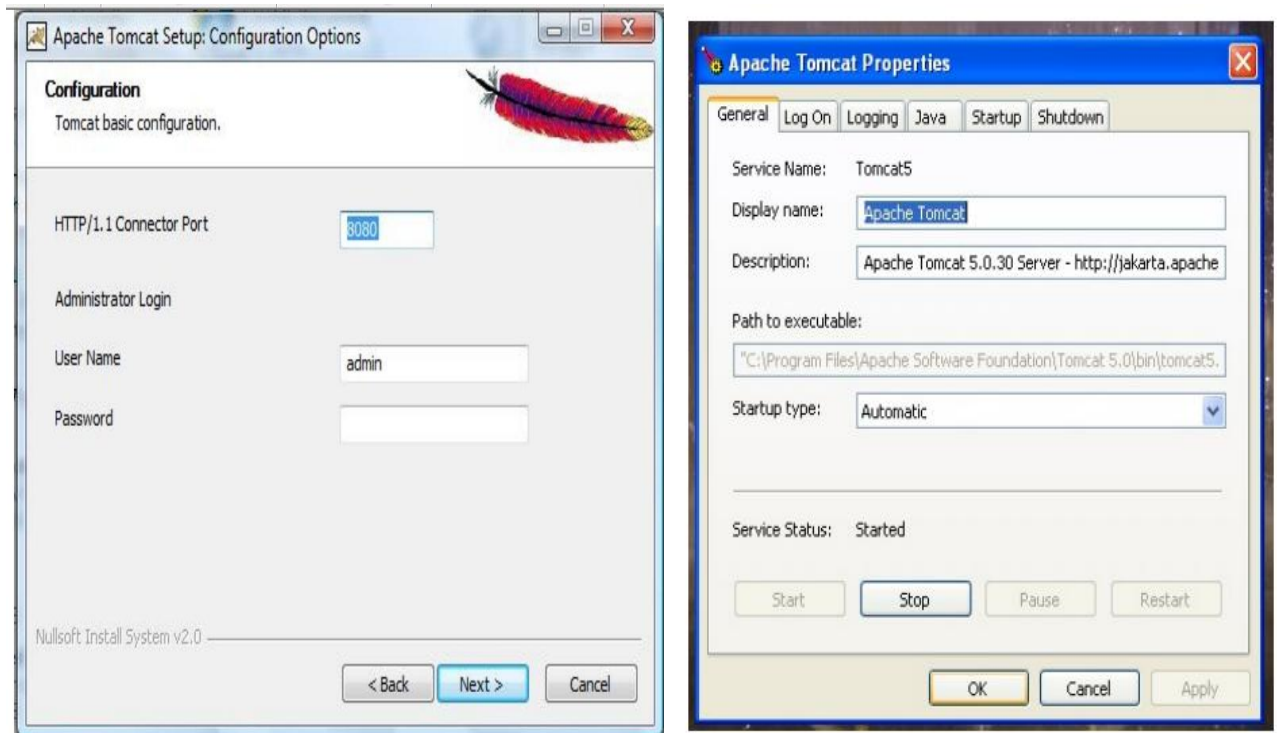
Since servlets and JSP are not part of the Java 2 platform, standard edition, you have to identify the servlet classes to the compiler. The server already knows about the servlet classes, but the compiler (i.e., javac) you use for development probably doesn't. So, if you don't set your CLASSPATH, attempts to compile servlets, tag libraries, or other classes that use the servlet and JSP APIs will fail with error messages about unknown classes.

**Name:** JAVA\_HOME

**Value:** install\_dir/common/lib/servlet-api.jar

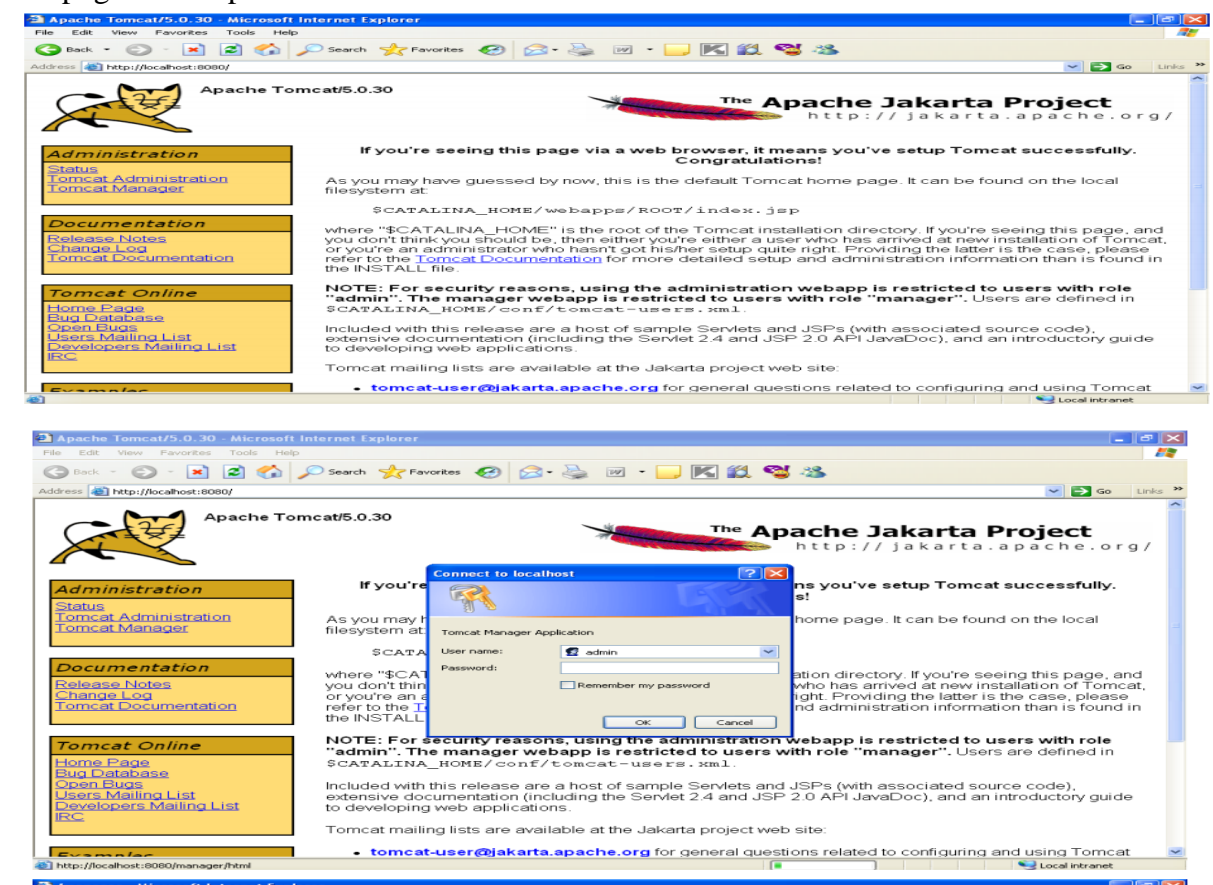
##### *Turn on Servlet Reloading*

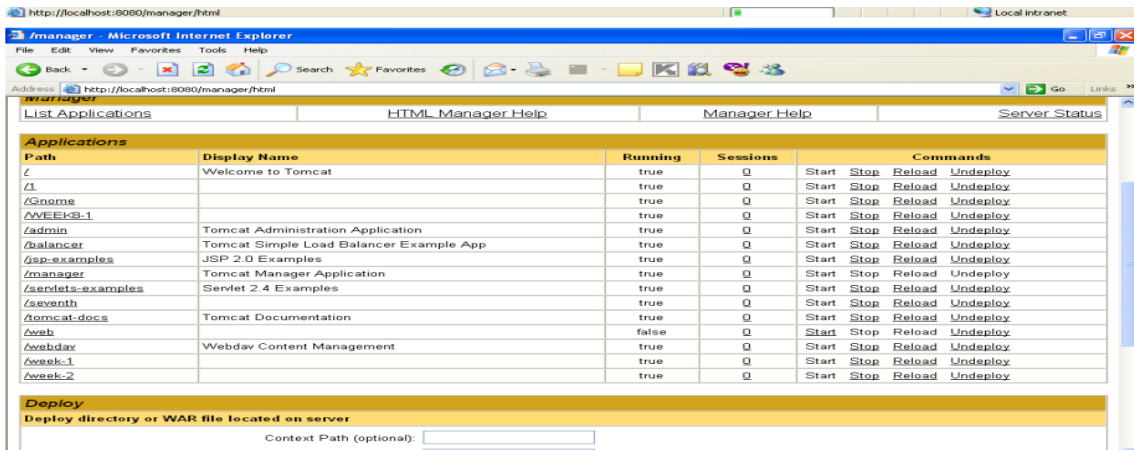
The next step is to tell Tomcat to check the modification dates of the class files of requested servlets and reload ones that have changed since they were loaded into the server's memory. This slightly degrades performance in deployment situations, so is turned off by default. However, if you fail to turn it on for your development server, you'll have to restart the server every time you recompile a servlet that has already been loaded into the server's memory.



**RESULT:** Thus TOMCAT web server was installed successfully.

Access the developed static web pages for books web site, using these servers by putting the web pages developed in week-1 and week-2 in the document root.





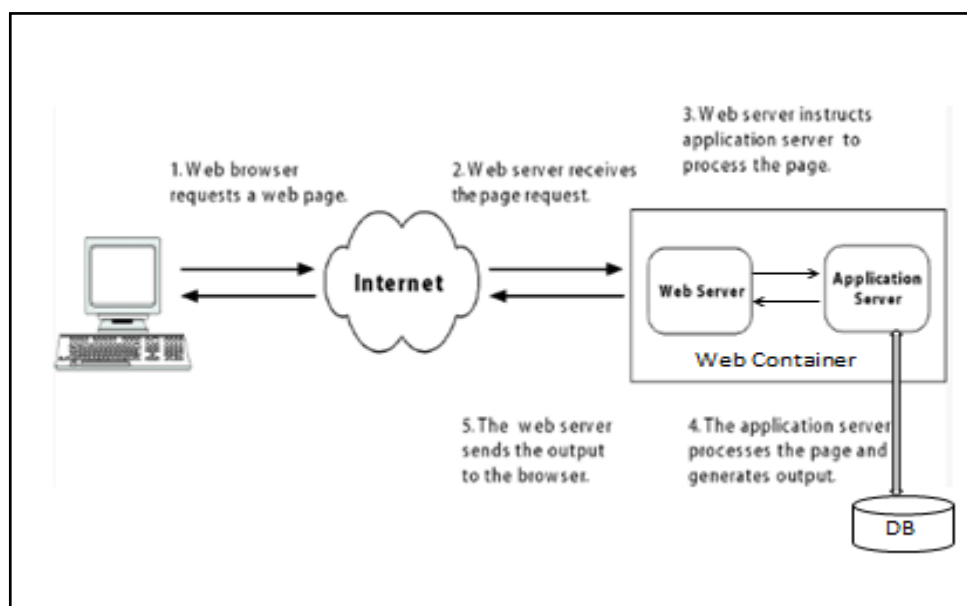
**RESULT:** These pages are accessed using the TOMCAT web server successfully.

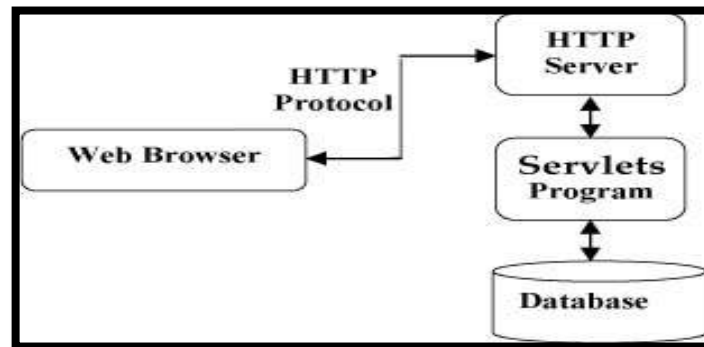
## INTRODUCTION TO SERVLETS

### Servlets:

- Servlets are server side programs that run on a Web or Application server and act as a middle layer between a requests coming from a Web browser and databases or applications on the server.
- Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.
- Servlets don't fork new process for each request, instead a new thread is created.
- Servlets are loaded and ready for each request.
- The same servlet can handle many requests simultaneously.

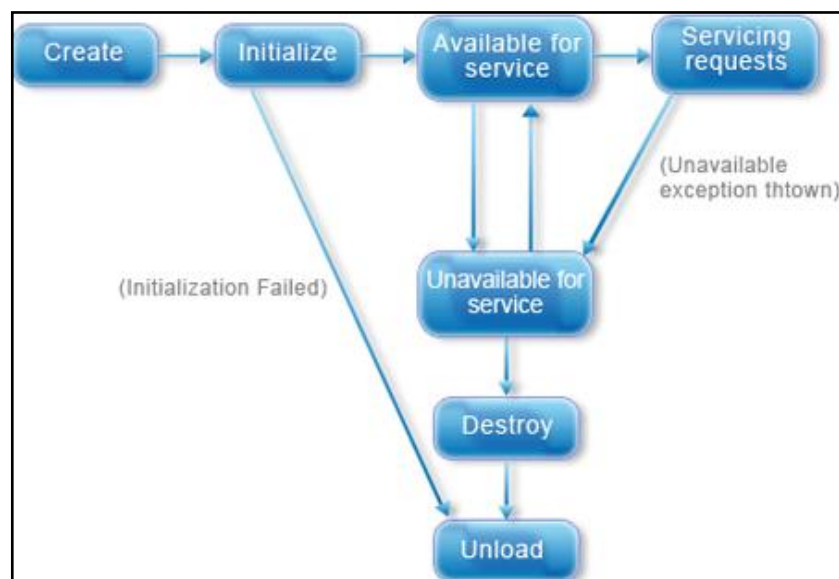
**Web Container:** It is web server that supports servlet execution. Individual Servlets are registered with a container. Tomcat is a popular servlet and JSP container.



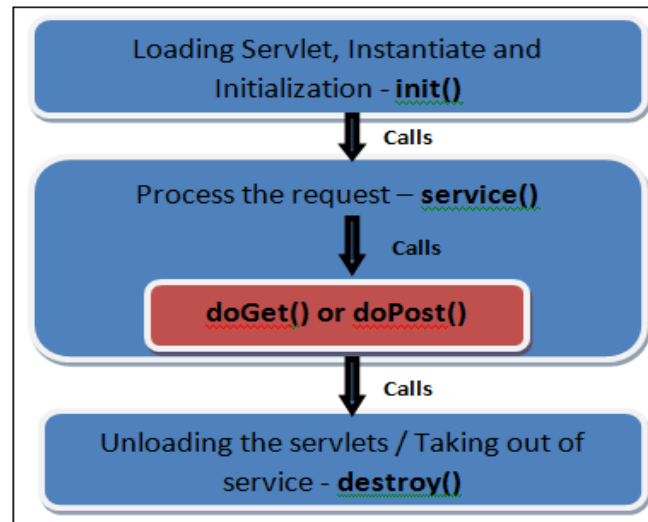
**Servlet Architecture:****Servlets Tasks:**

**Servlets perform the following major tasks:**

- Read the explicit data sent by the clients (browsers). This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP client program.
- Read the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands, and so forth.
- Process the data and generate the results. This process may require talking to a database, executing an RMI or CORBA call, invoking a Web service, or computing the response directly.
- Send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.
- Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.

**Life Cycle of Servlet**

**Life Cycle**

**Steps:**

The sequence in which the Web container calls the life cycle methods of a servlet is:

1. The Web container loads the servlet class and creates one or more instances of the servlet class.
2. The Web container invokes `init()` method of the servlet instance during initialization of the servlet. The `init()` method is invoked only once in the servlet life cycle.
3. The Web container invokes the `service()` method to allow a servlet to process a client request.
4. The `service()` method processes the request and returns the response back to the Web container.
5. The servlet then waits to receive and process subsequent requests as explained in steps 3 and 4.
6. The Web container calls the `destroy()` method before removing the servlet instance from the service. The `destroy()` method is also invoked only once in a servlet life cycle.

**The `init()` method :**

- The `init` method is designed to be called only once. It is called when the servlet is first created, and not called again for each user request.
- The servlet is normally created when a user first invokes a URL corresponding to the servlet.
- When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to `doGet` or `doPost` as appropriate.
- The `init()` method simply creates or loads some data that will be used throughout the life of the servlet.

```
public void init() throws ServletException {
 // Initialization code...
}
```

**The `service()` method :**

- The `service()` method is the main method to perform the actual task.
- The servlet container (i.e. web server) calls the `service()` method to handle requests coming from the client( browsers) and to write the formatted response back to the client.



- Each time the server receives a request for a servlet, the server spawns a new thread and calls service.
- The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

```
public void service(ServletRequest request,
 ServletResponse response)
 throws ServletException, IOException{
}
```

### The doGet() Method

- The doGet() method processes client request, which is sent by the client, using the HTTP GET method.
- To handle client requests that are received using GET method, we need to override the doGet() method in the servlet class.
- In the doGet() method, we can retrieve the client information of the HttpServletRequest object. We can use the HttpServletResponse object to send the response back to the client.

```
public void doGet(HttpServletRequest request,
 HttpServletResponse response)
 throws ServletException, IOException{
 // Servlet code
}
```

### The doPost() Method:

- The doPost() method handles requests in a servlet, which is sent by the client, using the HTTP POST method.
- For example, if a client is entering registration data in an HTML form, the data can be sent using the POST method.
- Unlike the GET method, the POST request sends the data as part of the HTTP request body. As a result, the data sent does not appear as a part of URL.
- To handle requests in a servlet that is sent using the POST method, we need to override the doPost() method. In the doPost() method, we can process the request and send the response back to the client.

```
public void doPost(HttpServletRequest request,
 HttpServletResponse response)
 throws ServletException, IOException{
 // Servlet code
}
```

### The destroy() method :

- The destroy() method is called only once at the end of the life cycle of a servlet.
- This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.
- After the destroy() method is called, the servlet object is marked for garbage collection.

```
public void destroy()
{
 // Finalization code...
}
```



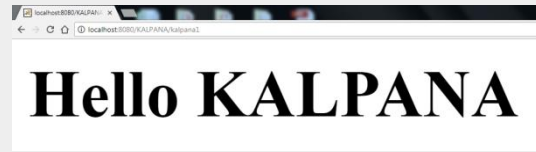
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {
 private String message;

 public void init() throws ServletException {
 // Do required initialization
 message = "Hello KALPANA";
 }

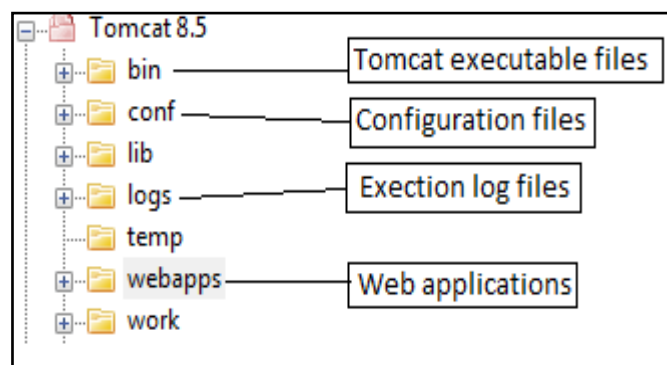
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 // Set response content type
 response.setContentType("text/html");
 // Actual logic goes here.
 PrintWriter out = response.getWriter();
 out.println("<h1>" + message + "</h1>");
 }

 public void destroy() {
 // do nothing.
 }
}
```



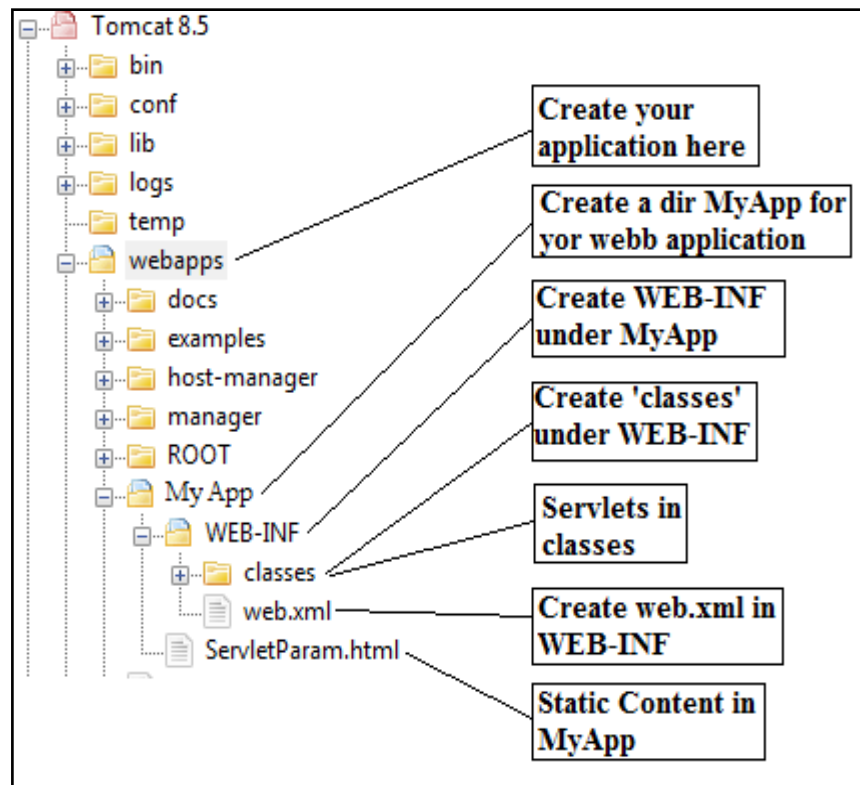
### Deploying a Servlet- Steps:

1. Download and install the Java Software Development kit(SDK).
2. Download a server(Tomcat).
3. Configure the server
  - After installation Tomcat folder will contain "Start Tomcat" and "Stop Tomcat" shortcuts.
  - The JAVA\_HOME environment variable should be set so that Tomcat can find JDK  
JAVA\_HOME = c:\jdk1.5
4. Setup deployment environment



**Tomcat Directory Structure**

- To set up a new application, add a directory under the **webapps** directory and create a subdirectory called **WEB-INF**.
- WEB-INF needs to contain **web.xml** (servlet configuration file)
- After WEB-INF dir is created, create a subdirectory **classes** under it. Java classes will go under this directory.



### 5. Creating ServletDemoServlet

There are three different ways to create a servlet.

- By implementing **Servlet** interface
- By extending **GenericServlet** class
- By extending **HttpServlet** class

6. Compile Servlet and save the class file in **classes** folder.

7. Create a Deployment Descriptor

- The **deployment descriptor** is an xml file, from which Web Container gets the information about the servlet to be invoked.
  - The web container uses the Parser to get the information from the web.xml file.
  - Add a servlet entry and a servlet-mapping entry for each servlet for Tomcat to run.
- Add entries after <web-app> tag inside web.xml

```
<web-app>
 <servlet>
 <servlet-name>Demo</servlet-name>
 <servlet-class>DemoServlet</servlet-class>
 </servlet>
 <servlet-mapping>
 <servlet-name>Demo</servlet-name>
 <url-pattern>/welcome</url-pattern>
 </servlet-mapping>
</web-app>
```

8. Start Tomcat server

9. Open browser and type <http://localhost/MyApp/DemoServlet>

## Servlet API

Servlet API consists of two important packages that encapsulates all the important classes and interface, namely :

1. **javax.servlet**
2. **javax.servlet.http**

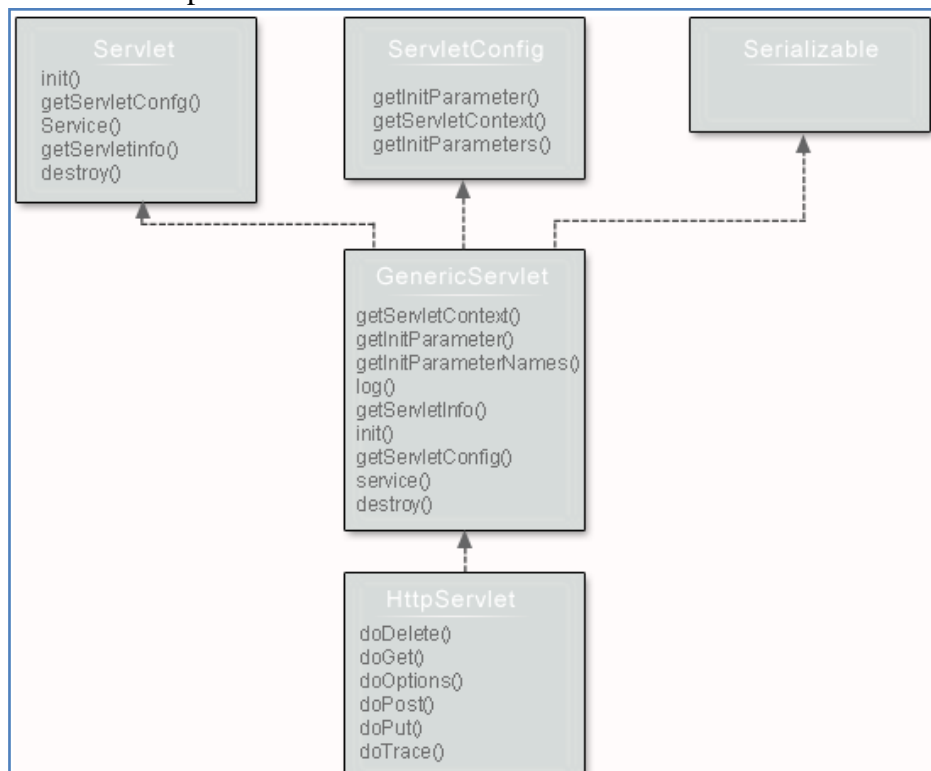
### 1. javax.servlet

#### **Interfaces**

1. Servlet – Declares life cycle methods for a servlet.
2. ServletConfig – To get initialization parameters
3. ServletContext- To log events and access information
4. ServletRequest- To read data from a client request
5. ServletResponse – To write data from client response

#### **Classes**

1. GenericServlet – Implements Servlet and ServletConfig
2. ServletInputStream – Provides an input stream for reading client requests.
3. ServletOutputStream - Provides an output stream for writing responses to a client.
4. ServletException – Indicates servlet error occurred.
5. UnavailableException - Indicates servlet is unavailable



## Servlet Interface

- Servlet interface provides common behaviour to all the servlets.
- Servlet interface needs to be implemented for creating any servlet (either directly or indirectly).
- It provides 3 life cycle methods that are used to initialize the servlet, to service the requests, and to destroy the servlet and 2 non-life cycle methods.

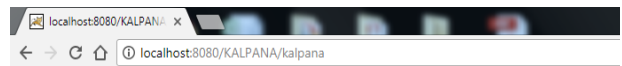
Method	Description
public void init(ServletConfig config)	initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once.
public void service(ServletRequest request, ServletResponse response)	provides response for the incoming request. It is invoked at each request by the web container.
public void destroy()	is invoked only once and indicates that servlet is being destroyed.
public ServletConfig getServletConfig()	returns the object of ServletConfig.
public String getServletInfo()	returns information about servlet such as writer, copyright, version etc.

```

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("hello KALPANA");
 out.print("</body></html>");
 }
 public void destroy(){
 System.out.println("servlet is destroyed");
 }
 public ServletConfig getServletConfig(){
 return config;
 }
 public String getServletInfo(){
 return "copyright 2007-1010";
 }
}

```



# hello KALPANA

### ServletConfig interface

- When the **Web Container** initializes a servlet, it creates a **ServletConfig** object for the servlet.
- ServletConfig object is used to pass information to a servlet during initialization by getting configuration information from **web.xml**(Deployment Descriptor).

**Methods**

- `getInitParameter(String name)`: returns a String value initialized parameter
- `getInitParameterNames()`: returns the names of the servlet's initialization parameters as an Enumeration of String objects
- `getServletContext()`: returns a reference to the ServletContext
- `getServletName()`: returns the name of the servlet instance

**ServletContext Interface**

- For every **Web application** a **ServletContext** object is created by the web container.
- ServletContext object is used to get configuration information from **Deployment Descriptor**(web.xml) which will be available to any servlet.

**Methods :**

- `getAttribute(String name)` - returns the container attribute with the given name
- `getInitParameter(String name)` - returns parameter value for the specified parameter name
- `getInitParameterNames()` - returns the names of the context's initialization parameters as an Enumeration of String objects
- `setAttribute(String name, Object obj)` - set an object with the given attribute name in the application scope
- `removeAttribute(String name)` - removes the attribute with the specified name from the application context

**Servlet RequestInterface**

- True job of a Servlet is to handle client request.
- Servlet API provides two important interfaces **javax.servlet.ServletRequest** to encapsulate client request.
- Implementation of these interfaces provides important information about client request to a servlet.

**Methods**

- `getAttribute(String name)`, `removeAttribute(String name)`, `setAttribute(String name, Object o)`, `getAttributeName()` – used to store and retrieve an attribute from request.
- `getParameter(String name)` - returns value of parameter by name
- `getParameterNames()` - returns an enumeration of all parameter names
- `getParameterValues(String name)` - returns an array of String objects containing all of the values the given request parameter has, or null if the parameter does not exist

**Servlet ResponseInterface**

- Servlet API provides `ServletResponse` to assist in sending response to client.

**Methods**

- `getWriter()` - returns a `PrintWriter` object that can send character text to the client.
- `setContentType(String type)` - sets the content type of the response being sent to the client before sending the respond.

### GenericServlet class

- GenericServlet class implements **Servlet**, **ServletConfig** and **Serializable** interfaces.
- It provides the implementation of all the methods of these interfaces except the service method.
- GenericServlet class can handle any type of request so it is protocol-independent.
- You may create a generic servlet by inheriting the GenericServlet class and providing the implementation of the service method.

### **Methods**

- **public void init(ServletConfig config)** is used to initialize the servlet.
- **public abstract void service(ServletRequest request, ServletResponse response)** provides service for the incoming request. It is invoked at each time when user requests for a servlet.
- **public void destroy()** is invoked only once throughout the life cycle and indicates that servlet is being destroyed.
- **public ServletConfig getServletConfig()** returns the object of ServletConfig.
- **public String getServletInfo()** returns information about servlet such as writer, copyright, version etc.
- **public void init()** it is a convenient method for the servlet programmers, now there is no need to call super.init(config)
- **public ServletContext getServletContext()** returns the object of ServletContext.
- **public String getInitParameter(String name)** returns the parameter value for the given parameter name.
- **public Enumeration getInitParameterNames()** returns all the parameters defined in the web.xml file.
- **public String getServletName()** returns the name of the servlet object.
- **public void log(String msg)** writes the given message in the servlet log file.
- **public void log(String msg, Throwable t)** writes the explanatory message in the servlet log file and a stack trace.

### ServletInputStream Class

- It provides stream to read binary data such as image etc. from the request object. It is an abstract class.
- The **getInputStream()** method of **ServletRequest** interface returns the instance of ServletInputStream class
- **intreadLine(byte[] b, int off, intlen)** it reads the input stream.

### ServletOutputStream Class

- It provides a stream to write binary data into the response. It is an abstract class.
- The **getOutputStream()** method of **ServletResponse** interface returns the instance of ServletOutputStream class.
- ServletOutputStream class provides print() and println() methods that are overloaded.

### ServletException and UnavailableException

- ServletException is a general exception that the servlet container will catch and log. The cause can be anything.

- The exception contains a root cause exception.
- Defines an exception that a servlet or filter throws to indicate that it is permanently or temporarily unavailable.
- When a servlet or filter is permanently unavailable, something is wrong with it, and it cannot handle requests until some action is taken. For example, a servlet might be configured incorrectly, or a filter's state may be corrupted.

## 2. javax.servlet.http

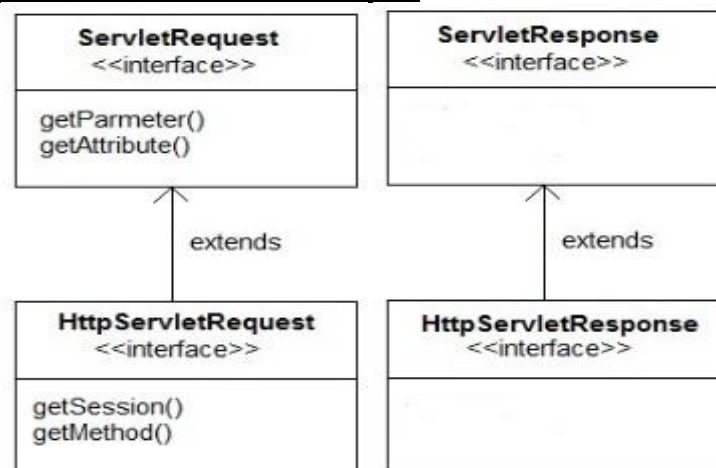
### Interfaces

1. HttpServletRequest
2. HttpServletResponse
3. HttpSession

### Classes

1. HttpServlet
2. Cookie

### HttpServletRequest and HttpServletResponse



- **HttpServletRequest** Extends the ServletRequest interface to provide request information for HTTP servlets.
- The servlet container creates an HttpServletRequest object and passes it as an argument to the servlet's service methods (doGet, doPost, etc).
- It Contains all the client's request information.
- The HttpServletRequest breaks a request down into parsed elements, such as request URI, query arguments and headers. Various get methods allow you to access different parts of the request.

#### 1. requestURI – URL sent by browser

#### 2. Parameters -

The `HttpServletRequest` provides methods for accessing parameters of a request. The methods `getParameter()`, `getParameterValues()` and `getParameterNames()` are offered as ways to access the arguments.

**3. Attributes –**

The request object defines a method called `getAttribute()`. The servlet interface provides this as a way to include extra information about the request that is not covered by any of the other `HttpServletRequest` methods.

**4. ServletInputStream –**

The `ServletInputStream` is an `InputStream` that allows your servlets to read all of the request's input following the headers.

- ***HTTPServletResponse*** Extends the `ServletResponse` interface and can perform these tasks

**1. Set Response Codes –**

The response code for a request is a numeric value that represents the status of the response. For example, 200 represents a successful response, 404 represents a file not found.

**2. Set Headers –**

Headers for the response can be set by calling `setHeader`, specifying the name and value of the header to be set.

**3. Send Redirects –**

The `sendRedirect` method is used to issue a redirect to the browser, causing the browser to issue a request to the specified URL. The URL passed to `sendRedirect` must be an absolute URL—it must include protocol, machine, full path, and so on.

**4. Set ServletOutputStream –**

The `ServletOutputStream` is obtained by calling `getOutputStream` on the `HttpServletResponse`. It is a subclass of `OutputStream` that contains a number of convenient print and `println` methods. Data written to the `ServletOutputStream` goes straight back to the browser.

**HTTPSession**

- **HttpSession** object is used to store entire session with a specific client.
- We can store, retrieve and remove attribute from **HttpSession** object.
- Any servlet can have access to **HttpSession** object throughout the `getSession()` method of the **HttpServletRequest** object.

**HTTPServlet**

- `HttpServlet` extends from `GenericServlet` and does not override `init`, `destroy` and other methods.
- It implements the `service()` method which is abstract method in `GenericServlet`.
- A subclass of `HttpServlet` must override at least one method, usually one of these:
  - `doGet()`, if the servlet supports HTTP GET requests
  - `doPost()`, for HTTP POST requests
  - `doPut()`, for HTTP PUT requests
  - `doDelete()`, for HTTP DELETE requests
  - `init()` and `destroy()`, to manage resources that are held for the life of the servlet
  - `getServletInfo()`, which the servlet uses to provide information about itself



### Cookie

- A **cookie** is a small piece of information that is persisted between the multiple client requests.
- **javax.servlet.http.Cookie** class provides the functionality of using cookies. It provides a lot of useful methods for cookies.
- **public void addCookie(Cookie ck):** method of HttpServletResponse interface is used to add cookie in response object.
- **public Cookie[] getCookies():** method of HttpServletRequest interface is used to return all the cookies from the browser.

### Reading Servlet Parameters(or) Handling HTTPRequest and HTTPResponse

- The parameters are the way in which a client or user can send information to the Http Server.
- The **HttpServletRequest** interface includes methods that allow you to read the names and values of parameters that are included in a client request.
- The **HttpServletResponse** Interface provides functionality for sending response to client.
- The browser uses two methods to pass this information to web server. These methods are GET Method and POST Method.

#### **GET method:**

- The GET method sends the encoded user information appended to the page request.
- The page and the encoded information are separated by the ? character as follows:

**http://www.test.com/hello?key1=value1&key2=value2**

- The GET method is the default method to pass information from browser to web server.
- Never use the GET method if you have password or other sensitive information to pass to the server.
- The GET method has size limitation: only 1024 characters can be in a request string.
- This information is passed using QUERY\_STRING header and will be accessible through QUERY\_STRING environment variable.
- Servlet handles this type of requests using **doGet()** method.

#### **POST method:**

- A generally more reliable method of passing information to a backend program is the POST method.
- This message comes to the backend program in the form of the standard input which you can parse and use for your processing.
- Servlet handles this type of requests using **doPost()** method.

### **Reading Form Data using Servlet:**

Servlets handles form data parsing automatically using the following methods depending on the situation:

- **getParameter():** You call request.getParameter() method to get the value of a form parameter.
- **getParameterValues():** Call this method if the parameter appears more than once and returns multiple values, for example checkbox.
- **getParameterNames():** Call this method if you want a complete list of all parameters in the current request.

**Sending Data to Client:**

Obtain a PrintWriter object HttpServletResponse that can send character text to the client.

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

**POST method example**

Let us consider **HelloForm.java**

```
import java.io.*;
import java.util.*;
import javax.servlet.http.*;
public class HelloForm extends HttpServlet {
 public void doPost(HttpServletRequest request, HttpServletResponse response) throws
 IOException, ServletException {
 PrintWriter pw = response.getWriter();
 pw.print("<html><body>");
 pw.print("Name: "+request.getParameter("first_name")+
 " "+request.getParameter("last_name"));
 pw.print("</body></html>");
 pw.close();
 }
}
```

- Compile HelloForm.java as follows: \$javac HelloForm.java
- Compilation would produce **HelloForm.class** file.
- Next you would have to copy this class file in  
    <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/classes
- Create following entries in **web.xml** file located in  
    <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/

```
<servlet>
 <servlet-name>HelloForm</servlet-name>
 <servlet-class>HelloForm</servlet-class>
</servlet>
```

```
<servlet-mapping>
 <servlet-name>HelloForm</servlet-name>
 <url-pattern>/HelloForm</url-pattern>
</servlet-mapping>
```

- Now create a HTML page Hello.html and put it in  
    <Tomcat-installation-directory>/webapps/ROOT directory

```
<html>
<body>
 <form action="HelloForm" method="GET">
 First Name: <input type="text" name="first_name">

 Last Name: <input type="text" name="last_name"/>

 <input type="submit" value="Submit"/>
 </form>
</body>
</html>
```

- When you access ***http://localhost:8080/Hello.html***, then output of the above form.

First Name:	<input type="text" value="Kalpana"/>
Last Name:	<input type="text" value="Mrcet"/>
<input type="button" value="Submit"/>	

- Start Tomcat Server and open browser.
- Now enter firstname and lastname, Click Submit
- It will generate result

Name: KalpanaMrcet

## **Reading Initialization Parameters**

### **1. Using Servlet Config:**

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

#### **Methods**

- `getInitParameter(String name)`: returns a String value initialized parameter
- `getInitParameterNames()`: returns the names of the servlet's initialization parameters as an Enumeration of String objects
- `getServletContext()`: returns a reference to the ServletContext
- `getServletName()`: returns the name of the servlet instance

#### **Syntax to provide the initialization parameter for a servlet**

The `init-param` sub-element of `servlet` is used to specify the initialization parameter for a servlet.

```
<web-app>
 <servlet>

 <init-param>
 <param-name>email</param-name>
 <param-value>kalpana@gamil.com</param-value>
 </init-param>

 </servlet>
</web-app>
```

#### **Retrieve ServletConfig**

```
ServletConfig sc = getServletConfig();
out.println(sc.getInitParameter("email"));
```

**Ex: web.xml**


```
<web-app>
 <servlet>
 <servlet-name>TestInitParam</servlet-name>
 <servlet-class>TestInitParam</servlet-class>
 <init-param>
 <param-name>email</param-name>
 <param-value>kalpana@gmail.com</param-value>
 </init-param>
 </servlet>
 <servlet-mapping>
 <servlet-name>TestInitParam</servlet-name>
 <url-pattern>/TestInitParam</url-pattern>
 </servlet-mapping>
</web-app>
```

**TestInitParam.java**

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

public class TestInitParam extends HttpServlet {
 protected void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html;charset=UTF-8");
 PrintWriter out = response.getWriter();
 ServletConfig sc = getServletConfig();
 out.print("<html><body>");
 out.print("" + sc.getInitParameter("email") + "");
 out.print("</body></html>");
 out.close();
 }
}
```

- It will generate result



**kalpana@gmail.com**

**2. Using ServletContext**

- An object of ServletContext is created by the web container at time of deploying the project.
- This object can be used to get configuration information from web.xml file.
- There is only one ServletContext object per web application.
- If any information is shared to many servlet, it is better to provide it from the web.xml file using the **<context-param>** element.

## Advantage

- **Easy to maintain** if any information is shared to all the servlet, it is better to make it available for all the servlet.
- We provide this information from the web.xml file, so if the information is changed, we don't need to modify the servlet. Thus it removes maintenance problem.

## Uses

1. The object of ServletContext provides an interface between the container and servlet.
2. The ServletContext object can be used to get configuration information from the web.xml file.
3. The ServletContext object can be used to set, get or remove attribute from the web.xml file.
4. The ServletContext object can be used to provide inter-application communication.

## Methods:

- `getAttribute(String name)` - returns the container attribute with the given name
- `getInitParameter(String name)` - returns parameter value for the specified parameter name
- `getInitParameterNames()` - returns the names of the context's initialization parameters as an Enumeration of String objects
- `setAttribute(String name, Object obj)` - set an object with the given attribute name in the application scope
- `removeAttribute(String name)` - removes the attribute with the specified name from the application context

## Retrieve ServletContext

`ServletContextapp = getServletContext();`

*OR*

`ServletContextapp = getServletConfig().getServletContext();`

## Ex: web.xml

```
<web-app>
 <context-param>
 <param-name>driverName</param-name>
 <param-value>sun.jdbc.JdbcOdbcDriver</param-value>
 </context-param>

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

**TestServletContext.java**

```

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

public class TestServletContext extends HttpServlet {
 protected void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html;charset=UTF-8");
 PrintWriter out = response.getWriter();
 ServletContext sc = getServletContext();
 out.println(sc.getInitParameter("driverName"));
 }
}

```

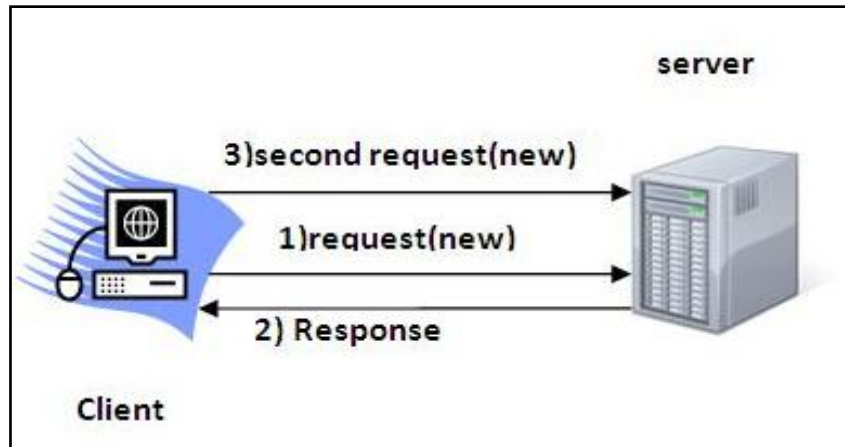
- It will generate result

sun.jdbc.JdbcOdbcDriver

Context Init parameters	Servlet Init parameter
Available to all servlets and JSPs that are part of web	Available to only servlet for which the <init-param> was configured
Context Init parameters are initialized within the <web-app> not within a specific <servlet> elements	Initialized within the <servlet> for each specific servlet.
ServletContext object is used to get Context Init parameters	ServletConfig object is used to get Servlet Init parameters
Only one ServletContext object for entire web app	Each servlet has its own ServletConfig object

**Session Tracking**

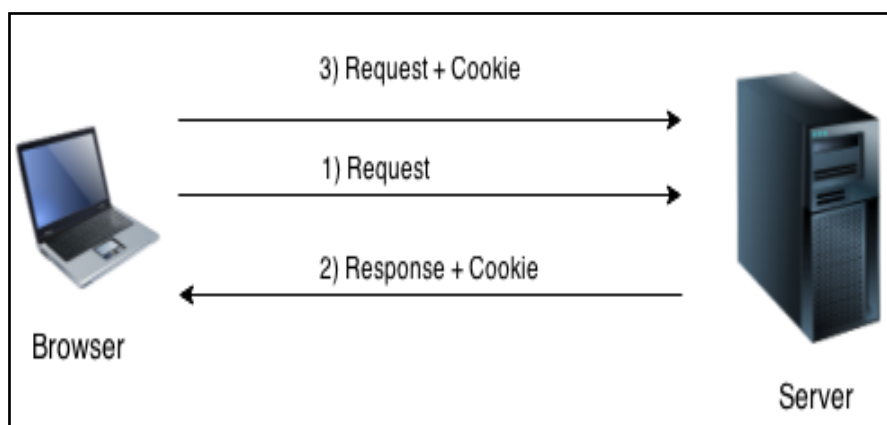
- Session simply means a particular interval of time.
- Session Tracking is a way to maintain state (data) of an user.
- Http protocol is a stateless, each request is considered as the new request, so we need to maintain state using session tracking techniques.
- Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.



- We use session tracking to recognize the user. It is used to recognize the particular user.
- Session Tracking Techniques
  - Cookies
  - Hidden Form Field
  - URL Rewriting
  - HttpSession

### Cookies

- Cookies are text files stored on the client computer and they are kept for various information tracking purpose.
- There are three steps involved in identifying returning users:
  - Server script sends a set of cookies to the browser in response header.
  - Browser stores this information on local machine for future use.
  - When next time browser sends any request to web server then it sends those cookies information to the server in request header and server uses that information to identify the user.
- Cookies are created using **Cookie** class present in Servlet API.
- For adding cookie or getting the value from the cookie, we need some methods provided by other interfaces. They are:
  - a. **public void addCookie(Cookie ck):** method of HttpServletResponse interface is used to add cookie in response object.
  - b. **public Cookie[] getCookies():** method of HttpServletRequest interface is used to return all the cookies from the browser.



**Disadvantage of Cookies**

- It will not work if cookie is disabled from the browser.
- Only textual information can be set in Cookie object.

**Methods**

public void setMaxAge(int expiry)	Sets the maximum age of the cookie in seconds.
public String getName()	Returns the name of the cookie. The name cannot be changed after creation.
public String getValue()	Returns the value of the cookie.
public void setName(String name)	changes the name of the cookie.
public void setValue(String value)	changes the value of the cookie.

**Create Cookie**

```
Cookie ck=new Cookie("user","kalpana ");//creating cookie object
response.addCookie(ck);//adding cookie in the response
```

**Delete Cookie**

It is mainly used to logout or signout the user.

```
Cookie ck=new Cookie("user","");//deleting value of cookie
ck.setMaxAge(0);//changing the maximum age to 0 seconds
response.addCookie(ck);//adding cookie in the response
```

**Get Cookies**

```
Cookie ck[]=request.getCookies();
for(int i=0;i<ck.length;i++)
 out.print("
" +ck[i].getName()+" "+ck[i].getValue());

//printing name and value of cookie
```

**Sending the Cookie into the HTTP response headers:**

We use **response.addCookie** to add cookies in the HTTP response header as follows:

```
response.addCookie(cookie);
```

**Ex: List and AddCookie.java**

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ListandAddCookie extends HttpServlet {
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 Cookie cookie = null;
 out.println("<html><body>" +
 "<form method='get' action='/mrcet/CookieLab'>" +
 "Name:<input type='text' name='user' />
" +
```



```
"Password:<input type='text' name='pass' >
" +
"<input type='submit' value='submit'>" +
"</form>");
```

```
String name = request.getParameter("user");
String pass = request.getParameter("pass");
```

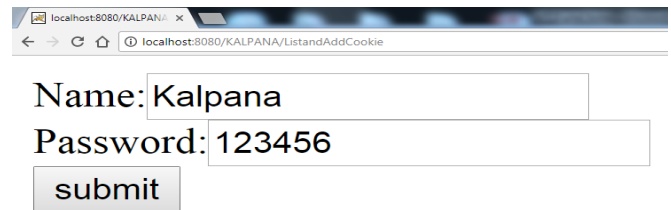
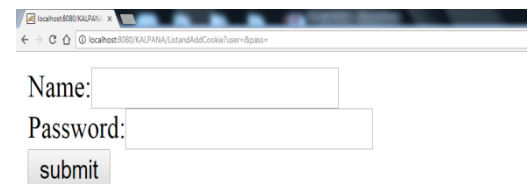
```
if(!pass.equals("") || !name.equals("")) {
 Cookie ck = new Cookie(name,pass);
 response.addCookie(ck);
}
```

```
Cookie[] cookies = request.getCookies();
if(cookies != null){
 out.println("<h2> Found Cookies Name and Value</h2>");
 for (inti = 0; i<cookies.length; i++){
 cookie = cookies[i];
 out.print("Cookie Name : " + cookie.getName() + ", ");
 out.print("Cookie Value: " + cookie.getValue()+"
");
 }
}
out.println("</body></html>");
```

```
}
```

#### web.xml

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

### Found Cookies Name and Value

Cookie Name : Kalpana, Cookie Value: 123456

### Session

- HttpSession Interface provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.
- Web container creates a session id for each user. The container uses this id to identify the particular user.
- The servlet container uses this interface to create a session between an HTTP client and an HTTP server.
- The session persists for a specified time period, across more than one connection or page request from the user.

### Get the HttpSession object

The HttpServletRequest interface provides two methods to get the object of HttpSession:

1. **public HttpSession getSession():** Returns the current session associated with this request, or if the request does not have a session, creates one.
2. **public HttpSession getSession(boolean create):** Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.

### Destroy Session

```
session.invalidate();
```

### Set/Get data in session

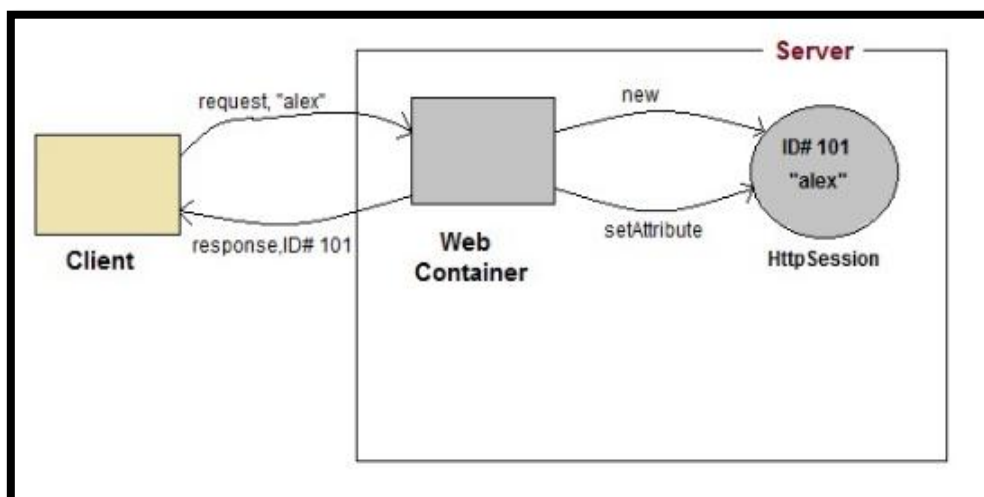
```
session.setAttribute(name,value);
session.getAttribute(name);
```

### Methods

1. **public String getId():** Returns a string containing the unique identifier value.
2. **public long getCreationTime():** Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
3. **public long getLastAccessedTime():** Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.
4. **public void invalidate():** Invalidates this session then unbinds any objects bound to it.

### Steps

- On client's first request, the **Web Container** generates a unique session ID and gives it back to the client with response. This is a temporary session created by web container.
- The client sends back the session ID with each request. Making it easier for the web container to identify where the request is coming from.
- The **Web Container** uses this ID, finds the matching session with the ID and associates the session with the request.



**Ex: SessionTrack.java**

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

public class SessionTrack extends HttpServlet {

 public void doGet(HttpServletRequest request,
 HttpServletResponse response) throws ServletException, IOException {
 // Create a session object if it is already not created.
 HttpSession session = request.getSession(true);

 String title = "Welcome to my website";
 String userID = "";
 Integer visitCount = new Integer(0);

 if (session.isNew())
 {
 userID = "Kalpana";
 session.setAttribute("UserId", "Kalpana");
 }
 else {
 visitCount = (Integer)session.getAttribute("visitCount");
 visitCount = visitCount + 1;
 userID = (String)session.getAttribute("UserId");
 }
 session.setAttribute("visitCount", visitCount);

 response.setContentType("text/html");
 PrintWriter out = response.getWriter();

 out.println("<html>" +
 "<body>" +
 "<h1>Session Infomation</h1>" +
 "<table border='1'>" +
 "<tr><th>Session info</th><th>value</th></tr>" +
 "<tr><td>id</td><td>" + session.getId() + "</td></tr>" +
 "<tr><td>User ID</td><td>" + userID + "</td></tr>" +
 "<tr><td>Number of visits</td><td>" + visitCount + "</td></tr>" +
 "</table></body></html>");
 }
}
```

**web.xml**

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

**Output:**

## UNIT IV

### JAVA SERVER PAGES

**Introduction to JSP:** The Problem with Servlet. The Anatomy of a JSP Page, JSP Processing. JSP Application Design with MVC Setting Up and JSP Environment, JSP Declarations, Directives, Expressions, Code Snippets, implement objects, Requests, Using Cookies and Session for Session

The Servlet technology and JavaServer Pages (JSP) are the two main technologies for developing java Web applications. When first introduced by Sun Microsystems in 1996, the Servlet technology was considered superior to the reigning Common Gateway Interface (CGI) because servlets stay in memory after they service the first requests. Subsequent requests for the same servlet do not require instantiation of the servlet's class therefore enabling better response time.

Servlets are Java classes that implement the `javax.servlet.Servlet` interface. They are compiled and deployed in the web server. The problem with servlets is that you embed HTML in Java code. If you want to modify the cosmetic look of the page or you want to modify the structure of the page, you have to change code. Generally speaking, this is left to the better hands (and brains) of a web page designer and not to a Java developer.

```
PrintWriter pw = response.getWriter();
pw.println("<html><head><title>Testing</title></head>"); pw.println("<body
bgcolor=\"# ffdddd\">");
```

As seen from the example above this method presents several difficulties to the web developer:

1. The code for a servlet becomes difficult to understand for the programmer.
2. The HTML content of such a page is difficult if not impossible for a web designer to understand or design.
3. This is hard to program and even small changes in the presentation, such as the page's background color, will require the servlet to be recompiled. Any changes in the HTML content require the rebuilding of the whole servlet.
4. It's hard to take advantage of web-page development tools when designing the application interface. If such tools are used to develop the web page layout, the generated HTML must then be manually embedded into the servlet code, a process which is time consuming, error prone, and extremely boring.
5. In many Java servlet-based applications, processing the request and generating the response are both handled by a single servlet class.
6. The servlet contains request processing and business logic (implemented by methods), and also generates the response HTML code, are embedded directly in the servlet code.

JSP solves these problems by giving a way to include java code into an HTML page using scriptlets. This way the HTML code remains intact and easily accessible to web designers, but the page can still perform its task.

In late 1999, Sun Microsystems added a new element to the collection of Enterprise Java tools: JavaServer Pages (JSP). JavaServer Pages are built on top of Java servlets and designed to increase the efficiency in which programmers, and even nonprogrammers, can create web content.

Instead of embedding HTML in the code, you place all static HTML in a JSP page, just as in a regular web page, and add a few JSP elements to generate the dynamic parts of the page. The request processing can remain the domain of the servlet, and the business logic can be handled by JavaBeans and EJB components.

A JSP page is handled differently compared to a servlet by the web server. When a servlet is deployed into a web server in compiled (bytecode) form, then a JSP page is deployed in its original, human-readable form.

When a user requests the specific page, the web server compiles the page into a servlet and from there on handles it as a standard servlet.

This accounts for a small delay, when a JSP page is first requested, but any subsequent requests benefit from the same speed effects that are associated with servlets.

### The Problem with Servlet

- Servlets are difficult to code which are overcome in JSP. Other way, we can say, JSP is almost a replacement of Servlets, (by large, the better word is extension of Servlets), where coding decreases more than half.
- In Servlets, both static code and dynamic code are put together. In JSP, they are separated. For example, In Servlets:  
`out.println("Hello Mr." + str + " you are great man");`  
where str is the name of the client which changes for each client and is known as dynamic content. The strings, "Hello Mr." and "you are great man" are static content which is the same irrespective of client. In Servlets, in `println()`, both are put together.
- In JSP, the static content and dynamic content is separated. Static content is written in HTML and dynamic content in JSP. As much of the response comprises of static content (nearly 70%) only, the JSP file more looks as a HTML file.
- Programmer inserts, here and there, chunks of JSP code in a running HTML developed by Designer. As much of the response delivered to client by server comprises of static content (nearly 70%), the JSP file more looks like a HTML file. Other way we can say, JSP is nothing but Java in HTML (servlets are HTML in Java); java code embedded in HTML.
- When the roles of Designer and Programmer are nicely separated, the product development becomes cleaner and fast. Cost of developing Web site becomes cheaper as Designers are much paid less than Programmers, especially should be thought in the present competitive world.
- Both presentation layer and business logic layer put together in Servlets. In JSP, they can be separated with the usage of JavaBeans.
- The objects of `PrintWriter`, `ServletConfig`, `ServletContext`, `HttpSession` and `RequestDispatcher` etc. are created by the Programmer in Servlets and used. But in JSP, they are builtin and are known as "implicit objects". That is, in JSP, Programmer never creates these objects and straightaway use them as they are implicitly created and given by JSP container. This decreases lot of coding.
- JSP can easily be integrated with JavaBeans.
- JSP is much used in frameworks like Struts etc.
- With JSP, Programmer can build custom tags that can be called in JavaBeans directly. Servlets do not have this advantage. Reusability increases with tag libraries and JavaBean etc.
- Writing alias name in `<url-pattern>` tag of `web.xml` is optional in JSP but mandatory in Servlets.
- A Servlet is simply a Java class with extension `.java` written in normal Java code.
- A Servlet is a Java class. It is written like a normal Java. JSP is comes with some elements that are easy to write.

- JSP needs no compilation by the Programmer. Programmer deploys directly a JSP source code file in server where as incase of Servlets, the Programmer compiles manually a Servlet file and deploys a .class file in server.
- JSP is so easy even a Web Designer can put small interactive code (not knowing much of Java) in static Web pages.
- First time when JSP is called it is compiled to a Servlet. Subsequent calls to the same JSP will call the same compiled servlet (instead of converting the JSP to servlet), Ofcourse, the JSP code would have not modified. This increases performance.

## Anatomy of JSP

### Anatomy of a jsp page

```
<%@page contentType = "text/html" language = "java%">
<%@page import = "java.util.Date" session = "false"%>
```

} Jsp elements

**%@ is jsp directive**

```
<html>
<head>
<title> simple jsp page demo</title>
</head>
<body>
<h3> current time is : </h3>
```

} Template data

```
<%= new Date()%> --> jsp elements
```

**%= is jsp element**

```
</body>
</html> Template data
```

## JSP Processing

Once you have a JSP capable web-server or application server, you need to know the following information about it:

- Where to place the files
- How to access the files from your browser (with an http: prefix, not as file:)

You should be able to create a simple file, such as

```
<HTML>
<BODY>
Hello, world
</BODY> </HTML>
```

Know where to place this file and how to see it in your browser with an http:// prefix.

Since this step is different for each web-server, you would need to see the web-server documentation to find out how this is done. Once you have completed this step, proceed to the next.

### Your first JSP

JSP simply puts Java inside HTML pages. You can take any existing HTML page and change its extension to ".jsp" instead of ".html". In fact, this is the perfect exercise for your first JSP. Take the HTML file you used in the previous exercise. Change its extension from ".html" to ".jsp". Now load the new file, with the ".jsp" extension, in your browser.

**You will see the same output, but it will take longer! But only the first time. If you reload it again, it will load normally.**

What is happening behind the scenes is that your JSP is being turned into a Java file, compiled and loaded. This compilation only happens once, so after the first load, the file doesn't take long to load anymore. (But everytime you change the JSP file, it will be re-compiled again.)

Of course, it is not very useful to just write HTML pages with a .jsp extension! We now proceed to see what makes JSP so useful

Adding dynamic content via expressions

As we saw in the previous section, any HTML file can be turned into a JSP file by changing its extension to .jsp. Of course, what makes JSP useful is the ability to embed Java. Put the following text in a file with .jsp extension (let us call it hello.jsp), place it in your JSP directory, and view it in a browser.

```
<HTML>
<BODY>
Hello! The time is now <%= new java.util.Date() %>
</BODY>
</HTML>
```

Notice that each time you reload the page in the browser, it comes up with the current time. The character sequences

<%= and %> enclose Java expressions, which are evaluated at run time.

This is what makes it possible to use JSP to generate dynamic HTML pages that change in response to user actions or vary from user to user.

### Explain about JSP Elements

In this lesson we will learn about the various elements available in JSP with suitable examples. In JSP elements can be divided into 4 different types.

**These are:**

#### 1. Expressions

We can use this tag to output any data on the generated page. These data are automatically converted to string and printed on the output stream.

Syntax of JSP Expressions are: <%= "Any thing" %>

JSP Expressions start with Syntax of JSP Scriptlets are with <%= and ends with %>. Between these this you can put anything and that will convert to the String and that will be displayed.

**Example:** <%= "Hello World!" %> Above code will display 'Hello World!'



## 2. Scriptlets

In this tag we can insert any amount of valid java code and these codes are placed in `_jspService` method by the JSP engine.

### Syntax of JSP Scriptlets are:

```
<% //java codes
%>
```

JSP Scriptlets begins with `<%` and ends `%>`. We can embed any amount of java code in the JSP Scriptlets. JSP Engine places these code in the `_jspService()` method. Variables available to the JSP Scriptlets are:

**a. Request:** Request represents the clients request and is a subclass of `HttpServletRequest`. Use this variable to retrieve the data submitted along the request.

Example: `<% //java codes`

```
String userName=null; userName=request.getParameter("userName");
```

```
%>
```

**b. Response:** Response represents the server response and is a subclass of `HttpServletResponse`.

```
<% response.setContentType("text/html"); %>
```

**c. Session:** represents the HTTP session object associated with the request. Your Session ID: `<%= session.getId() %>`

**d. Out:** out is an object of output stream and is used to send any output to the client.

## 3. Directives

A JSP "directive" starts with `<%@` characters. In the directives we can import packages, define error handling pages or the session information of the JSP page.

### Syntax of JSP directives is:

```
<%@ directive attribute="value" %>
```

**a. page:** page is used to provide the information about it. Example: `<%@ page language="java" %>`

**b. include:** include is used to include a file in the JSP page. Example: `<%@ include file="/header.jsp" %>`

**c. taglib:** taglib is used to use the custom tags in the JSP pages (custom tags allows us to defined our own tags). Example: `<%@ taglib uri="tlds/taglib.tld" prefix="mytag" %>`

Page tag attributes are:

**a. language="java"**

This tells the server that the page is using the java language. Current JSP specification supports only java language. Example: `<%@ page language="java" %>`

**b. extends="mypackage.myclass"**

This attribute is used when we want to extend any class. We can use comma(,) to import more than one packages. Example: `%@ page language="java" import="java.sql.\*" %`

**c. session="true"**

When this value is true session data is available to the JSP page otherwise not. By default this value is true.

Example: `<%@page language="java" session="true" %>`

**d. `errorPage="error.jsp"`**

`errorPage` is used to handle the un-handled exceptions in the page. Example: `<%@page session="true" errorPage="error.jsp" %>`

**e. `contentType="text/html; charset=ISO-8859-1"`**

Use this attribute to set the mime type and character set of the JSP. Example: `<%@page contentType="text/html; charset=ISO-8859-1" %>`

#### 4. Declarations

This tag is used for defining the functions and variables to be used in the JSP. Syntax of JSP Declaratives are:

```
<%!
```

```
//java codes
```

```
%>
```

JSP Declaratives begins with `<%!` and ends `%>` with . We can embed any amount of java code in the JSP Declaratives. Variables and functions defined in the declaratives are class level and can be used anywhere in the JSP page.

**Example**

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

```
<HTML>
```

```
<BODY>
```

```
<%!
```

```
Date theDate = new Date(); Date getDate()
```

```
{
```

```
System.out.println("In getDate() method"); return theDate;
```

```
}
```

```
%>
```

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

```
</BODY>
```

```
</HTML>
```

#### Expalin about Jsp programs?

A Web Page with JSP code

```
<HTML>
```

```
<HEAD>
```

```
<TITLE>A Web Page</TITLE>
```

```
</HEAD>
```

```
<BODY>
```

```
<% out.println("Hello there!"); %>
```

```
</BODY>
```

```
</HTML>
```

**Using a Literal**

```
<HTML>
<HEAD>
<TITLE>Using a Literal</TITLE>
</HEAD>
<BODY>
<H1>Using a Literal</H1>
<%
out.println("Number of days = "); out.println(365);
%>
</BODY>
</html>
```

**Declaration Tag Example**

```
<%!
String name = "Joe";
String date = "8th April, 2002";
%>
<HTML>
<TITLE>Declaration Tag Example</TITLE>
<BODY>
This page was last modified on <%= date %> by <%= name %>.
</BODY>
</HTML>
```

**Embedding Code**

```
<%!
String[] names = { "A", "B", "C", "D" };
%>
<HTML>
<HEAD><TITLE>Embedding Code</TITLE></HEAD>
<BODY>
<H1>List of people</H1>
<TABLE BORDER="1">
<TH>Name</TH>
<% for (int i=0; i<names.length; i++) { %>
<TR><TD><%= names[i] %></TD></TR>
<% } %>
</TABLE>
</BODY>
</HTML>
```

**Use out**

```
<%@ page language="java" %>
<HTML>
<HEAD><TITLE>JSP Example</TITLE></HEAD>
<BODY>
<H1>Quadratic Equation: $y = x^2$ </H1>
<TABLE BORDER="1">
<TH>x</TH><TH>y</TH>
<%
for (int i=0; i<10; i++)
out.print("<TR><TD WIDTH='100'>" + i + "</TD><TD WIDTH='100'>" + (i*i) +
"</TD></TR>");
%>
</TABLE>
</BODY>
</HTML>
```

**Casting to a New Type**

```
<HTML>
<HEAD>
<TITLE>Casting to a New Type</TITLE>
</HEAD>
<BODY>
<H1>Casting to a New Type</H1>
<%
float float1;
double double1 = 1;
float1 = (float) double1;

out.println("float1 = " + float1);
%>
</BODY>
</HTML>
```

**Creating a String**

```
<HTML>
<HEAD>
<TITLE>Creating a String</TITLE>
</HEAD>

<BODY>
<H1>Creating a String</H1>
```

```
<%
String greeting = "Hello from JSP!";
out.println(greeting);
%>
</BODY>
</HTML>
```

**Use for loop to display string array**

```
<% @ page session="false" %>
<%
String[] colors = { "red", "green", "blue" };
for (int i = 0; i < colors.length; i++) { out.print("<P>" + colors[i] + "</p>");
}
%>
```

**Creating an Array**

```
<HTML>
<HEAD>
<TITLE>Creating an Array</TITLE>
</HEAD>
<BODY>
<H1>Creating an Array</H1>
<%
double accounts[];
accounts = new double[100]; accounts[3] = 119.63;
out.println("Account 3 holds $" + accounts[3]);
%>
</BODY>
</HTML>
```

**Using Multidimensional Arrays**

```
<HTML>
<HEAD>
<TITLE>Using Multidimensional Arrays</TITLE>
</HEAD>
<BODY>
<H1>Using Multidimensional Arrays</H1>
<%
double accounts[][] = new double[2][100];
accounts[0][3] = 119.63;
accounts[1][3] = 194.07;
```

```
out.println("Savings Account 3 holds $" + accounts[0][3] + "
"); out.println("Checking
Account 3 holds $" + accounts[1][3]);
%>
</BODY>
</HTML>
```

### Finding a Factorial

```
<HTML>
<HEAD>
<TITLE>Finding a Factorial</TITLE>
</HEAD>
<BODY>
<H1>Finding a Factorial</H1>
<%
int value = 6, factorial = 1, temporaryValue = value;
while (temporaryValue > 0) { factorial *= temporaryValue; temporaryValue--;
}
out.println("The factorial of " + value + " is " + factorial + ".");
%>
</BODY>
</HTML>
```

### Get Form Button Value

```
<HTML>
<HEAD>
<TITLE>Using Buttons</TITLE>
</HEAD>
<BODY>
<H1>Using Buttons</H1>
<FORM NAME="form1" ACTION="basic.jsp" METHOD="POST">
<INPUT TYPE="HIDDEN" NAME="buttonName">
<INPUT TYPE="BUTTON" VALUE="Button 1" ONCLICK="button1()">
<INPUT TYPE="BUTTON" VALUE="Button 2" ONCLICK="button2()">
<INPUT TYPE="BUTTON" VALUE="Button 3" ONCLICK="button3()">
</FORM>
<SCRIPT LANGUAGE="JavaScript">
<!--
function button1()
{
document.form1.buttonName.value = "button 1" form1.submit()
}
-->
```

```
function button2()
{
document.form1.buttonName.value = "button 2" form1.submit()
}
function button3()
{
document.form1.buttonName.value = "button 3" form1.submit()
}
// -->
</SCRIPT>
</BODY>
</HTML>
```

**basic.jsp**

```
<HTML>
<HEAD>
<TITLE>Determining Which Button Was Clicked</TITLE>
</HEAD>
<BODY>
<H1>Determining Which Button Was Clicked</H1> You clicked
<%= request.getParameter("buttonName") %>
</BODY>
</HTML>
```

**Read Form Checkboxes****index.jsp**

```
<HTML>
<HEAD>
<TITLE>Submitting Check Boxes</TITLE>
</HEAD>
<BODY>
<H1>Submitting Check Boxes</H1>
<FORM ACTION="basic.jsp" METHOD="post">
<INPUT TYPE="CHECKBOX" NAME="check1" VALUE="check1" CHECKED>
Checkbox 1

<INPUT TYPE="CHECKBOX" NAME="check2" VALUE="check2">
Checkbox 2

<INPUT TYPE="CHECKBOX" NAME="check3" VALUE="check3">
Checkbox 3
```

```


<INPUT TYPE="SUBMIT" VALUE="Submit">
</FORM>
</BODY>
</HTML>
```

**basic.jsp**

```
<HTML>
<HEAD>
<TITLE>Reading Checkboxes</TITLE>
</HEAD>
<BODY>
<H1>Reading Checkboxes</H1>
<%
if(request.getParameter("check1") != null) { out.println("Checkbox 1 was checked.
");
}
else {
out.println("Checkbox 1 was not checked.
");
}
if(request.getParameter("check2") != null) { out.println("Checkbox 2 was checked.
");
}
else {
out.println("Checkbox 2 was not checked.
");
}
if(request.getParameter("check3") != null) { out.println("Checkbox 3 was checked.
");
}
else {
out.println("Checkbox 3 was not checked.
");
}
%>
</BODY>
</HTML>
```

**Model View Controller**

JSP technology can play a part in everything from the simplest web application to complex enterprise applications. How large a part JSP plays differs in each case, of course. Let introduce a design model called Model- View-Controller (MVC), suitable for both simple and complex applications.

MVC was first described by Xerox in a number of papers published in the late 1980s. The key point of using MVC is to separate logic into three distinct units: the Model, the View, and the Controller. In a server application, we commonly classify the parts of the application as business logic, presentation, and request processing.



Business logic is the term used for the manipulation of an application's data, such as customer, product, and order information. Presentation refers to how the application data is displayed to the user, for example, position, font, and size. And finally, request processing is what ties the business logic and presentation parts together.

In MVC terms, presentation should be separated from the business logic. Presentation of that data (the View) changes fairly often. Just look at all the face-lifts many web sites go through to keep up with the latest fashion in web design. Some sites may want to present the data in different languages or present different subsets of the data to internal and external users.

### cookies:

- A **cookie** is a small piece of information created by a JSP program that is stored in the client's hard disk by the browser. Cookies are used to store various kind of information such as username, password, and user preferences, etc.
- **Different methods in cookie class are:**
  - 1.**String getName()**- Returns a name of cookie
  - 2.**String getValue()**-Returns a value of cookie
  - 3.**int getMaxAge()**-Returns a maximum age of cookie in millisecond
  4. **String getDomain()**-Returns a domain
  - 5.**boolean getSecure()**-Returns true if cookie is secure otherwise false
  - 6.**String getPath()**-Returns a path of cookie
  - 7.**void setPath(String)**- set the path of cookie
  - 8.**void setDomain(String)**-set the domain of cookie
  - 9.**void setMaxAge(int)**-set the maximum age of cookie
  - 10.**void setSecure(Boolean)**-set the secure of cookie.

### **Creating cookie:**

Cookie are created using cookie class constructor.

Content of cookies are added the browser using addCookies() method.

### **Reading cookies:**

Reading the cookie information from the browser using getCookies() method.

Find the length of cookie class.

Retrive the information using different method belongs the cookie class

**PROGRAM: To create and read the cookie for the given cookie name as "EMPID" and its value as"AN2356".**

### **JSP program to create a cookie**

```
<%!
```

```
Cookie c=new Cookie("EMPID","AN2356");
```

```
response.addCookie(c);
```

```
%>
```

**JSP program to read a cookie**

```
<%!
Cookie c[]=request.getCookies();
for(i=0;i<c.length;i++)
{
String name=c[i].getName();
String value=c[i].getValue();
out.println("name="+name);
out.println("value="+value);
}
%>
```

**Session object(session tracking or session uses)**

- The HttpSession object associated to the request
- Session object has a session scope that is an instance of javax.servlet.http.HttpSession class. Perhaps it is the most commonly used object to manage the state contexts.
- This object persist information across multiple user connection.
- Created automatically by
- Different methods of HttpSession interface are as follows:

- 1.**object getAttribute(String)**-Returns the value associated with the name passed as argument.
- 2.**long getCreationTime()**-Returns the time when session created.
- 3.**String getID()**-Returns the session ID
- 4.**long getAccessedTime()**-returns the time when client last made a request for this session.
- 5.**void setAttribute(String,object)**-Associates the values passed in the object name passed.

**Program:**

```
<%!
HttpSession h=req.getSession(true);
Date d=(Date) h.getAttribute("Date");
out.println("last date and time"+d);
Date d1=new Date();
d1=h.setAttribute("date",d1);
out.println("current date and time="+d1);
%>
```

## UNIT-5

### DATABASE ACCESS & JAVA BEANS

**Database Access:** Database Programming using JDBC, JDBC drivers, Studying Javax.sql.\* package, Connecting to database in PHP, Execute Simple Queries, Accessing a Database from a Servlet and JSP page.  
**Java Beans:** Introduction to Beans, Deploying java Beans in a JSP page.

#### What is JDBC Driver?

JDBC drivers implement the defined interfaces in the JDBC API, for interacting with your database server.

For example, using JDBC drivers enable you to open database connections and to interact with it by sending SQL or database commands then receiving results with Java.

The Java.sql package that ships with JDK, contains various classes with their behaviours defined and their actual implementations are done in third-party drivers. Third party vendors implements the java.sql.Driver interface in their database driver.

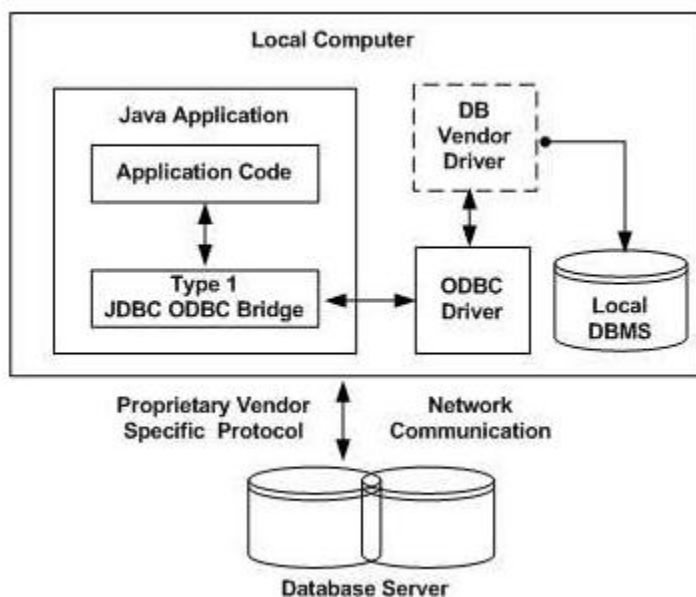
#### JDBC Drivers Types

JDBC driver implementations vary because of the wide variety of operating systems and hardware platforms in which Java operates. Sun has divided the implementation types into four categories, Types 1, 2, 3, and 4, which is explained below –

##### Type 1: JDBC-ODBC Bridge Driver

In a Type 1 driver, a JDBC bridge is used to access ODBC drivers installed on each client machine. Using ODBC, requires configuring on your system a Data Source Name (DSN) that represents the target database.

When Java first came out, this was a useful driver because most databases only supported ODBC access but now this type of driver is recommended only for experimental use or when no other alternative is available.

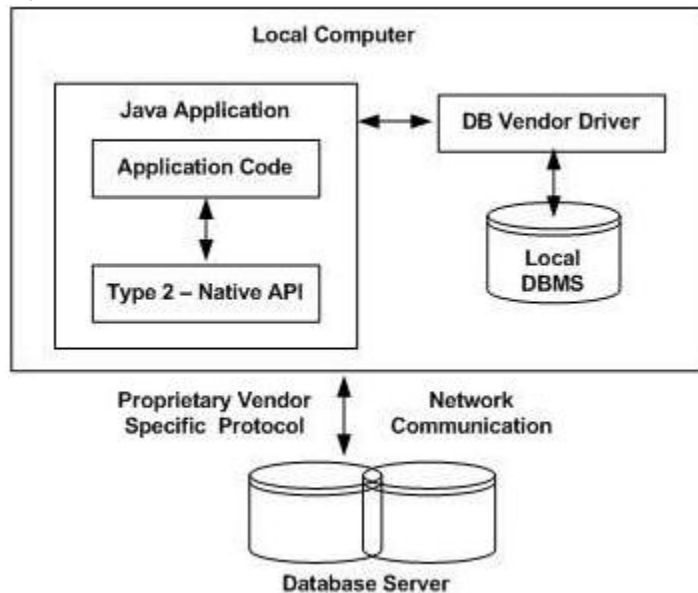


The JDBC-ODBC Bridge that comes with JDK 1.2 is a good example of this kind of driver.

### Type 2: JDBC-Native API

In a Type 2 driver, JDBC API calls are converted into native C/C++ API calls, which are unique to the database. These drivers are typically provided by the database vendors and used in the same manner as the JDBC-ODBC Bridge. The vendor-specific driver must be installed on each client machine.

If we change the Database, we have to change the native API, as it is specific to a database and they are mostly obsolete now, but you may realize some speed increase with a Type 2 driver, because it eliminates ODBC's overhead.

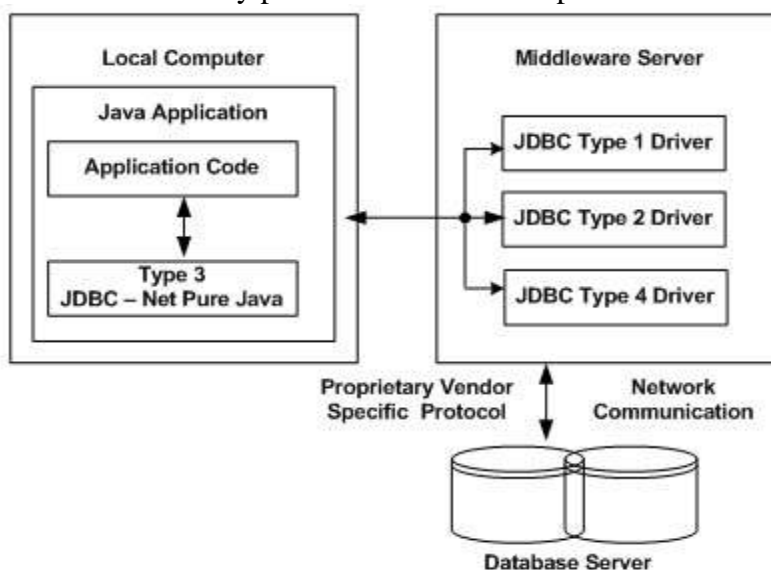


The Oracle Call Interface (OCI) driver is an example of a Type 2 driver.

### Type 3: JDBC-Net pure Java

In a Type 3 driver, a three-tier approach is used to access databases. The JDBC clients use standard network sockets to communicate with a middleware application server. The socket information is then translated by the middleware application server into the call format required by the DBMS, and forwarded to the database server.

This kind of driver is extremely flexible, since it requires no code installed on the client and a single driver can actually provide access to multiple databases.



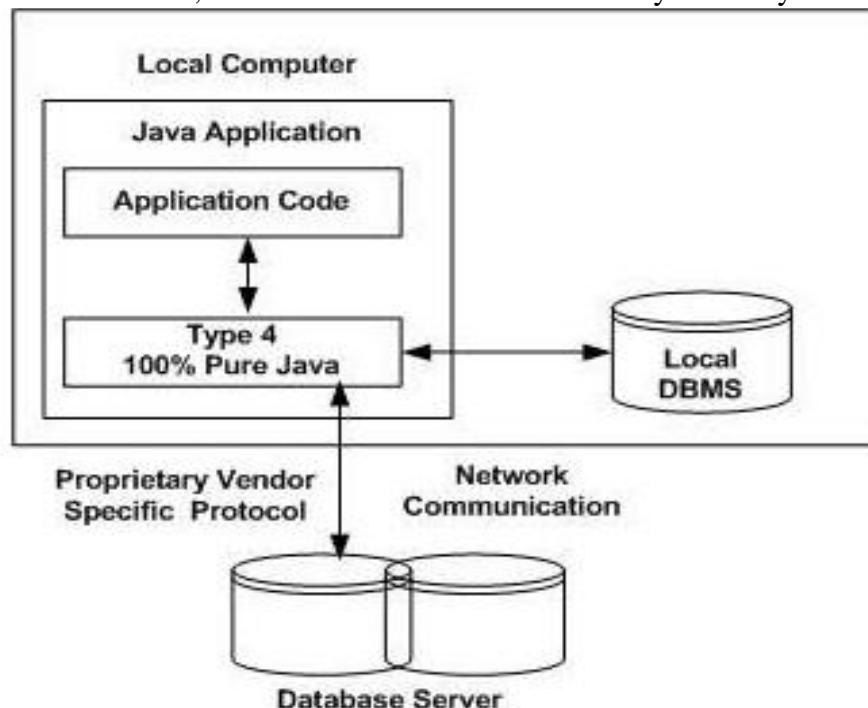
You can think of the application server as a JDBC "proxy," meaning that it makes calls for the client application. As a result, you need some knowledge of the application server's configuration in order to effectively use this driver type.

Your application server might use a Type 1, 2, or 4 driver to communicate with the database, understanding the nuances will prove helpful.

#### **Type 4: 100% Pure Java**

In a Type 4 driver, a pure Java-based driver communicates directly with the vendor's database through socket connection. This is the highest performance driver available for the database and is usually provided by the vendor itself.

This kind of driver is extremely flexible, you don't need to install special software on the client or server. Further, these drivers can be downloaded dynamically.



MySQL's Connector/J driver is a Type 4 driver. Because of the proprietary nature of their network protocols, database vendors usually supply type 4 drivers.

#### **Which Driver should be Used?**

If you are accessing one type of database, such as Oracle, Sybase, or IBM, the preferred driver type is 4.

If your Java application is accessing multiple types of databases at the same time, type 3 is the preferred driver.

Type 2 drivers are useful in situations, where a type 3 or type 4 driver is not available yet for your database.

The type 1 driver is not considered a deployment-level driver, and is typically used for development and testing purposes only.

#### **JDBC( Java Database Connectivity):**

The first thing you need to do is check that you are set up properly. This involves the following steps:

**1. Install Java and JDBC on your machine.**

To install both the Java <sup>tm</sup> platform and the JDBC API, simply follow the instructions for downloading the latest release of the JDK <sup>tm</sup> (Java Development Kit <sup>tm</sup> ). When you download the JDK, you will get JDBC as well.

**2. Install a driver on your machine.**

Your driver should include instructions for installing it. For JDBC drivers written for specific DBMSs, installation consists of just copying the driver onto your machine; there is no special configuration needed.

The JDBC-ODBC Bridge driver is not quite as easy to set up. If you download JDK, you will automatically get the JDBC-ODBC Bridge driver, which does not itself require any special configuration. ODBC, however, does. If you do not already have ODBC on your machine, you will need to see your ODBC driver vendor for information on installation and configuration.

**3. Install your DBMS if needed.**

If you do not already have a DBMS installed, you will need to follow the vendor's instructions for installation. Most users will have a DBMS installed and will be working with an established database.

**Configuring Database:**

Configuring a database is not at all difficult, but it requires special permissions and is normally done by a database administrator.

First, open the control panel. You might find "Administrative tools" select it, again you may find shortcut for "Data Sources (ODBC)". When you open the "Data Source (ODBC)" 32bit ODBC icon, you'll see a "ODBC Data Source Administrator" dialog window with a number of tabs, including "User DSN," "System DSN," "File DSN," etc., in which "DSN" means "Data Source Name." Select "System DSN," and add a new entry there, Select appropriate driver for the data source or directory where database lives. You can name the entry anything you want, assume here we are giving our data source name as "MySource".

**JDBC Database Access**

JDBC was designed to keep simple things simple. This means that the JDBC API makes everyday database tasks, like simple SELECT statements, very easy.

**Import a package `java.sql.*` :** This package provides you set of all classes that enables a network interface between the front end and back end database.

- DriverManager will create a Connection object.
- `java.sql.Connection` interface represents a connection with a specific database. Methods of connection is `close()`, `createStatement()`, `prepareStatement()`, `commit()`, `close()` and `prepareCall()`
- Statement interface used to interact with database via the execution of SQL statements. Methods of this interface are `executeQuery()`, `executeUpdate()`, `execute()` and `getResultSet()`.
- A `ResultSet` is returned when you execute an SQL statement. It maintains a pointer to a row within the tabular results. Methods of this interface are `next()`, `getBoolean()`, `getByte()`, `getDouble()`, `getString()` `close()` and `getInt()`.

**Establishing a Connection**

The first thing you need to do is establish a connection with the DBMS you want to use. This involves two

steps: (1) loading the driver and (2) making the connection.

**Loading Drivers:** Loading the driver or drivers you want to use is very simple and involves just one line of code. If, for example, you want to use the JDBC-ODBC Bridge driver, the following code will load it

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

Your driver documentation will give you the class name to use. For instance, if the class name is `jdbc.DriverXYZ`, you would load the driver with the following line of code:

```
Class.forName("jdbc.DriverXYZ");
```

**Making the Connection:** The second step in establishing a connection is to have the appropriate driver connect to the DBMS. The following line of code illustrates the general idea:

```
Connection con = DriverManager.getConnection(url, "myLogin", "myPassword");
```

If you are using the JDBC-ODBC Bridge driver, the JDBC URL will start with `jdbc:odbc:`. The rest of the URL is generally your data source name or database system. So, if you are using ODBC to access an ODBC data source called "MySource," for example, your JDBC URL could be `jdbc:odbc:MySource`. In place of "myLogin" you put the name you use to log in to the DBMS; in place of "myPassword" you put your password for the DBMS. So if you log in to your DBMS with a login name of "scott" and a password of "tiger" just these two lines of code will establish a connection:

```
String url = "jdbc:odbc:MySource";
Connection con = DriverManager.getConnection(url, "scott", "tiger");
```

The connection returned by the method `DriverManager.getConnection` is an open connection you can use to create JDBC statements that pass your SQL statements to the DBMS. In the previous example, `con` is an open connection, and we will use it in the forthcoming examples.

**Creating JDBC Statements**

A `Statement` object is what sends your SQL statement to the DBMS. You simply create a `Statement` object and then execute it, supplying the appropriate execute method with the SQL statement you want to send. For a `SELECT` statement, the method to use is `executeQuery`. For statements that create or modify tables, the method to use is `executeUpdate`.

It takes an instance of an active connection to create a `Statement` object. In the following example, we use our `Connection` object `con` to create the `Statement` object `stmt`:

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

At this point `stmt` exists, but it does not have an SQL statement to pass on to the DBMS. We need to supply that to the method we use to execute `stmt`.

For example, in the following code fragment, we supply executeUpdate with the SQL statement from the example above:

```
stmt.executeUpdate("CREATE TABLE STUDENT " +
"(S_NAME VARCHAR(32), S_ID INTEGER, COURSE VARCHAR2(10), YEAR
VARCHAR2(3)");
```

Since the SQL statement will not quite fit on one line on the page, we have split it into two strings concatenated by a plus sign (+) so that it will compile. Executing Statements.

Statements that create a table, alter a table, or drop a table are all examples of DDL statements and are executed with the method executeUpdate. The method executeUpdate is also used to execute SQL statements that update a table. In practice, executeUpdate is used far more often to update tables than it is to create them because a table is created once but may be updated many times.

The method used most often for executing SQL statements is executeQuery. This method is used to execute SELECT statements, which comprise the vast majority of SQL statements.

### Entering Data into a Table

We have shown how to create the table STUDENT by specifying the names of the columns and the data types to be stored in those columns, but this only sets up the structure of the table. The table does not yet contain any data. We will enter our data into the table one row at a time, supplying the information to be stored in each column of that row. Note that the values to be inserted into the columns are listed in the same order that the columns were declared when the table was created, which is the default order.

The following code inserts one row of data,

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

```
stmt.executeUpdate("INSERT INTO STUDENT VALUES ('xStudent', 501, '
B.Tech', 'IV')");
```

Note that we use single quotation marks around the student name because it is nested within double quotation marks. For most DBMSs, the general rule is to alternate double quotation marks and single quotation marks to indicate nesting.

The code that follows inserts a second row into the table STUDENT. Note that we can just reuse the Statement object stmt rather than having to create a new one for each execution.

```
stmt.executeUpdate("INSERT INTO STUDENT " + "VALUES ('yStudent', 502,
'B.Tech'. 'III')");
```

### Getting Data from a Table

Now that the table STUDENT has values in it, we can write a SELECT statement to access those values. The star (\*) in the following SQL statement indicates that all columns should be selected. Since there is no WHERE clause to narrow down the rows from which to select, the following SQL statement selects the whole table:

```
SQL> SELECT * FROM STUDENT;
```

### Retrieving Values from Result Sets



We now show how you send the above SELECT statements from a program written in the Java programming language and how you get the results we showed.

JDBC returns results in a ResultSet object, so we need to declare an instance of the class ResultSet to hold our results. The following code demonstrates declaring the ResultSet object rs and assigning the results of our earlier query to it:

```
ResultSet rs = stmt.executeQuery("SELECT S_NAME, YEAR FROM STUDENT");
```

The following code accesses the values stored in the current row of rs. Each time the method next is invoked, the next row becomes the current row, and the loop continues until there are no more rows in rs .

```
String query = "SELECT COF_NAME, PRICE FROM STUDENT";
ResultSet rs = stmt.executeQuery(query);
while (rs.next())
{
String s = rs.getString("S_NAME");
Integer i = rs.getInt("S_ID");
String c = rs.getString("COURSE");
String y = rs.getString("YEAR");
System.out.println(i + " " + s + " " + c + " " + y);
}
```

### Updating Tables

Suppose that after a period of time we want update the YEAR column in the table STUDENT. The SQL statement to update one row might look like this:

```
String updateString = "UPDATE STUDENT " +
"SET YEAR = IV WHERE S-NAME LIKE 'yStudent'";
```

Using the Statement object stmt , this JDBC code executes the SQL statement contained in updateString :

```
stmt.executeUpdate(updateString);
```

### Using try and catch Blocks:

Something else all the sample applications include is try and catch blocks. These are the Java programming language's mechanism for handling exceptions. Java requires that when a method throws an exception, there be some mechanism to handle it. Generally a catch block will catch the exception and specify what happens (which you may choose to be nothing). In the sample code, we use two try blocks and two catch blocks. The first try block contains the method Class.forName, from the java.lang package. This method throws a ClassNotFoundException, so the catch block immediately following it deals with that exception. The second try block contains JDBC methods, which all throw SQLExceptions, so one catch block at the end of the application can handle all of the rest of the exceptions that might be thrown because they will all be SQLException objects.

### Retrieving Exceptions

JDBC lets you see the warnings and exceptions generated by your DBMS and by the Java compiler. To see exceptions, you can have a catch block print them out. For example, the following two catch blocks from the sample code print out a message explaining the exception:

Try

```
{
// Code that could generate an exception goes here.
// If an exception is generated, the catch block below
// will print out information about it.
} catch(SQLException ex)
{
System.err.println("SQLException: " + ex.getMessage());
}
```

## JavaBeans

### JavaBeans:

JavaBeans is architecture for both using and building components in Java. This architecture supports the features of software reuse, component models, and object orientation. One of the most important features of JavaBeans is that it does not alter the existing Java language.

Although Beans are intended to work in a visual application development tool, they don't necessarily have a visual representation at run-time (although many will). What this does mean is that Beans must allow their property values to be changed through some type of visual interface, and their methods and events should be exposed so that the development tool can write code capable of manipulating the component when the application is executed.

**Bean Development Kit (BDK)** is a tool for testing whether your JavaBeans meets the JavaBean specification.

### Features of JavaBeans

**Compact and Easy:** JavaBeans components are simple to create and easy to use. This is an important goal of the JavaBeans architecture. It doesn't take very much to write a simple Bean, and such a Bean is lightweight, it doesn't have to carry around a lot of inherited baggage just to support the Beans environment.

**Portable:** Since JavaBeans components are built purely in Java, they are fully portable to any platform that supports the Java run-time environment. All platform specifics, as well as support for JavaBeans, are implemented by the Java virtual machine.

**Introspection:** Introspection is the process of exposing the properties, methods, and events that a JavaBean component supports. This process is used at run-time, as well as by a visual development tool at design-time. The default behavior of this process allows for the automatic introspection of any Bean. A low-level reflection mechanism is used to analyze the Bean's class to determine its methods. Next it applies some simple design patterns to determine the properties and events that are supported. To take advantage of reflection, you only need to follow a coding style that matches the design pattern. This is an important feature of JavaBeans. It means that you don't have to do anything more than code your methods using a simple convention. If you do, your Beans will automatically support introspection without you having to write any extra code.

**Customization:** When you are using a visual development tool to assemble components into applications, you will be presented with some sort of user interface for

customizing Bean attributes. These attributes may affect the way the Bean operates or the way it looks on the screen. The application tool you use will be able to determine the properties that a Bean supports and build a property sheet dynamically. This property sheet will contain editors for each of the properties supported by the Bean, which you can use to customize the Bean to your liking. The Beans class library comes with a number of property editors for common types such as float, boolean, and String. If you are using custom classes for properties, you will have to create custom property editors to associate with them.

**Persistence:** It is necessary that Beans support a large variety of storage mechanisms. This way, Beans can participate in the largest number of applications. The simplest way to support persistence is to take advantage of Java Object Serialization. This is an automatic mechanism for saving and restoring the state of an object. Java Object Serialization is the best way to make sure that your Beans are fully portable, because you take advantage of a standard feature supported by the core Java platform. This, however, is not always desirable. There may be cases where you want your Bean to use other file formats or mechanisms to save and restore state. In the future, JavaBeans will support an alternative externalization mechanism that will allow the Bean to have complete control of its persistence mechanism.

### BDK (Bean Development Kit):

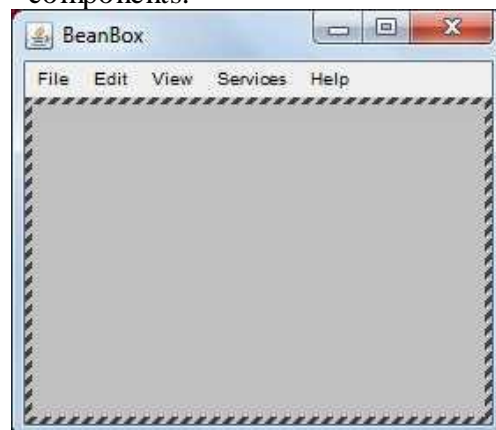
The Bean Development kit is a tool that allows the user to configure and interconnect a set of beans. The user can change the properties of a Bean, link two or more Beans and execute Beans.

Start the "beanbox" by running "\$jdk\beanbox\run.bat".

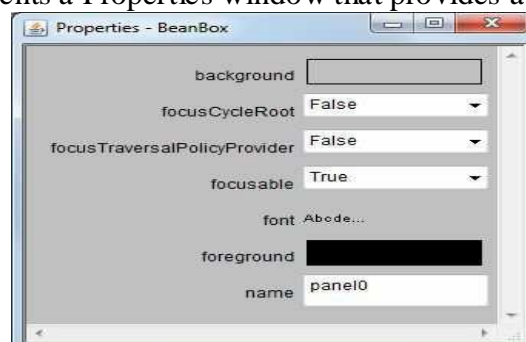
The Bean Development Kit (BDK) represents a Toolbox, a Bean Box and a Property Window. The Toolbox window that lists the demonstration Beans.



The BeanBox window that provides an area in which the user can connect the components.



The BDK is also represents a Properties window that provides an interface through which the user configure a Bean.



**Deploying java beans in jsp:**

A JavaBean can be defined as a reusable software component. A 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 set and get the values of the properties, known as getter and setter methods.

Java beans- development phases

1. The Construction Phase
2. The Build Phase
3. The Execution Phase

JavaBeans component design conventions govern the properties of the class, and the public methods that give access to the properties.

**A JavaBeans component property can be:**

Read/write, read-only, or write-only.

It means it contains a single value, or indexed, i.e. it represents an array of values.

There is no requirement that a property be implemented by an instance variable; the property must simply be accessible using public methods that conform to certain conventions:

For each readable property, the bean must have a method of the form:

```
PropertyClass getProperty () { ... }
```

For each writable property, the bean must have a method of the form:

```
setProperty (PropertyClass pc) { ... }
```

In addition to the property methods, a JavaBeans component must define a constructor that takes no parameters.

**Steps to deploy and run this JSP using JavaBean Project**

1. Write a java file and name it as **FindAuthor.java**
2. Write a jsp file and name it as **GetAuthorName.jsp**
3. Write a html file and name it as **WelcomePage.html**