

GUDLAVALLERU ENGINEERING COLLEGE

(An Autonomous Institute with Permanent Affiliation to JNTUK, Kakinada)
Seshadri Rao Knowledge Village, Gudlavalleru – 521 356.

Department of Computer Science and Engineering



HANDOUT

on

WEB TECHNOLOGIES

Vision

To be a Centre of Excellence in computer science and engineering education and training to meet the challenging needs of the industry and society.

Mission

- To impart quality education through well-designed curriculum in tune with the growing software needs of the industry.
- To serve our students by inculcating in them problem solving, leadership, teamwork skills and the value of commitment to quality, ethical behavior & respect for others.
- To foster industry-academia relationship for mutual benefit and growth.

Program Educational Objectives

- PEO1** : Identify, analyze, formulate and solve Computer Science and Engineering problems both independently and in a team environment by using the appropriate modern tools.
- PEO2** : Manage software projects with significant technical, legal, ethical, social, environmental and economic considerations.
- PEO3** : Demonstrate commitment and progress in lifelong learning, professional development, leadership and Communicate effectively with professional clients and the public.

HANDOUT ON WEB TECHNOLOGIES

Class & Sem.: III B.Tech – I Semester

Year : 2016-17

Branch : CSE

Credits : 3

1. Brief History and Scope of the Subject

- **HTML** : The founder of **HTML** was Tim Berners-Lee and his product was made attractive to the general public by Mosaic browser which was evolved at *NCSA*. It has become extremely popular and well-known in the 1990's when the Internet had been developing rapidly. During this period, HTML was broadened and presented in different modifications. The Internet strongly depends on vendors and page creators who share the joint conventions for HTML. The understanding that success of Web development is based on integration of the rules has helped the Web community to create united specifications for HTML.
- The vision of the HTML developers is that all devices must be able to reach the data on the Internet: computers with different platforms, browsers and characteristics, pocket devices, cell phones, devices for speech, and many others.
- **JavaScript** (sometimes abbreviated **JS**) is a prototype-based scripting language that is dynamic, weakly typed and has first-class functions. It is a multi-paradigm language, supporting object-oriented, imperative, and functional programming styles.
- **A servlet** is a Java programming language class used to extend the capabilities of servers that host applications access via a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by Web servers. Thus, it can be thought of as a Java Applet that runs on a server instead of a browser.
- **Java Server Pages (JSP)** is a technology that helps software developers create dynamically generated web pages based on HTML, XML, or other document types. Released in 1999 by Sun Microsystems^[1], JSP is similar to PHP, but it uses the Java programming language.
- To deploy and run Java Server Pages, a compatible web server with a servlet container, such as Apache Tomcat or Jetty, is required. The JSP technology is an *open*, freely available specification developed by Sun Microsystems as an alternative to Microsoft's Active Server Pages (ASP) technology, and a key component of the Java 2 Enterprise Edition (J2EE) specification. Many of the

commercially available application servers (such as BEA WebLogic, IBM WebSphere, Live JRun, Orion, and so on) already support JSP technology.

2. Pre-Requisites

- Need to have an idea on Object oriented concepts and Java Programming language
- Need to have a practice on various programming language constructs

3. Course Objectives:

- To develop real time web applications.
- To get acquainted with skills for creating websites and web apps through learning various technologies like HTML, CSS, JavaScript, XML, Servlets, JSP and JDBC.

4. Course Outcomes:

CO1: develop UI for web applications using markup languages.

CO2: build dynamic web pages using Java Script.

CO3: build web pages using XML.

CO4: design and implement one or more Java servlets; test and debug the servlets; deploy the servlets.

CO5: design and implement one or more Java Server Pages; test and debug the JSPs; deploy the JSPs.

CO6: update and retrieve the data from the databases using JDBC-ODBC.

5. Program Outcomes:

Graduates of the Computer Science and Engineering Program will have

- a. apply knowledge of computing, mathematics, science and engineering fundamentals to solve complex engineering problems.
- b. formulate and analyze a problem, and define the computing requirements appropriate to its solution using basic principles of mathematics, science and computer engineering.
- c. design, implement, and evaluate a computer based system, process, component, or software to meet the desired needs.

- d. design and conduct experiments, perform analysis and interpretation of data and provide valid conclusions.
- e. use current techniques, skills, and tools necessary for computing practice.
- f. understand legal, health, security and social issues in Professional Engineering practice.
- g. understand the impact of professional engineering solutions on environmental context and the need for sustainable development.
- h. understand the professional and ethical responsibilities of an engineer.
- i. function effectively as an individual, and as a team member/ leader in accomplishing a common goal.
- j. communicate effectively, make effective presentations and write and comprehend technical reports and publications.
- k. learn and adopt new technologies, and use them effectively towards continued professional development throughout the life.
- l. An ability to understand engineering and management principles and their application to manage projects in the software industry.

6. Mapping of Course Outcomes with Program Outcomes:

	a	b	c	d	e	f	g	h	i	j	k	l
CO1			3	2	3				3		3	1
CO2	2	2	2	2	3				3		3	1
CO3			2	3	2				1		2	1
CO4	2	2	2	2	2				2		2	1
CO5	2	2	3	2	3				3		3	1
CO6			3	3	3				3		3	1
Assessment Value												

7. Prescribed Text Books

- a. Web Technologies, "Black book", Kogent Learning Solutions, Dreamtech press.
- b. Chris Bates, "Web Programming: building internet applications", WILEY Dreamtech, 2nd edition

8. Reference Text Books

- a. Uttam K Roy, "Web Technologies", Oxford.
- b. John Duckett, "Beginning Web Programming".
- c. Wang Thomson, "An Introduction to web design and Programming".
- d. Robert W Sebesta, "Programming the World Wide Web", Pearson publications, Fourth edition

9. URLs and Other E-Learning Resources

- www.w3schools.com
- www.roseindia.net
- [https://msdn.microsoft.com/en-us/library/office/aa218647\(v=office.11\).aspx](https://msdn.microsoft.com/en-us/library/office/aa218647(v=office.11).aspx)
- http://www.tutorialspoint.com/web_development_tutorials.htm
- https://en.wikipedia.org/wiki/Web_2.0

10. Digital Learning Materials:

- <http://nptel.ac.in/courses/>
- <http://www.nptelvideos.com/lecture.php?id=6231>
- <http://www.nptelvideos.com/lecture.php?id=6233>
- <http://192.168.0.49/videos/videosListing/49#> (our library IP)

11. Lecture Schedule / Lesson Plan(5+1*)

Topic	No. of Periods	
	Theory	Tutorial
UNIT- 1 HTML & CSS		
Basic HTML Tags	2	1
Working with Lists	1	
Tables	1	

Forms	2	1
Frames	2	
Images & Image Maps	1	1
Cascading style sheets: CSS rules, Selectors	1	
CSS Properties: Background, Text, Font, Link, List, Table, Position	2	
UNIT-II: JAVASCRIPT		
Introduction to Java Script, Variables	1	1
Data types, Functions	2	
Operators, Control flow statements	1	
Objects in Java Script	2	1
Event Handling	2	
DHTML with Java Script	1	
UNIT-III: XML		
Basic building blocks	1	1
Validating XML Documents using DTD and XML Schemas	2	
XML DOM	1	
XML Parsers- DOM and SAX	2	1
XSLT	1	
using CSS with XML	1	
UNIT - IV WEB SERVERS AND SERVLETS		
Tomcat web server, Introduction to Servlets	2	1
Lifecycle of a Servlet, JSDK	2	
The Servlet API, The javax.servlet Package	2	

Reading Servlet parameters	2	1
Reading Initialization parameters	1	1
The javax.servlet.http package	2	
Using Cookies-Session Tracking	1	
UNIT - V JSP		
The Problem with Servlet	1	1
The Anatomy of a JSP Page	2	
Generating Dynamic Content	1	
Using Scripting Elements, Implicit JSP Objects	2	1
Declaring Variables and Methods	1	
Passing Control and Data between Pages	1	
Sharing Session and Application Data	1	
UNIT - VI: DATABASE ACCESS		
JDBC Drivers,	2	1
Database Programming using JDBC	2	
Studying javax.sql.* package	1	1
Accessing a database from a JSP Page and a Servlet page,	2	
Introduction to struts.	1	
Total No of Periods:	58	14

12. Seminar Topics

- Cascading Style Sheets
- XML Parsers
- Servlet API
- JDBC Drivers

UNIT - I

Objective:

- To develop real time web applications.

Syllabus:

HTML & CSS

HTML - Basic HTML Tags, Working with Lists, Tables, Forms, Frames, Images and Image maps.

Cascading Style sheets- CSS rules, Selectors, Types of CSS, CSS Properties for Styling Backgrounds, Text, Fonts, Links, Lists, Tables and Positioning.

Learning Outcomes:

Students will be able to

- use HTML tags and tag attributes to control a Web page's appearance and how to place Lists in HTML pages
- Create HTML Tables and Frames that can be used as navigational aids on a Web site.
- Design basic HTML forms that can used to collect data from users.
- Design Static Web Pages by combining various basic HTML elements.
- Use CSS to control the presentation style of a Web page using different types of style sheets.
- Apply simple formatting such as text, font, and background styles to a web page using various CSS properties.

LEARNING MATERIAL

➤ INTRODUCTION TO HTML:

- HTML stands for **H**yper **T**ext **M**arkup **L**anguage.
- HTML is a method of describing the format of web documents
- It is used to display the document in the web browsers.
- HTML was developed by Tim Berners-Lee. HTML standards are created by a group of interested organizations called W3C (World Wide Web consortium).

HTML Tags:

- In HTML, formatting is specified by using tags.
- A tag is a format name surrounded by angle brackets.
- End tags which switch a format off also contain a forward slash.

Points to be remembered for HTML tags:

- They are not case sensitive i.e., <head>, <HEAD> and <Head> is equivalent.
- If a browser does not understand a tag it will usually ignore it.
- White spaces, tabs and newlines are ignored by the browser.

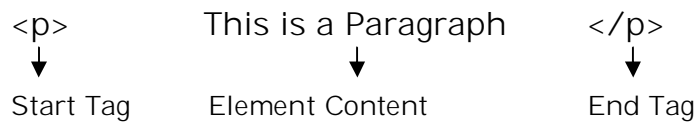
STRUCTURE OF A HTML DOCUMENT:

- HTML document consists of 2 sections.
 1. Head Section
 2. Body Section
- The basic document is shown below.

```
<html>
  <head>
    <!-- Head Section -->
  </head>
  <body>
    <!-- Body Section -->
  </body>
</html>
```

HTML ELEMENTS:

An HTML element is everything from the start tag to the end tag



HTML ATTRIBUTES:

HTML Elements can have Attributes. Attribute provide additional information about an element and are always specified in the start tag.

Syntax: `<tag attributename="value" > Content </tag>`

Sample.html

```
<html>
  <head>
    <title> Basic HTML document </title>
  </head>
  <body>
    <h1> Welcome to the world of Web Technologies</h1>
    <p> A sample HTML program </p>
  </body>
</html>
```

Output:



BASIC HTML TAGS:

1. `<html>` :

- The `<html>` tag tells the browser that this is an HTML document.
- The `<html>` tag represents the root of an HTML document.
- The `<html>` tag is the container for all other HTML elements.

2. <title>:

- defines a title in the browser toolbar
- provides a title for the page when it is added to favorites
- displays a title for the page in search-engine results

3. <body>:

- The <body> tag defines the document's body.
- The <body> element contains all the contents of an HTML document, such as text, hyperlinks, images, tables, lists, etc.

Attribute	Value	Meaning
background	URL	Specifies a background image for a document
bgcolor	Color	Specifies the background color of a document
text	Color	Specifies the color of the text in a document

4. <!-- --> Comment Tag:

- The comment tag is used to insert comments in the source code. Comments are not displayed in the browsers.

5. Heading Tags:

- There are 6 heading tags.
- The <h1> to <h6> tags are used to define HTML headings.
- <h1> defines the most important heading. <h6> defines the least important heading.

Attribute	Value	Meaning
Align	Left Right Center	Specifies the alignment of a heading

Example: Headings.html

```

<html>
  <head>
    <title>Heading Taje</title>
  </head>
  <body bgcolor=yellow text=blue>
    <!-- This is a Comment -->
    <h1 align="left">This is Heading 1</h1>
    <h2 align="center">This is Heading 2</h2>
    <h3 align="right">This is Heading 3</h3>
    <h4>This is Heading 4</h4>
  </body>
</html>

```

```

<h5>This is Heading 5</h5>
<h6 align="right">This is Heading 6</h6>
</body>
</html>

```

Output:



6. <p>: paragraph Tag

- Browser automatically add some space before and after each <p> element

Attribute	Value	Meaning
Align	Left, Right, Center Justify	Specifies the alignment of text within a paragraph

7. <a>: Anchor Tag

- The <a> tag defines a hyperlink, which is used to link from one page to another.
- The most important attribute of the <a> element is the href attribute, which indicates the link's destination.
- By default, links will appear as follows in all browsers:
 - An unvisited link is underlined and blue
 - A visited link is underlined and purple
 - An active link is underlined and red

Attribute	Value	Meaning
href	URL	Specifies the destination of the link
target	_blank _self _parent _top framename	Specifies where to open the linked document

Example: Link.html

```

<html>
<body>
  <a href="http://www.google.com" target="_self"> GOOGLE</a>
  <br>
  <a href="http://www.yahoo.com" target="_blank">YAHOO</a>

```

```

    <br>
    <a href="Headings.html" target="_parent"> My Page</a>
  </body>
</html>

```

Output:**8. :**

The tag specifies the font face, font size, and color of text.

Attribute	Value	Meaning
Color	rgb(x,x,x) #xxxxxx colorname	Specifies the color of text
Face	font_family	Specifies the font of text
Size	Number	Specifies size of text

9. <link>:

- The <link> tag defines a link between a document and an external resource.
- The <link> tag is used to link to external style sheets.

Attribute	Value	Meaning
Href	URL	Specifies the location of the linked document
Target	_blank _self _parent _top framename	Specifies where the linked document is to be loaded
Rel	Stylesheet	Specifies the relationship between current document and the linked document

10. <div>:

- The <div> tag defines a division or a section in an HTML document.
- The <div> tag is used to group block-elements to format them with CSS.

Attribute	Value	Meaning
Align	Left, Right, Center Justify	Specifies the alignment of a heading

11.
:

- The
 tag inserts a single line break.
- The
 tag is an empty tag which means that it has no end tag.

12. Various Text Formatting Tags:

The following HTML tags are used for format the appearance of the text on your web page.

(a). Headings - <h1> to <h6>

(b). Bold - or

The text in between the tags will be displayed in bold

(c). Italic - <i> </i>

Renders the text in italics i.e displays the text at a slight angle.

(d). Underline - <u> </u>

Underlines the text written in between the tags

(e). Strike out - <strike> </strike>

Defines strike through text, puts a line right through the center of the text, crossing it out.

(f). Preformatted text - <pre> </pre>

Text in <pre> element is displayed in fixed width font, and it preserves both spaces and line breaks.

(g). Typewriter Text - <tt> </tt>

The text appears to have been typed by a type writer\

(h). <big> </big> - Defines bigger text

(i). <small> </small> - Defines smaller text

(j). - Defines a subscript text. Subscript that appears half a character below the baseline.

(k). - Defines a superscript text. Superscript that appears half a character above the baseline.

(l). <center></center> - It align the text to the center of the page

Example: TextFormattingTags.html

```

<html>
  <body>
    <h1>This is Heading 1</h1>
    <b>This text is in bold</b><br>
    <i>This text is in Italics</i><br>
    <u>This text is in Underlined</u><br>
    <del>This text is Striked</del><br>
    <em>This text is Emphasized</em><br>
    <tt>This text is Type Writer Text</tt><br>
    <big>This text is Bigger</big><br>
    <small>This text is Smaller</small><br>
    H<sub>2</sub>O<br>
    (a+b)<sup>2</sup>=a<sup>2</sup>+2ab+b<sup>2</sup><br>
    <center>This Text is aligned to Center</center><br>
  </body>
</html>

```

Output:**13. <marquee>:**

It is used for Scrolling images and text in the web page

Attribute	Value	Meaning
behavior	Scroll, slide alternate	Defines the type of scrolling.
bgcolor	rgb(x,x,x) #xxxxxx colorname	<i>Deprecated</i> -Defines the direction of scrolling the content.
direction	Up, down, left, right	Defines the direction of scrolling the content.

loop	Number	Specifies how many times to loop. The default value is INFINITE, which means that the marquee loops endlessly.
scrollDelay	Seconds	Defines how long to delay between each jump.
scrollAmount	number	Defines how far to jump.

➤ WORKING WITH LISTS:

- Lists are used to collect a group of items.
- There are 3 types of Lists in HTML
 1. Ordered List
 2. Unordered List
 3. Definition List

1. ORDERED LIST:

- These are those in which the items are arranged in some specific order.
- This list can be numerical or alphabetic.
- ** tag:** The tag defines an ordered list.

Attributes :

Name	Value	Meaning
type	1 A a i	Specifies the kind of marker to use in the list
start	number	Specifies the start value of an ordered list
reversed	reversed	Specifies that the list order should be descending

- ** tag:** defines a list item.
- **Example:**

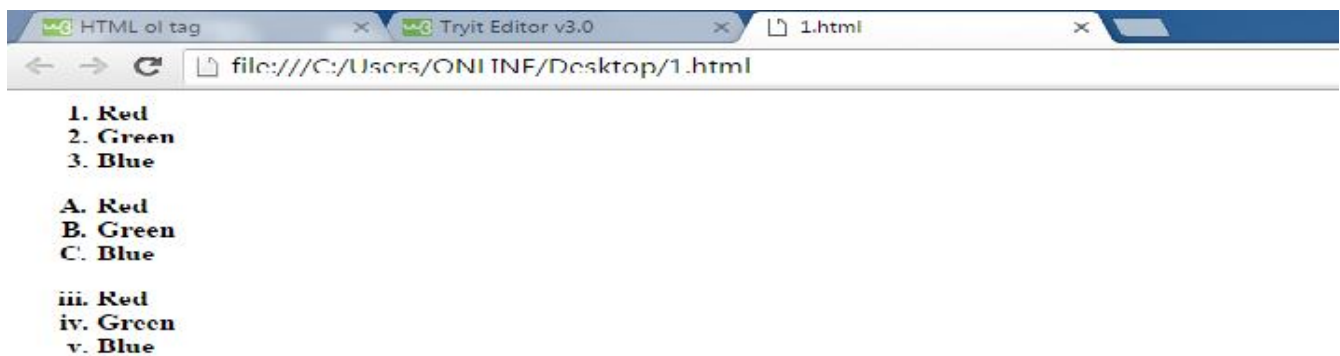
```
<html>
<body>
<ol>
  <li>Red</li>
  <li>Green</li>
  <li>Blue</li>
</ol>
```

```

<ol type="A">
  <li>Red</li>
  <li>Green</li>
  <li>Blue</li>
</ol>
<ol start=3 type="i">
  <li>Red</li>
  <li>Green</li>
  <li>Blue</li>
</ol>
</body>
</html>

```

Output:



2. UNORDERED LIST:

- The Unordered lists are those in which the items are not arranged in any order.
- This defines a Bulleted List.
- ** tag:** defines an unordered (bulleted) list.

Attributes:

Name	Value	Meaning
Type	Disc Square Circle	Specifies the kind of marker to use in the list

- ** tag:** defines a list item.
- **Example:**

```

<html>
<body>
  <ul>

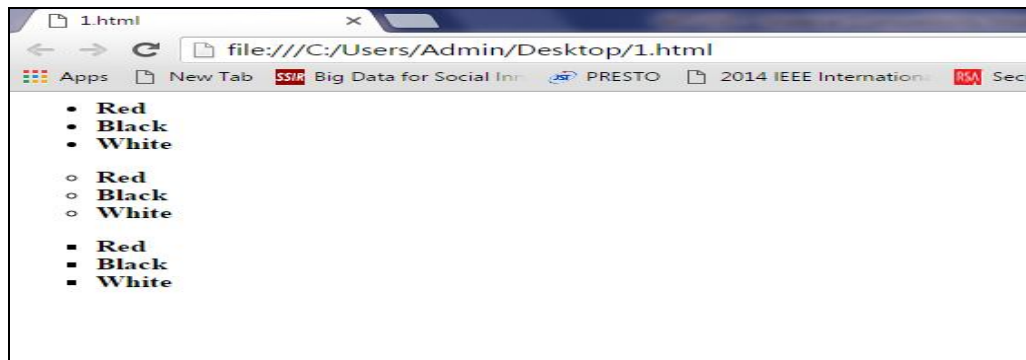
```

```

        <li>Red</li>
        <li>Black</li>
        <li>White</li>
    </ul>
    <ul type="circle">
        <li>Red</li>
        <li>Black</li>
        <li>White</li>
    </ul>
    <ul type="square">
        <li>Red</li>
        <li>Black</li>
        <li>White</li>
    </ul>
</body>
</html>

```

Output:



3. DEFINITION LIST

- These are lists of items that have 2 parts, a term to be defined and the definition.
- This create lists similar to a dictionary.
- **<dl> tag:** defines a definition list. It is used in conjunction with <dt> and <dd>
- **<dt> tag:** defines a term/name in a definition list.
- **<dd> tag:** used to describe a term/name in a definition list.
- **Example:**

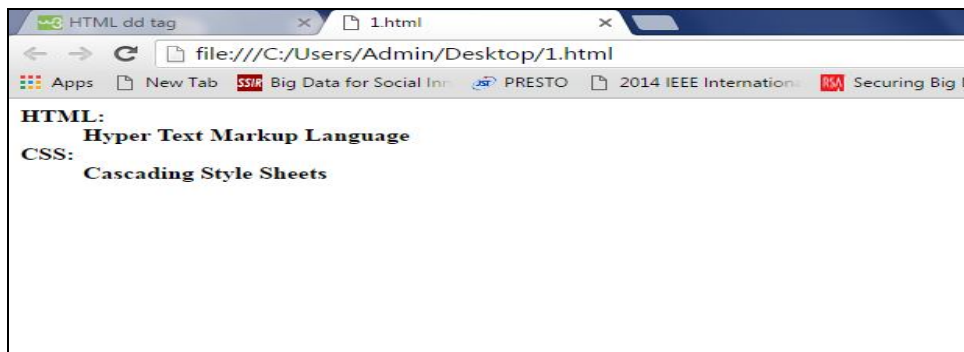
```

<html>
    <body>

```

```
<dl>
  <dt>HTML:</dt>
  <dd>Hyper Text Markup Language</dd>
  <dt>CSS:</dt>
  <dd>Cascading Style Sheets</dd>
</dl>
</body>
</html>
```

Output:



➤ TABLES:

- For Systematic arrangement of information we often require Tabular Structure.
- The biggest advantage of using tables on the web page is that the information gets arranged systematically.
- The <table> tag defines an HTML table.
- An HTML table consists of the <table> element and one or more <tr>, <th>, and <td> elements.
- The <tr> element defines a table row, the <th> element defines a table header, and the <td> element defines a table cell.
- An HTML table has two kinds of cells:
 - Header cells - contains header information (created with the <th> element)
 - Standard cells - contains data (created with the <td> element)
- The text in <th> elements are bold and centered by default.

- The text in <td> elements are regular and left-aligned by default.
- **Attributes of <table> tag:**

Name	Value	Meaning
align	Left Right center	Specifies the alignment of a table according to surrounding text
bgcolor	rgb(x,x,x) #xxxxxx colorname	Specifies the background color for a table
border	0 1	Specifies whether or not the table is being used for layout purposes
cellpadding	pixels	Specifies the space between the cell wall and the cell content
cellspacing	pixels	Specifies the space between cells
Width	Pixels %	Specifies the width of a table

- **Attributes of <tr> tag:**

Name	Value	Meaning
align	Left Right Center justify	Aligns the content in a table row
bgcolor	rgb(x,x,x) #xxxxxx colorname	Specifies a background color for a table row
valign	top middle bottom baseline	Vertical aligns the content in a table row

- **Attributes of <th> and <td> tags:**

Name	Value	Meaning
align	Left, Right Center justify	Aligns the content in a cell
bgcolor	rgb(x,x,x) #xxxxxx colorname	Aligns the content in a cell
rowspan	number	Specifies the number of rows a cell should span

colspan	number	Specifies the number of rows a cell should span
valign	Top, middle bottom baseline	Vertical aligns the content in a cell
Width	Pixels %	Specifies the width of a cell

- **Example:**

```

<html>
<body>
<table bgcolor="yellow" border="1" cellspacing="0" cellpadding="10"
bordercolor="green" align="center">
<tr>
<th rowspan="2">Header1</th>
<th colspan="3">Header2</th>

</tr>
<tr>
<td>r1,c1</td>
<td>r1,c2</td>
<td>r1,c3</td>

</tr>
<tr>
<td colspan="2">r2,c1</td>
<td>r2,c2</td>
<td>r2,c3</td>

</tr>
<tr>
<td>r3,c1</td>
<td>r3,c2</td>
<td colspan="2">
<table border="1" bgcolor="cyan" cellspacing="0"
bordercolor="red">
<tr>
<th colspan="2">Nested Table</th>

</tr>
<tr>
<td>One</td>
<td>Two</td>

</tr>
<tr>
<td>Three</td>
<td>Four</td>

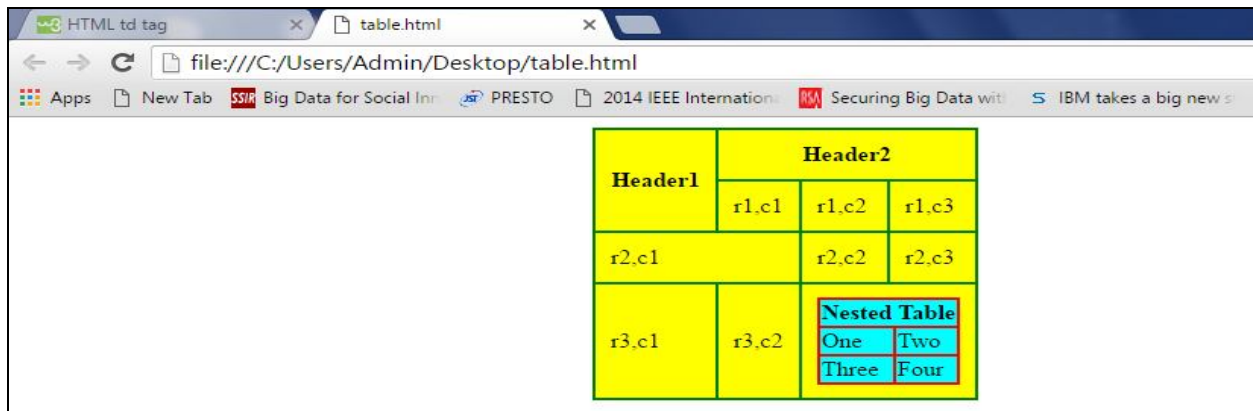
</tr>

```

```

        </table>
    </td>
</tr>
</table>
</body>
</html>
Output:

```



➤ FORMS:

- Form is a typical layout on the web page by which a user can interact with the web page.
- The `<form>` tag is used to create an HTML form for user input.
- The `<form>` element can contain one or more of the following form elements:
 - `<input>` `<textarea>` `<select>` `<option>` `<label>`
- **Attributes of `<form>` tag:**

Name	Value	Meaning
action	<i>URL</i>	Specifies where to send the form-data when a form is submitted
method	get post	Specifies the HTTP method to use when sending form-data
name	<i>text</i>	Specifies the name of a form
target	_blank _self _parent _top	Specifies where to display the response that is received after submitting the form

- **`<input>`:**
 - The `<input>` tag specifies an input field where the user can enter data.
 - `<input>` elements are used within a `<form>` element to declare input controls that allow users to input data.

- An input field can vary in many ways, depending on the type attribute.
- **Attributes of <input > tag:**

Name	Value	Meaning
type	Button, checkbox date, file, hidden image, month, number password, radio, reset submit, text	Specifies the type <input> element to display
name	text	Specifies the name of an <input> element
checked	checked	Specifies that an <input> element should be pre-selected when the page loads (for type="checkbox" or type="radio")
value	<i>Text</i>	Specifies the value of an <input> element

- **<textarea>:**
 - The <textarea> tag defines a multi-line text input control.
 - A text area can hold an unlimited number of characters, and the text renders in a fixed-width font (usually Courier).
 - The size of a text area can be specified by the cols and rows attributes
 - **Attributes of <textarea > tag:**

Name	Value	Meaning
name	text	Specifies a name for a text area
rows	number	Specifies the visible number of lines in a text area
cols	number	Specifies the visible width of a text area

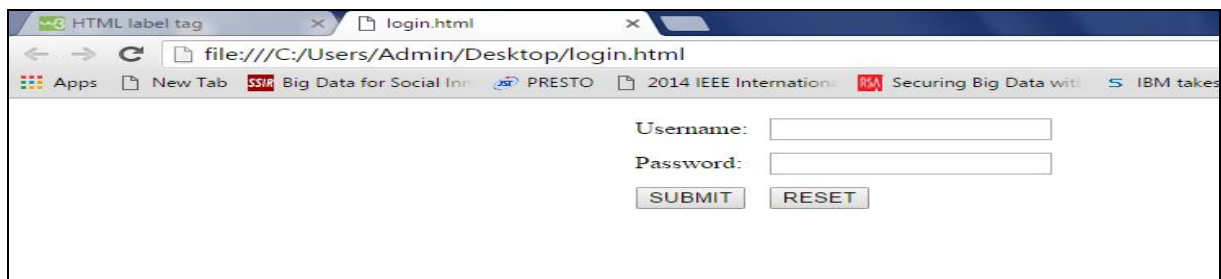
- **<select>:**
 - The <select> element is used to create a drop-down list.
 - The <option> tags inside the <select> element define the available options in the list.
 - **Attributes of <select > tag:**

Name	Value	Meaning
name	name	Defines a name for the drop-down list
multiple	multiple	Specifies that multiple options can be selected at once
size	number	Defines the number of visible options in a drop-down list

- **<label>:** The <label> tag defines a label for an <input> element.
- **Example:** Login.html

```
<html>
<body>
<form name="f1" method="post" action="">
<table align="center" cellspacing="10">
<tr>
<td><label> Username: </label> </td>
<td><input type="text" name="t1"></td>
</tr>
<tr>
<td><label> Password: </label></td>
<td><input type="password" name="t2"></td>
</tr>
<tr>
<td><input type="submit" value="SUBMIT" ></td>
<td><input type="reset" value="RESET" ></td>
</form>
</body>
</html>
```

Output:



➤ **FRAMES:**

- HTML Frames divide a browser window into several pieces or panes, each pane containing a separate HTML page.
- Each portion is called as a **Frame**.
- A Collection of Frames in the browser window is known as a **Frameset**.
- HTML Frames allow authors to present documents in multiple views, which may be independent windows or sub windows.

- One of the Key advantages that frames offer is that you can load and reload single frames without having to reload the entire contents of the browser window.
- **<frameset>:**
 - The <frameset> tag defines a frameset.
 - The <frameset> element holds one or more <frame> elements. Each <frame> element can hold a separate document.
 - The <frameset> element specifies how many columns or rows there will be in the frameset, and how much percentage/pixels of space will occupy each of them.
 - **Attributes of <frameset> tag:**

Name	Value	Meaning
cols	Pixels, % *	Specifies the number and size of columns in a frameset
rows	Pixels, % *	Specifies the number and size of rows in a frameset

- **<frame>:**
 - The <frame> tag defines one particular window (frame) within a <frameset>.
 - Each <frame> in a <frameset> can have different attributes, such as border, scrolling, the ability to resize, etc.
 - **Attributes of <frame> tag:**

Name	Value	Meaning
src	URL	Specifies the URL of the document to show in a frame
frameborder	0 1	Specifies whether or not to display a border around a frame
name	<i>text</i>	Specifies the name of a frame
noresize	noresize	Specifies that a frame is not resizable
scrolling	yes no auto	Specifies whether or not to display scrollbars in a frame

- **Example:**

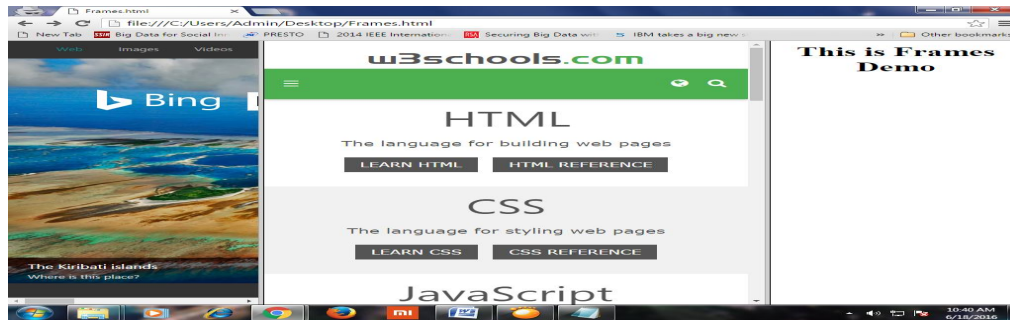
Frames.html:

```
<html>
  <frameset cols="25%,*,25%">
    <frame src="http://www.bing.com" name="f1" noresize>
    <frame src="http://www.w3schools.com" name="f2" noresize>
    <frame src="Mypage.html" name="f3" noresize>
  </frameset>
</html>
```

Mypage.html:

```
<html>
  <body>
    <h1 align="center">This is Frames Demo</h1>
  </body>
</html>
```

Output:



➤ **IMAGES:**

- Images increase the visual appearance of web pages and makes your web pages more attractive.
- The tag defines an image in an HTML page.
- The tag has two required attributes: src and alt.
- **Attributes of tag:**

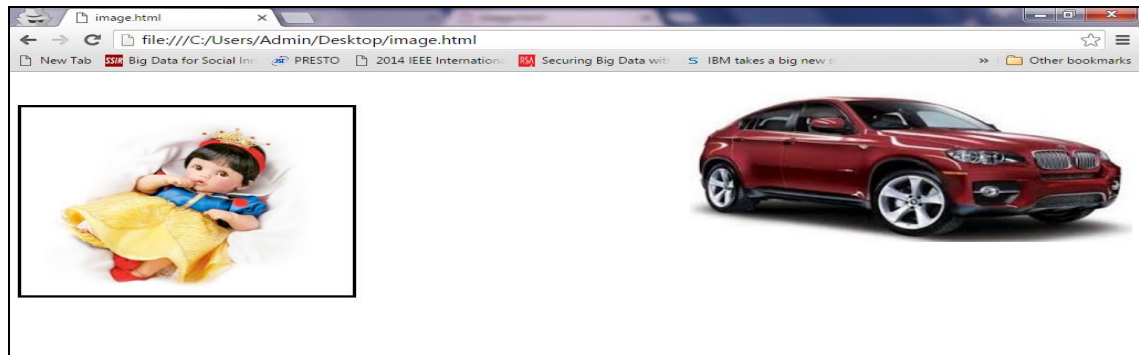
Name	Value	Meaning
src	URL	Specifies the URL of an image
align	Top, bottom middle left, right	Specifies the alignment of an image according to surrounding elements
alt	text	Specifies an alternate text for an image
border	pixels	Specifies the width of the border around an image
width	pixels	Specifies the width of an image

height	pixels	Specifies the height of an image
hspace	pixels	Specifies the whitespace on left and right side of an image
vspace	pixels	Specifies the whitespace on top and bottom of an image
ismap	ismap	Specifies an image as a server-side image-map
usemap	#mapname	Specifies an image as a client-side image-map

- **Example:**

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

Output:



➤ IMAGEMAPS:

- An Image Map is a large picture which has areas that the user can click with a mouse.
- Each clickable area on an image map is called as a "HOT SPOT". It is the area of the image that provides a link and when clicked by the user navigates to different pages.
- There are 2 types of Image maps:
 1. Server – Side Image Maps
 2. Client – Side Image Maps
- With Client - Side Image maps, the browser indicates which page you should be taken to based upon where the user clicks.

- With Server - Side Image maps, the browser sends co-ordinates on the image where the user clicked, to the server and these are processed by a script file on the server that determines which page the user should be navigated to.
- **<map>:**
 - The <map> tag is used to define a client-side image-map.
 - The required name attribute of the <map> element is associated with the 's usemap attribute and creates a relationship between the image and the map.
 - The <map> element contains a number of <area> elements that defines the clickable areas in the image map.

- **<area>:**
 - The <area> tag defines an area inside an image-map (an image-map is an image with clickable areas).
 - The <area> element is always nested inside a <map> tag.
 - **Attributes of <area> tag:**

Name	Value	Meaning
coords	coordinates	Specifies the coordinates of the area
shape	Default, rect circle, poly	Specifies the shape of the area
href	URL	Specifies the hyperlink target for the area
target	_blank _parent, _self _top, framename	Specifies where to open the target URL

- **Example:**

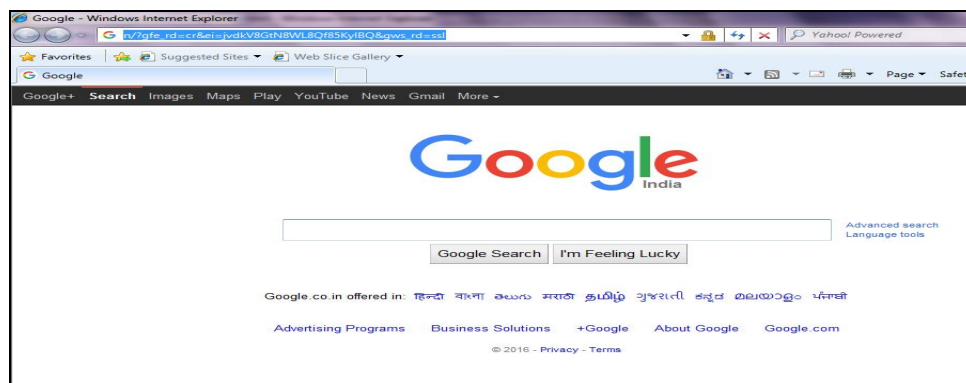
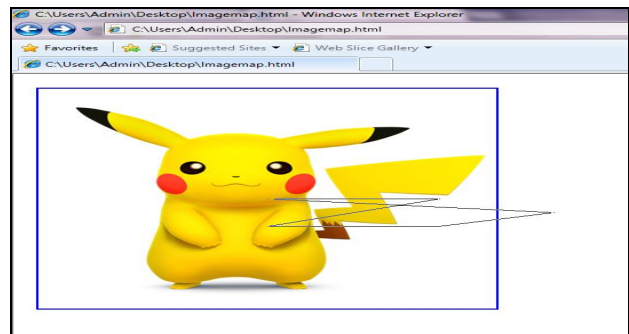
```
<html>
  <body>
    
    <map name="imagemap">
      <area shape="rect" coords="0,0,50,50"
href="http://www.google.com"
      alt="Search" target="_blank">
    <area shape="circle" coords="100,100,25"
      href="C:\\Users\\Admin\\Desktop\\1.jpg" alt="Flower">
```

```

<area
                                shape="poly"
coords="200,200,350,200,200,250,350,250,450,225"
                                href="14.jpg" alt="Car">
<area shape="default" href="http://www.yahoo.com" alt="Search"
                                target="_blank">
</map>
</body>
</html>

```

Output:



CASCADING STYLE SHEETS

➤ INTRODUCTION TO CSS:

- A Style sheet is a set of Stylistic rules that expresses the Presentation and Layout of Structured documents (Web Pages).
- Using CSS we can determine the style and layout of the web page.
- CSS is a style sheet language used to describe the presentation semantics of a document written in Markup Language.

- They allow us to specify rules for how the content of elements within your document appears.
- With CSS, all formatting could be removed from the HTML document and stored in a separate CSS file.
- **Advantages of CSS:**
 1. Improves the formatting capability of a HTML page
 2. Reduced Document size
 3. Reduced Complexity and repetition – can reuse the same style sheet with many different HTML documents.
 4. Saves time
 5. A style sheet can import and use styles from other style sheets.

➤ **CSS RULES:**

- CSS consists of set of rules that determines how the content of elements within your document should be formatted.

- **Syntax:**

Selector { property1:value ; property2:value; }

- CSS rule is made up of 2 parts:

1. Selector
2. Declaration

- **Selector :** Element/ set of elements to which declaration must be applied to

- **Declaration:**

(i). Property:	CSS Property that is to be applied
(ii). Value:	Value of CSS property

- **Example:**

```
h1
{
    font-family : arial;
    color : blue;
    text-align : center;
}
```

- **Grouping of Selectors:** Separate selector with a Comma

```
h1, h2, h3
{
    color : blue;
    font-family : calibri;
    text-align : center;
}
```

➤ 'CLASS' SELECTOR / STYLESHEET CLASS:-

- 'Class' selector allows us to define multiple styles for the same type of HTML element.

- **Syntax:**

```
selector.classname
{
    Property1 : value1; property2 : value;
}
```

- To define a style that can be used by multiple HTML elements remove tag name/selector.

- **Syntax:**

```
.classname
{
    Property1 : value1; property2 : value;
}
```

➤ THE 'id' SELECTOR:

- The #id selector styles the element with the specified id

- **Syntax:**

```
#id
{
    Property1 : value1; property2 : value;
}
```

➤ **EXAMPLE:**

```
<html>
<head>
  <style type="text/css">
    p.center
    { text-align:center; }

    p.right
    { text-align:right; }

    h2
    { text-align:center; color:orange; font-family:calibri; }
```



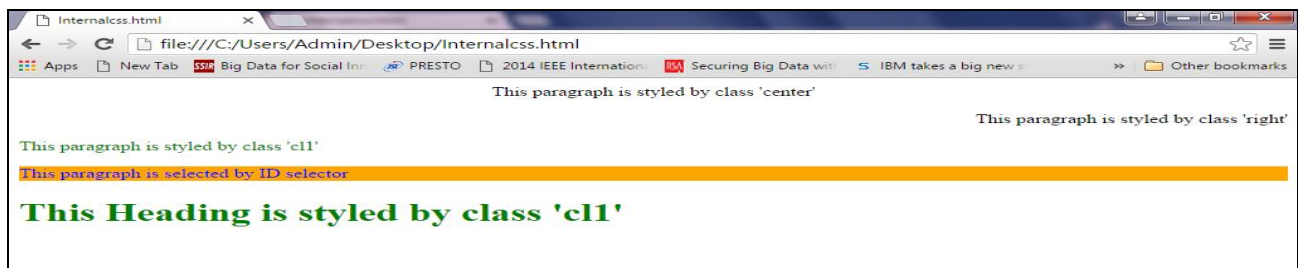
```

        .cl1
        { color:green;      }

        #id1
        { color:blue; background-color:orange; }
    </style>
</head>
<body>
    <p class="center">This paragraph is styled by class 'center'</p>
    <p class="right">This paragraph is styled by class 'right'</p>
    <p class="cl1">This paragraph is styled by class 'cl1'</p>
    <p id="id1">This paragraph is selected by ID selector</p>
    <h1 class="cl1">This Heading is styled by class 'cl1'</h1>
</body>
</html>

```

Output:



➤ TYPES OF CSS:

- When a browser reads a style sheet, it will format the HTML document according to the information in the style sheet.
- There are three ways of inserting a style sheet:
 1. Inline style sheet
 2. Internal/Embedded style sheet
 3. External style sheet

1. INLINE STYLE SHEET:

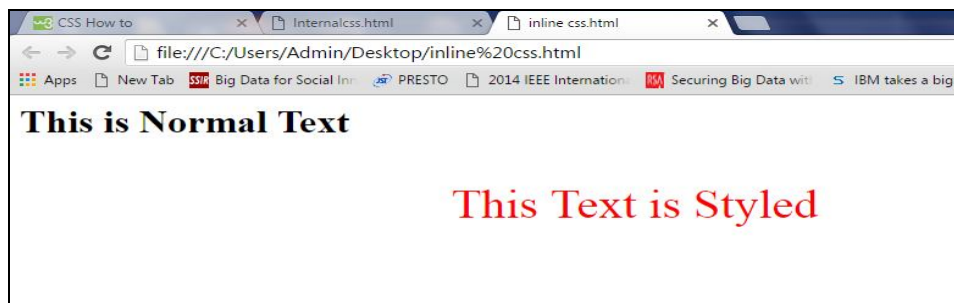
- Inline styles are placed directly inside a specific HTML element in the code.
- The style is applied at the occurrence of the HTML element by using "style" attribute in the relevant tag.
- The style attribute can contain any CSS Properties

- Inline styles cannot be reused at all

- **Example:**

```
<html>
<body>
<h1>This is Normal Text</h1>
<p style="color:red;font-size:30pt;text-align:center">This Text is Styled</p>
</body>
</html>
```

Output:



2. **INTERNAL STYLE SHEET:**

- An internal style sheet may be used if one single page has a unique style.
- Internal styles are defined within the <style> element, inside the <head> section of an HTML page.
- All the desired selectors along with the properties and values are included in the header section between <style> and </style> tags.

- **Example:**

```
<html>
<head>
<style>
    body {
        background-color:pink;
    }
    h1 {
        color: maroon;
        font-family: verdana;
    }
</style>
</head>
<body>
```

```
<h1>This is a heading</h1>
<p>This is a paragraph.</p>
</body>
</html>
```

Output:



3. EXTERNAL STYLE SHEET:

- External Style Sheets are useful when we need to apply particular style to more than one web page.
- The central idea in this type of style sheet is that the desired style is stored in an external **.css** file.
- The name of the external **.css** file has to be mentioned on our web pages. Then the styles defined in the **.css** file will be applied to all those web pages.
- **<link>** tag is used to link the external style sheet to a web page.

Example:

Mystyle.css:

```
p.left
{
    text-align:left;
    color:red;
    text-decoration:overline;
    font-family:tahoma;
    font-size:20pt;
}
p.center
{
    color:green;
    text-align:center;
```

```

text-decoration:underline;
font-family:calibri;
font-size:30pt;
}

```

Ext.html:

```

<html>
<head>
<link rel="stylesheet" href="Mystyle.css">
</head>
<body>
    <p class="left">This paragraph is styled by class 'left'</h1>
    <p class="center">This paragraph is styled by class
'center'</p>
</body>
</html>

```

Output:➤ **CSS PROPERTIES:**• **CSS BACKGROUND PROPERTIES:**

PROPERTY NAME	VALUE
background-attachment	fixed, scroll
background-color	Rgb(X,X,X), #XXXXXX, colorname
background-image	url(' url of image')
background-position	left top, left center, left bottom center top, center bottom, center center right top, right center, right bottom

Example:

```

<html>
<head>
<style type="text/css">
h1
{
background-image:url("2.gif");

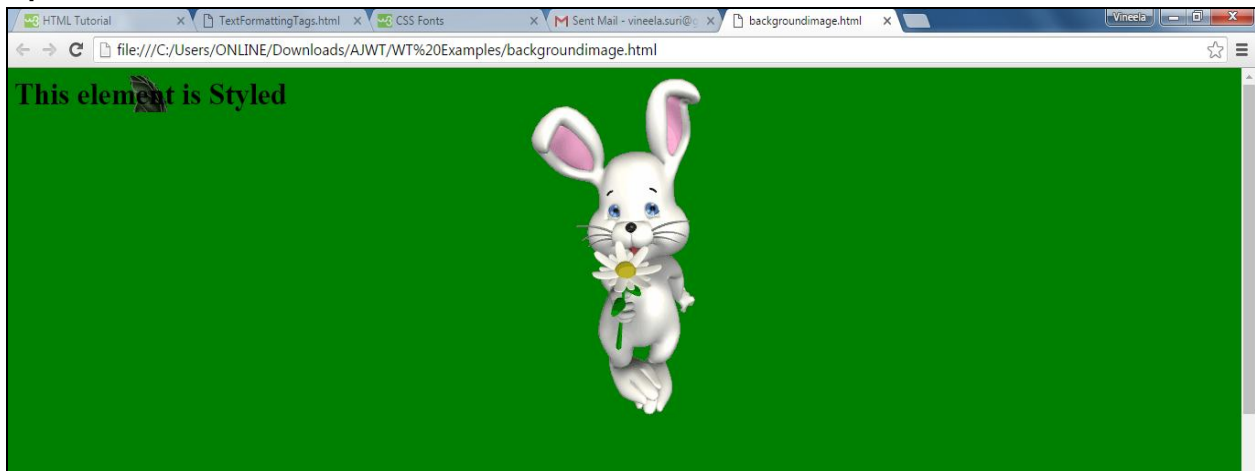
```

```

background-attachment:fixed;
background-repeat:no-repeat;
}
body
{
background-position:center top;
background-image:url('bunny giving flower.gif');
background-repeat:no-repeat;
background-attachment:fixed;
background-color:green;
}
</style>
</head>
<body>
<h1>This element is Styled</h1>
</body>
</html>

```

Output:



- **CSS TEXT PROPERTIES:**

PROPERTY NAME	VALUE
color	Color name
direction	ltr,rtl
text-align	left, right, center, justify
text-decoration	Underline, overline, Line-through, blink
text-transform	none, uppercase, lowercase, capitalize
text-indent	length, %

vertical-align	length, %, top, middle, bottom, sup, super
letter-spacing	normal, length(-ve)
word-spacing	normal, length

- **CSS FONT PROPERTIES:**

PROPERTY NAME	VALUE
font-family	Arial, Times New Roman, Etc.....
font-size	Small, smaller, medium, large, larger, length, %
font-style	normal, italic
font-variant	normal, small-caps
font-weight	normal, bold, bolder, 100-900
font-stretch	Normal, wider, narrower

Example:

TextFont.css:

```
p.right
{
    color:red;
    font-size:large;
    text-transform:capitalize;
    text-align:right;
    font-weight:200;
    letter-spacing:-3;
    word-spacing:5;
}
p.center
{
    color:blue;
    text-align:center;
    text-decoration:underline;
    text-transform:uppercase;
    font-style:italic;
    font-size:30;
}
.left
{
    color:green;
```

```
text-indent:20;
text-decoration:overline;
text-transform:lowercase;
font-family:tahoma;
font-size:small;
font-style:italic;
}

#id1
{
color:purple;
font-weight:900;
font-family:verdana;
text-decoration:line-through;
text-align:right;
font-variant:small-caps;
font-size:20;
}
```

TextFont.html:

```
<html>
<head>
<link rel="stylesheet" type="text/css" href="TextFont.css">
</head>
<body>
<p>This Paragraph is not styled</p>
<p class="left">This paragraph is styled by class left</p>
<p class="right">This paragraph is styled by class right</p>
<p class="center">This paragraph is styled by class center</p>
<p id="id1">This paragraph is styled by id</p>
</body>
</html>
```

Output:

- **CSS LINK PROPERTIES:**

The four link states are:

- a:link - a normal, unvisited link
- a:visited - a link the user has visited
- a:hover - a link when the user mouses over it
- a:active - a link the moment it is clicked

Example:**Links.html**

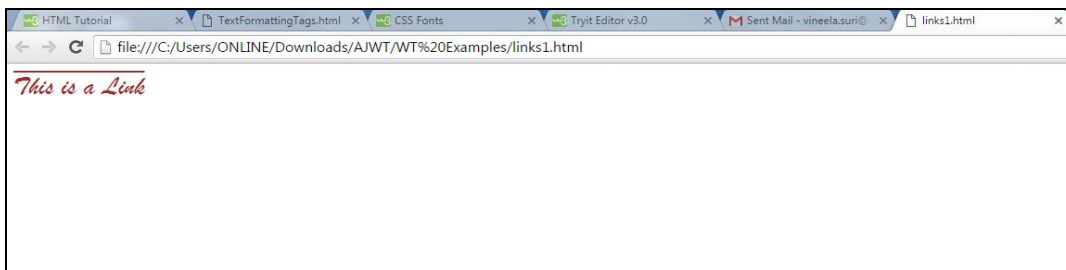
```
<html>
<head>
  <style type="text/css">
    a:link
    {
      color:red;
      font-family:tahoma;
      font-size:12pt;
    }
    a:visited
    {
      color:green;
      font-family:calibri;
      font-size:20pt;
      text-decoration:underline;
    }
    a:active
    {
      color:orange;
```



```

        font-family:cambria;
        font-size:20pt;
    }
    a:hover
    {
        color:brown;
        font-family:Brush Script MT;
        font-size:22pt;
        text-decoration:overline;
    }
</style>
</head>
<body>
    <a href="www.google.com">This is a Link</a>
</body>
</html>

```

Output:

- **CSS LIST PROPERTIES:**

PROPERTY NAME	DESCRIPTION
list-style	Sets all the properties for a list in one declaration
list-style-image	Specifies an image as the list-item marker
list-style-position	Specifies if the list-item markers should appear inside or outside the content flow
list-style-type	Specifies the type of list-item marker

Example:

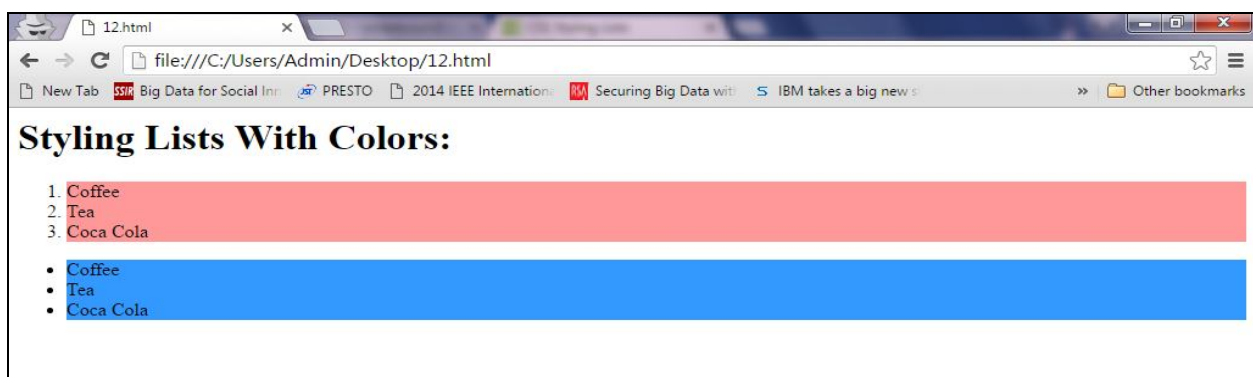
```

<html>
<head>
<style>
    ol.b {
        list-style-type: upper-roman;
        list-style-position:outside;
    }

```

```
ul.c {
    list-style-type:circle;
    list-style-position:inside;
}
ol.b
{
    background: #ff9999;
}
ul.c
{
    background: #3399ff;
}
</style>
</head>
<body>
  <h1>Styling Lists With Colors:</h1>
  <ol class="b">
    <li>Coffee</li>
    <li>Tea</li>
    <li>Coca Cola</li>
  </ol>
  <ul class="c">
    <li>Coffee</li>
    <li>Tea</li>
    <li>Coca Cola</li>
  </ul>
</body>
</html>
```

Output:



- **CSS TABLE PROPERTIES:**

PROPERTY NAME	VALUE
border-style	Dotted, dashed, double, groove, ridge, inset, outset
border-width	thin, medium, thick, pixels
border-color	Colorname, rgb(X,X,X), #XXXXXX
padding – top,padding-bottom padding – left,padding - right	pixels, %
text-align	left, right, center
vertical-align	top, bottom, middle

Example:

```

<html>
<head>
<style>
  table {
      border-collapse: collapse;
      width: 100%;
  }
  th, td {
      text-align: left;
      padding: 8px;
  }
  tr: nth-child(even){background-color: #f2f2f2}
  th {
      background-color: #4CAF50;
      color: white;
  }
</style>
</head>
<body>
  <h2>Colored Table Header</h2>
  <table>
    <tr>
      <th>Firstname</th>
      <th>Lastname</th>
      <th>Savings</th>
    </tr>
    <tr>
      <td>Indira</td>
      <td>Gamini</td>
      <td>$200</td>
    </tr>
  </table>

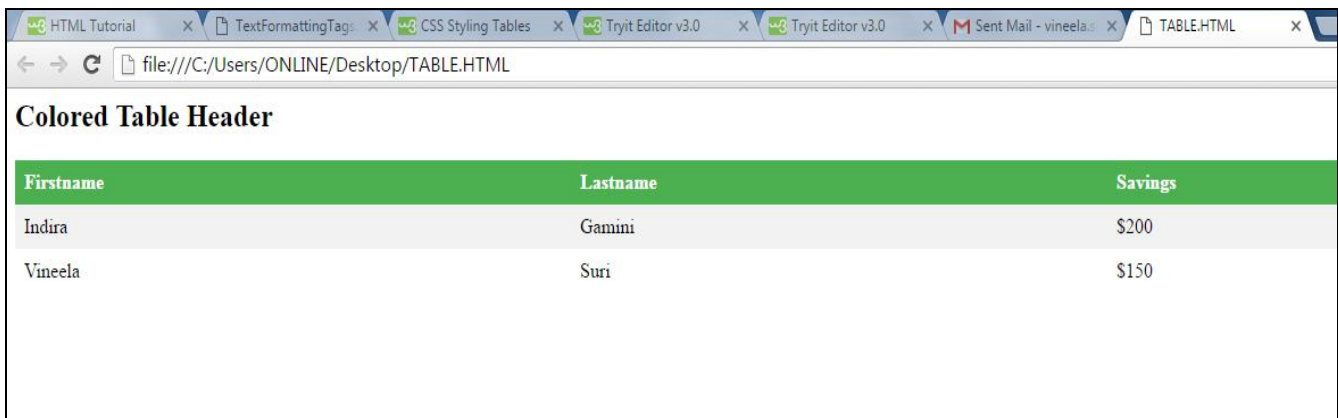
```

```

        <tr>
            <td>Vineela</td>
            <td>Suri</td>
            <td>$150</td>
        </tr>
    </table>
</body>
</html>

```

Output:



- **CSS POSITIONING ELEMENTS:**

PROPERTY NAME	VALUE
Position	static (normal flow), fixed (will not move) relative (can overlap), absolute (x & y co-ordinates)
Left, right, top, bottom	any Numeric value

Example:

```

<html>
<body>
<h1 style="position:relative;left:10;top:10;z-index:3;background-color:yellow">
    This is layer 1</h1>
<h1 style="position:relative;left:50;top:-20;z-index:2;background-color:red">
    This is layer 2</h1>
<h1 style="position:relative;left:100;top:-50;z-index:1;background-color:green">
    This is layer 3</h1>
<br><br><br>
<h1 style="position:relative;left:10;top:10;z-index:2;background-color:yellow">
    This is layer 1</h1>
<h1 style="position:relative;left:50;top:-20;z-index:3;background-color:red">

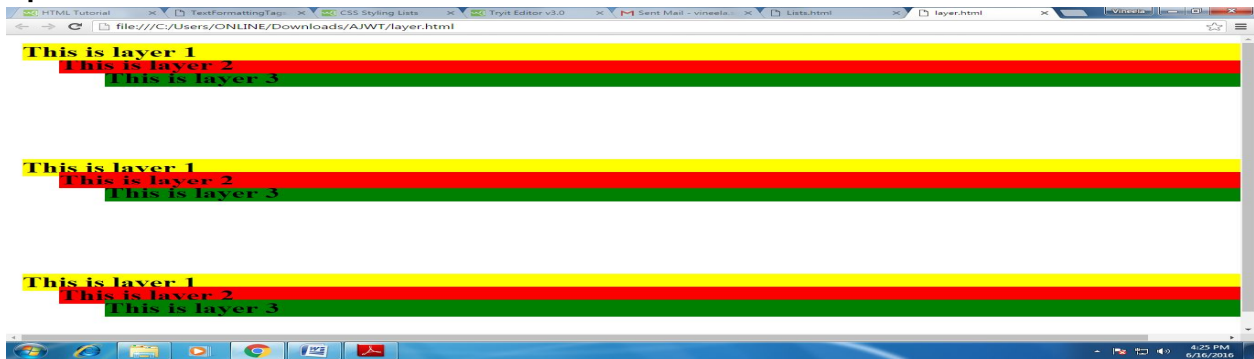
```

```
                This is layer 2</h1>
<h1 style="position:relative;left:100;top:-50;z-index:1;background-color:green">
                This is layer 3</h1>
<br><br><br>
<h1 style="position:relative;left:10;top:10;z-index:1;background-color:yellow">
                This is layer 1</h1>
<h1 style="position:relative;left:50;top:-20;z-index:2;background-color:red">
                This is layer 2</h1>

<h1 style="position:relative;left:100;top:-50;z-index:3;background-color:green">
                This is layer 3</h1>

</body>
</html>
```

Output:



Assignment-Cum-Tutorial Questions SECTION-A

1. Objective Questions

1. Latest version of HTML in use is ____ []
A).4 B).5 C). 6 D). 7
2. HTML Tags are Case Sensitive. (True / false) []
3. _____Tag is used to create Hyperlinks in HTML []
A). link B). href C). a D). src
4. The following <table> tag attribute specifies the space between Cell Wall and the Content present inside a cell. []
A). Cellspacing B). Cellpadding C).letter-spacing D).word-spacing
5. Match the following

1. src	[]	A). Specifies an image as a client-side image-map
2. href	[]	B). Specifies the URL of an Image
3. ismap	[]	C). Specifies an image as a client-side image-map
4. usemap	[]	D). Specifies the destination of a link
6. Which HTML attribute is used to define inline styles? []
A). style B). id C). Class D). styles
7. _____ tag is used to attach External CSS file to a Web Page []
A). a B). link C). href D). style
8. Which of the following property is used to control the scrolling of an image in the background? []
A). background-attachment B). background
C). background-repeat D). background-position
9. Which CSS property controls the text size? []
A). text-style B). text-size C). font-style D). font-size
10. What is the correct HTML for making a drop-down list []
A). <input type="list"> B). <list>
C). <select> D). <input type="dropdown">

11. Where in an HTML document is the correct place to refer to an external style sheet? []
- A). In the <body> section B). At the end of the document
C). In the <head> section D). At the top of the document
12. The parts of a CSS Style rule are []
- A). Selector, statements B). Tags, declarations
C). Selector, declarations D). Tags, statements
13. In CSS, declarations must be separated using _____ and Terminated using _____ []
- A). Colons, Semicolon B). Comma, semicolon
B). Semicolon, Comma D). Colon, Comma
14. How to define the link should open in new page in HTML? []
- A). Click Here
B). Click Here
C). Click Here
D). Click Here
15. Which of the following is correct HTML for inserting an image? []
- A). <image source= "admin.jpg" alt= "This is GEC" />
B).
C).
D). admin.jpg
16. How do you add a background color for all <p> elements? []
- A). all.p {background-color:#FFFFFF;}
B). p.all {background-color:#FFFFFF;}
C). p {background-color:#FFFFFF;}
D). p {backgroundcolor:#FFFFFF;}
17. Which is the correct CSS syntax? []
- A). body {color: black;} B). {body;color:black;}
C). body:color=black; D). {body:color=black;}

18. How do you display hyperlinks without an underline? []
- A). a {decoration:no underline} B). a {text-decoration: no underline}
- C). a {text-decoration: no-underline} D). a { decoration: no-underline}

SECTION-B

II. *Descriptive Questions & Problems*

- List out Common HTML tags and design a webpage using them.
- Elaborate on various types of Lists in HTML and Design the following web page using HTML lists

<p>I. Ice Creams</p> <ul style="list-style-type: none"> o Vanilla o Black Current o Strawberry <p>II. Cakes</p> <ul style="list-style-type: none"> C. Orange D. Pineapple E. Chocolate <p>III. Beverages</p> <ul style="list-style-type: none"> o Cool Drinks o Coffee: Black Hot Drink o Milk: White Cold Drink
--

- Explain <table> tag and its sub-tags with an example. Design the following table structure using HTML

HTML Table							
Cell 1	Cell 2						
	<table border="1"> <thead> <tr> <th colspan="2">Nested HTML Table</th> </tr> </thead> <tbody> <tr> <td>Cell 2.1</td> <td>Cell 2.2</td> </tr> <tr> <td>Cell 2.3</td> <td>Cell 2.4</td> </tr> </tbody> </table>	Nested HTML Table		Cell 2.1	Cell 2.2	Cell 2.3	Cell 2.4
Nested HTML Table							
Cell 2.1	Cell 2.2						
Cell 2.3	Cell 2.4						

4. Create a HTML table with columns for a country name, national sport, national flower, national animal and national tree. There must be at least 5 rows in the table.
5. Compare and contrast between Images and Imagemaps.
6. Define frame? What is the advantage of using a frame? Design the following web page using HTML Frames. Fill all the Frames with different colors



7. Create a HTML document that has two frames in one column. The top frame, which must be 20 percent of the column, must have at least four links to other documents; the bottom frame will display those documents. The links must be names of the departments in your college; the documents must be description of the departments.
8. Design the following Registration form using HTML

REGISTRATION FORM

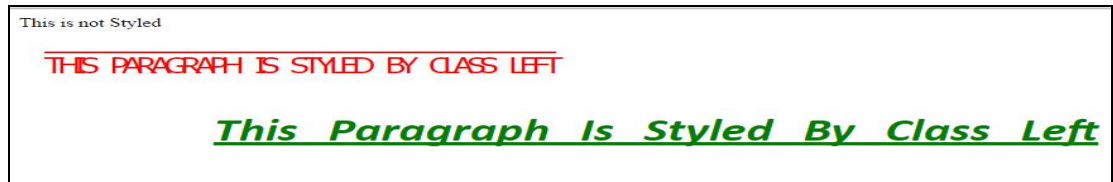
Name	<input style="width: 100%;" type="text"/>
Address	<input style="width: 100%;" type="text"/>
Zip Code	<input style="width: 100%;" type="text"/>
Country	<input style="width: 100%;" type="text" value="Please select..."/>
Gender	<input type="radio"/> Male <input type="radio"/> Female
Preferences	<input type="checkbox"/> Red <input type="checkbox"/> Green <input type="checkbox"/> Blue
Phone	<input style="width: 100%;" type="text"/>
Email	<input style="width: 100%;" type="text"/>
Password	<input style="width: 100%;" type="text"/>
Verify Password	<input style="width: 100%;" type="text"/>
	<input type="button" value="SEND"/> <input type="button" value="CLEAR"/>

9. Define CSS and explain in detail different types of cascading style sheets with examples.

10. Explain the following CSS Properties with example

- | | |
|---------------------------|----------------------|
| 1). background-attachment | 2). background-image |
| 3). text-decoration | 4). text-transform |
| 5). font-family | 6). font-weight |

11. Design the following web page using CSS Style sheet Class (Use CSS Text & Font Properties)



SECTION-C

III. Questions testing the analyzing / evaluating ability of students

1. Given below are several usages of the anchor tag in HTML. []

- I. Test Me
- II. Test Me
- III. Test Me
- IV. Test Me

Which of the above are valid?

GATE - 2004

- | | |
|------------------------|-----------------------|
| (A) I and II only | (B) I and III only |
| (C) I, II and III only | (D) I, II, III and IV |

2. A HTML form is to be designed to enable purchase of office stationery. Required items are to be selected (checked). Credit card details are to be entered and then the submit button is to be pressed. Which one of the following options would be appropriate for sending the data to the server. Assume that security is handled in a way that is transparent to the form design.

GATE - 2005

- | | |
|---------------------------|---------|
| A). Only GET | [] |
| B). Only POST | |
| C). Either of GET or POST | |
| D). Neither GET nor POST | |

3. Which of the following is an advantage of putting presentation information in a separate CSS file rather than in HTML itself? []
- A). The content becomes easy to manage. **GATE - 2015**
- B). Becomes easy to make site for different devices like mobile by making separate CSS files.
- C). CSS Files are generally cached and therefore decrease server load and network traffic.
- D). All of the above
4. To add a background color for all h1 elements, which of the following HTML syntax is used **ISRO CS - 2015**
- (A) `h1 { background-color :#FFFFFF }` []
- (B) `{ background-color :#FFFFFF } . h1`
- (C) `h1 { background-color :#FFFFFF } . h1(all)`
- (D) `h1. all{bgcolor= #FFFFFF }`
5. Create a HTML that has five frames. There must be two rows of frames the first with three frames and the other with two frames. The frames in the first row must have equal width. The left frame in the second row must be 50 percent of the width of display. Each of the frames in the top row must display a document that has a form. The left top frame must have two text boxes, each 30 characters wide, labeled Name and Address. The middle top must have five radio buttons with color name labels. The right top frame must have four check boxes, labeled with four kinds of car equipment such as a CD player and air Conditioning. The two bottom frames must have images of two different cars. The top row of frames must use 20 percent of the height of the display.

UNIT - II: JAVA SCRIPT

Objective:

To develop real time web applications.

Syllabus:

Introduction to Java Script, Variables, Data types, Functions, Operators, Control Flow statements, Objects in Java Script, Event Handling, DHTML with Java Script

Learning Outcomes:

Students will be able to

- Convert static HTML page to dynamic web page by adding JavaScript code
- Understand how to use variables, data types, operators and functions in writing simple JavaScript programs.
- Develop simple applications using various Control flow Statements.
- Understand various Objects in JavaScript and can use them in writing simple programs.
- Can dynamically change content on a web page by handling various events.
- Validate various HTML form elements using DHTML

LEARNING MATERIAL

1. INTRODUCTION TO JAVA SCRIPT:

- Web pages are two types
 - i. *Static web page*: there is no specific interaction with the client
 - ii. *Dynamic web page*: web page which is having interactions with client and as well as validations can be added.
- Script means small piece of Code.
- Scripting Language is a high-level programming language, whose programs are interpreted by another program at run time rather than compiled by the computer processor.
- BY using JavaScript we can create interactive web pages. It is designed to add interactivity to HTML pages.
- Scripting languages are of 2 types.
 - *client-side scripting languages*
 - *servers-side scripting languages*
- In general client-side scripting is used for performing simple validations at client-side; server-side scripting is used for database verifications.
- **Examples:**
 - VBScript, JavaScript and Jscript are examples for client-side scripting.
 - ASP, JSP, Servlets etc. are examples of serverside scripting.
- Simple HTML code is called static web page, if you add script to HTML page it is called dynamic page.
- Netscape Navigator developed JavaScript and Microsoft's version of JavaScript is Jscript.

Features of JavaScript:

- JavaScript is a lightweight, interpreted programming language means that scripts execute without preliminary compilation.
- It is an Object-based Scripting Language.
- Designed for creating network-centric applications.
- It is usually embedded directly into HTML Pages.
- Java script code as written between `<script>-----</script>` tags
- All Java script statements end with a semicolon
- Java script ignores white space
- Java script is case sensitive language
- Script program can be saved as either .js or .html
- Complementary to and integrated with Java.
- Open and cross-platform.

Advantages of JavaScript:

- Can put dynamic text into an HTML page
- Used to Validate form input data
- Javascript code can react to user events
- Can be used to detect the visitor's browser

Limitations of JavaScript:

- Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
- JavaScript cannot be used for networking applications because there is no such support available.
- JavaScript doesn't have any multithreading or multiprocessor capabilities.

JAVA Vs JAVASCRIPT:

<u>JAVA</u>	<u>JAVASCRIPT</u>
1. Object Oriented Programming Language	1. Object based Scripting Language
2. Platform Independent	2. Browser Dependant
3. It is both compiled and interpreted	3. It is interpreted at runtime
4. It is used to create server side applications and standalone programming	4. It is used to make the web pages more interactive
5. Java is a strongly typed language	5. JavaScript is not strongly typed(Loosely Typed)
6. Developed by sun Microsystems	6. Developed by Netscape
7. Java Programs can be standalone	7. JavaScript must be placed inside an HTML document to function

Embedding JavaScript in an HTML Page:

Embed a JavaScript in an HTML document by using <script> and </script> html tags.

Syntax:

```

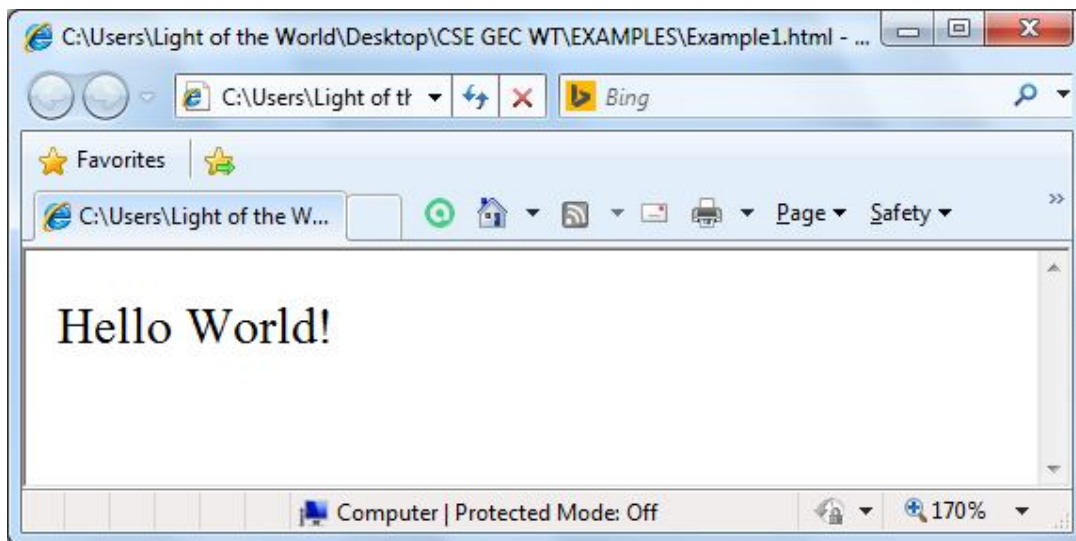
<script ...>
    JavaScript code
</script>
```

<script > tag has the following attributes.

| | |
|-----------------|---|
| Type | Refers to the MIME (Multipurpose Internet Mail Extensions) type of the script. |
| Language | This attribute specifies what scripting language you are using. Typically, its value will be javascript. Although recent versions of HTML (and XHTML, its successor) have phased out the use of this attribute. |

Example:

```
<html>
<body>
<script language="javascript" type="text/javascript">
    document.write ("Hello World!")
</script>
</body>
</html>
```

**Comments in JavaScript:**

JavaScript supports both C-style and C++-style comments. Thus:

- Any text between a // and the end of a line is treated as a comment and is ignored by JavaScript.
- Any text between the characters /* and */ is treated as a comment. This may span multiple lines.

2. VARIABLES:

- Like any programming language JavaScript has variables.
- Stores data items used in the script.
- Strict rules governing how you name your variables (Much like other languages):

Naming Conventions for Variables:

- Variable names **must** begin with a letter, digit or underscore;
- You can't use spaces in names
- Names are **case sensitive** so the variables fred, FRED and frEd all refer to **different** variables,
- It is not a good idea to name variables with similar names
- You **can't use a reserved word** as a variable name, **e.g.** var.

Creating Variables

- Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the **var** keyword as follows.

```
<script type="text/javascript">
  var name;
  var rollno;
</script>
```

- Storing a value in a variable is called **variable initialization**. You can do variable initialization at the time of variable creation or at a later point in time when you need that variable.

```
<script type="text/javascript">

  var name = "Aziz";
  var rollno=501;

</script>
```

Scope of Variables in JavaScript:

The scope of a variable is the region of your program in which it is defined and is accessible. JavaScript variables have only two scopes.

- **Global Variables:** A global variable has global scope which means it can be defined and used anywhere in your JavaScript code.
- **Local Variables:** A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

Automatically Global:

- If you assign a value to a variable that has not been declared, it will automatically become a **GLOBAL** variable.
- This code example will declare a global variable **price**, even if the value is assigned inside a function.

Example:

```
myFunction();  
// code here can use price  
function myFunction()  
{  
    price = 250; //has Global scope  
}
```

3. DATA TYPES:

- JavaScript has only four types of data
 - Numeric
 - String
 - Boolean
 - Null
- Numeric :
 - Integers such as 108 or 1120 or 2016
 - Floating point values like 23.42, -56.01 and 2E45.
 - No need to differentiate between.
 - In fact variables can change type within program.
- String:
 - A String is a Collection of character.
 - All of the following are strings:
"Computer", "Digital" , "12345.432".
 - Put quotes around the value to assign a variable:
name = "Uttam K.Roy";
- Boolean:
 - Variables can hold the values true and false.
 - Used a lot in conditional tests (later).
- Null:
 - Used when you don't yet know something.
 - A null value means one that has not yet been decided.
 - It does not mean **nil** or **zero** and **should NOT** be used in that way.

4. **FUNCTIONS:**

- A function is a group of reusable code which can be called anywhere in your program.
- This eliminates the need of writing the same code again and again.
- It helps programmers in writing modular codes. Functions allow a programmer to divide a big program into a number of small and manageable functions.
- Like any other advanced programming language, JavaScript also supports all the features necessary to write modular code using functions.
- We were using these functions again and again, but they had been written in core JavaScript only once.
- JavaScript allows us to write our own functions as well.
- **Function Definition**
 - Before we use a function, we need to define it.
 - The most common way to define a function in JavaScript is
 - By using keyword **function**, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces.

- **Syntax:**

```
<script type="text/javascript">
function functionname(parameter-list)
{
    statements
}
</script>
```

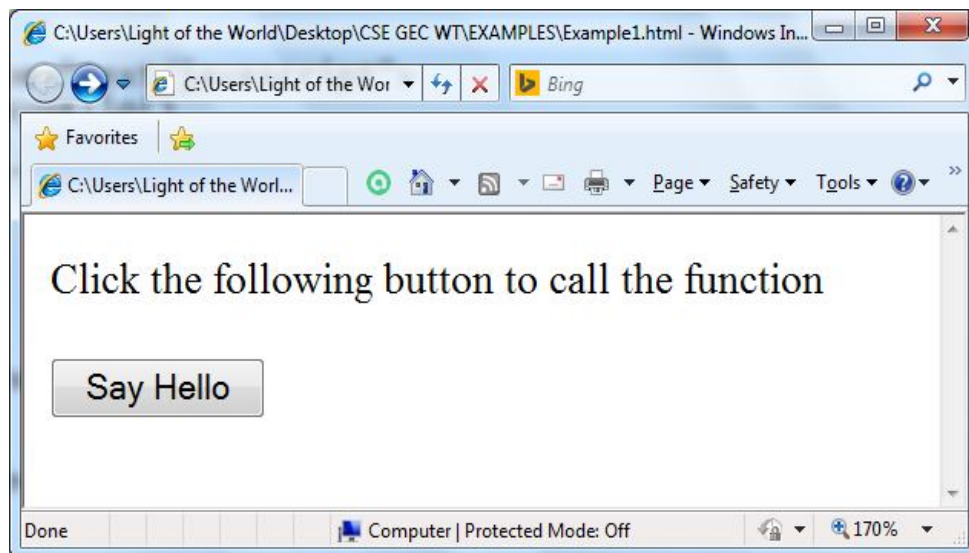
- **Example:**

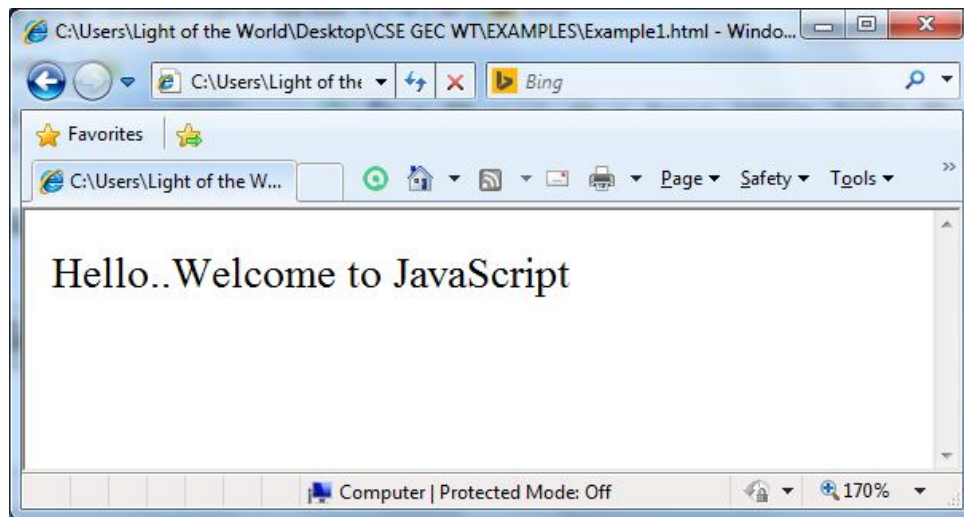
```
<script type="text/javascript">
function sayHello()
{
    alert("Hello.. How are You");
}
</script>
```

- **Calling a Function:**

To invoke a function somewhere later in the script, you would simply need to write the name of that function as shown in the following code.

```
<html>
<head>
<script type="text/javascript">
function sayHello()
{
    document.write ("Hello there!");
}
</script>
</head>
<body>
    <p>Click the following button to call the function</p>
<form>
    <input type="button" onclick="sayHello()" value="Say Hello">
</form>
<p>Use different text in write method and then try...</p>
</body>
</html>
```





5. OPERATORS:

JavaScript supports the following types of operators.

1. Arithmetic Operators
2. Assignment Operators
3. Comparison Operators
4. Logical (or Relational) Operators
5. Conditional (or ternary) Operators

1. Arithmetic Operators:

- JavaScript supports the following arithmetic operators:
- Assume variable A holds 10 and variable B holds 20, then:

Operator	Description	Example
+	Adds two numbers or joins two strings	20+10 returns 30
-	Subtracts two numbers or represents a negative number	20-10 returns 10
*	Multiplies two numbers	20*10 returns 200
/	Divides two numbers evenly and returns the quotient	20/10 returns 2
%	Divides two numbers and returns the remainder	20%10 returns 0

++	Increments the value of a number by 1 <ul style="list-style-type: none"> • Prefix (Pre-increment) • Suffix (Post-increment) 	$m = 20$ $n = ++m$ assigns 21 to n
		$m = 20$ $n = m++$ assigns 20 to n
--	Decrements the value of a number by 1 <ul style="list-style-type: none"> • Prefix (Pre-Decrement) • Suffix (Post-Decrement) 	$m = 20$ $n = --m$ assigns 19 to n
		$m = 20$ $n = m--$ assigns 20 to n

2. Assignment Operators:

Operator	Description	Example
=	Assigns the value on the right hand side to the variable on left hand side	$m=20$
+=	Adds the right hand side operand to the left hand side operand and assigns the result to the left hand side operand.	$m = 20$ $n = 10$ $m+=n$ assigns 30 to m
-=	Subtracts the right hand side operand from the left hand side operand and assigns the result to the left hand side operand.	$m = 20$ $n = 5$ $m-=n$ assigns 15 to m
=	Multiplies the right hand side operand and the left hand side operand and assigns the result to the left hand side operand.	$m = 20$ $n = 10$ $m=n$ assigns 200 to m
/=	Divides the left hand side operand by the right hand side operand and assigns the quotient to the left hand side operand.	$m = 20$ $n = 10$ $m/=n$ assigns 2 to m
%=	Divides the left hand side operand by the right hand side operand and assigns the remainder to the left hand side operand.	$m = 20$ $n = 10$ $m%=n$ assigns 0 to m

3. Comparison Operators:

Operator	Description	Example
==	Returns true if both the operands are equal otherwise returns false	20==10 returns false
!=	Returns true if both the operands are not equal otherwise returns false	20 !=10 returns true
>	Returns true if left hand side operand is greater than the right hand side operand. otherwise returns false	20 > 10 returns true
>=	Returns true if left hand side operand is greater than or equal to the right hand side operand. otherwise returns false	20 >= 10 returns true
<	Returns true if left hand side operand is less than the right hand side operand. otherwise returns false	20 < 10 returns false
<=	Returns true if left hand side operand is less than or equal to the right hand side operand. otherwise returns false	20 <= 10 returns false

4. Logical (or Relational) Operators:

Operator	Description	Example
&&	Returns true only if both the operands are true, otherwise returns false	True && True returns True
	Returns true only if either of the operands are true. It returns false when both the operands are false	True False returns True
!	Negates the operand	!true returns false

5. Conditional (or ternary) Operators:

Operator	Description	Example
?:	Returns the second operand if the first operand is true, otherwise returns the third operand.	Result=(20 > 10)? 20 : 10 Here, 20 is assigned to Result

6. CONTROL FLOW STATEMENTS:

In JavaScript we have the following conditional statements:

- Use **if** to specify a block of code to be executed, if a specified condition is true
- Use **else** to specify a block of code to be executed, if the same condition is false
- Use **else if** to specify a new condition to test, if the first condition is false
- Use **switch** to specify many alternative blocks of code to be executed

The if Statement

Syntax

```
if (condition)
{
    block of code to be executed if the condition is true
}
```

The else Statement

Use the **else** statement to specify a block of code to be executed if the condition is false.

```
if (condition)
{
    block of code to be executed if the condition is true
}
else
{
    block of code to be executed if the condition is false
}
```

The else if Statement

Use the **else if** statement to specify a new condition if the first condition is false.

Syntax:

```
if (condition1)
{
    block of code to be executed if condition1 is true
}
else if (condition2)
{
    block of code to be executed if the condition1 is false and condition2 is true
}
```

```
}  
else  
{  
    block of code to be executed if the condition1 is false and condition2 is false  
}
```

Switch Statement:

Use the switch statement to select one of many blocks of code to be executed.

Syntax:

```
switch(expression) {  
    case n:  
        code block  
        break;  
    case n:  
        code block  
        break;  
    default:  
        default code block  
}
```

This is how it works:

- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.

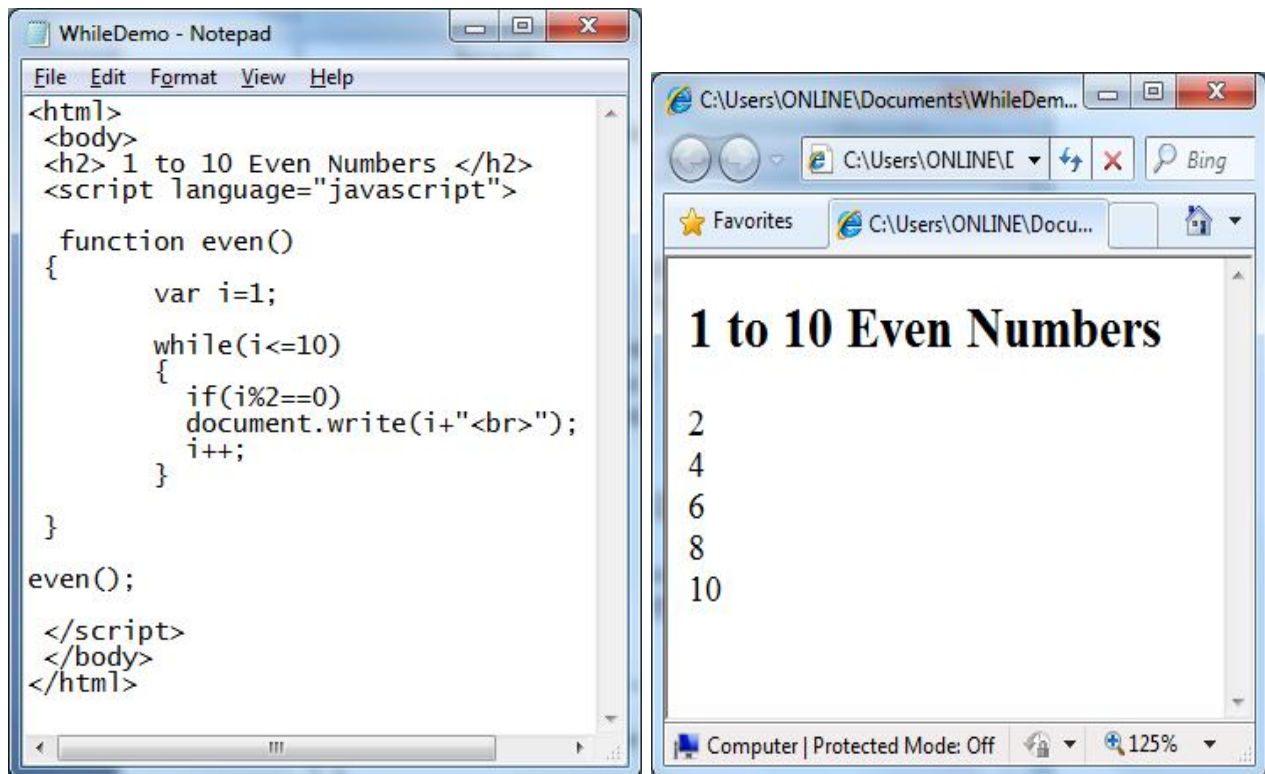
The While Loop

Syntax:

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

Example:

Write a JavaScript code to print 1 to 10 even numbers using while loop.



The Do/While Loop

The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax

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

The For Loop

The for loop has the following syntax:

```
for (statement 1; statement 2; statement 3)
{
    code block to be executed
}
```

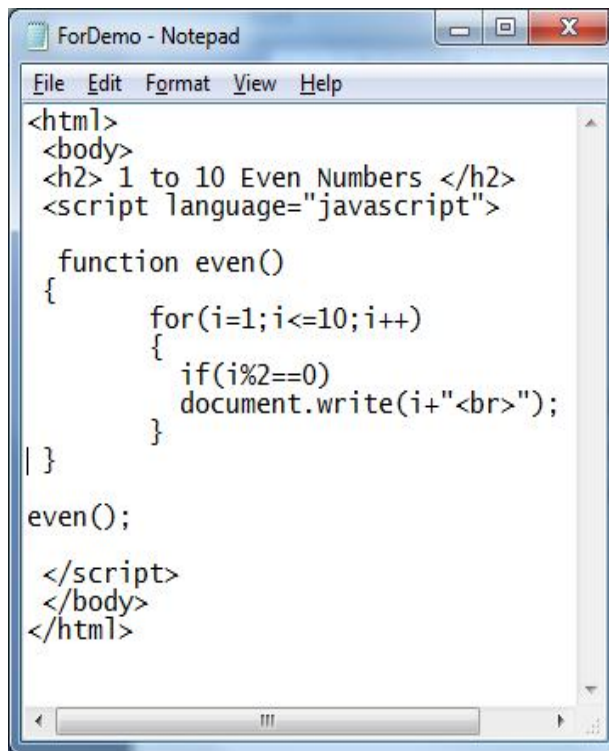
Statement 1 is executed before the loop (the code block) starts.

Statement 2 defines the condition for running the loop (the code block).

Statement 3 is executed each time after the loop (the code block) has been executed.

Example:

Write a JavaScript code to print 1 to 10 even numbers using for loop.

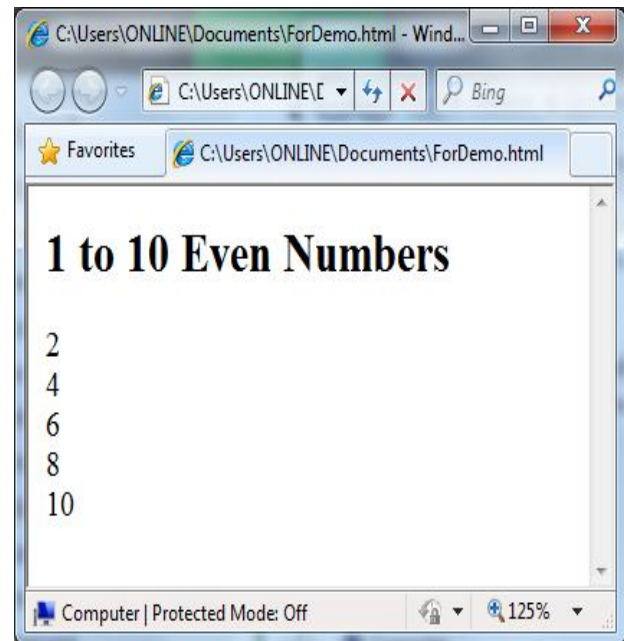


```
ForDemo - Notepad
File Edit Format View Help
<html>
<body>
<h2> 1 to 10 Even Numbers </h2>
<script language="javascript">

function even()
{
    for(i=1;i<=10;i++)
    {
        if(i%2==0)
        document.write(i+"<br>");
    }
}

even();

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



7. OBJECTS IN JAVA SCRIPT: (BUILT-IN OBJECTS)

- An Object is a thing.
- There are pre defined objects and user defined objects in Javascript.
- Each object can have properties and methods:
 - A property tells you something about an object.
 - A method performs an action
- The following are some of the Pre defined objects/Built-in Objects.
 - Document
 - Window
 - Browser/Navigator

- Form
- String
- Math
- Array
- Date

HTML DOM

The way a document content is accessed and modified is called the Document Object Model, or DOM.

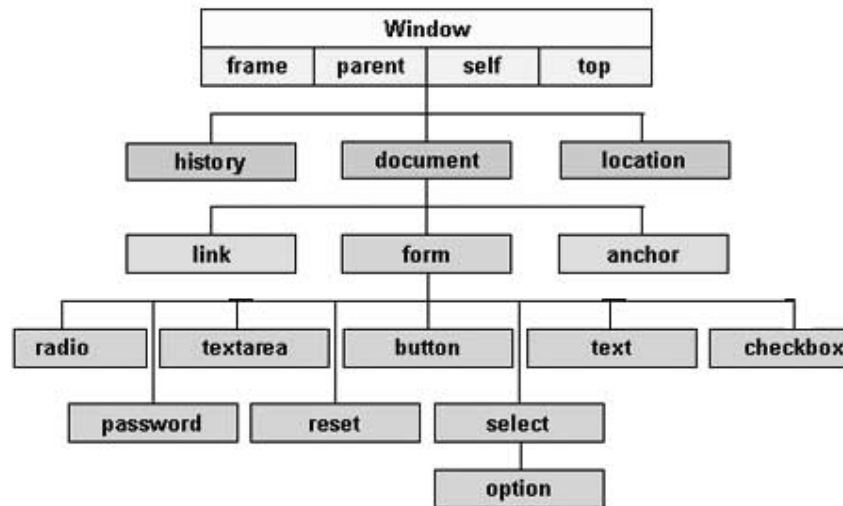
In the HTML DOM (Document Object Model), everything is a **node**:

- The document itself is a document node
- All HTML elements are element nodes
- All HTML attributes are attribute nodes
- Text inside HTML elements are text nodes
- Comments are comment nodes

The Objects are organized in a hierarchy. This hierarchical structure applies to the organization of objects in a Web document.

- **Window object** – Top of the hierarchy. It is the outmost element of the object hierarchy.
- **Document object** – Each HTML document that gets loaded into a window becomes a document object. The document contains the contents of the page.
- **Form object** – Everything enclosed in the <form>...</form> tags sets the form object.
- **Form control elements** – The form object contains all the elements defined for that object such as text fields, buttons, radio buttons, and checkboxes.

Here is a simple hierarchy of a few important objects –



THE DOCUMENT OBJECT

When an HTML document is loaded into a web browser, it becomes a **document object**.

The document object is the root node of the HTML document and the "owner" of all other nodes:

(element nodes, text nodes, attribute nodes, and comment nodes).

The document object provides properties and methods to access all node objects, from within JavaScript.

Tip: The document is a part of the Window object and can be accessed as window.document.

Properties

alinkColor-	The color of active links
bgColor-	Sets the background color of the web page. It is set in the <body> tag. The following code sets the background color to white.
title-	The name of the current document as described between the header TITLE tags.
URL-	The location of the current document.
vlinkColor-	The color of visited links as specified in the <body> tag

Methods

<code>getElementById(<i>id</i>)-</code>	Find an element by element id
<code>getElementsByTagName(<i>name</i>)-</code>	Find elements by tag name
<code>getElementsByClassName(<i>name</i>)-</code>	Find elements by class name
<code>write(text)-</code>	Write into the HTML output stream
<code>writeln(text)-</code>	Same as write() but adds a new line at the end of the output

WINDOW OBJECT:

- The **window** object is supported by all browsers. It represents the browser's window.
- All global JavaScript objects, functions, and variables automatically become members of the window object.
- Global variables are properties of the window object.
- Global functions are methods of the window object.
- Even the document object (of the HTML DOM) is a property of the window object:

```
window.document.getElementById("header");
```

is the same as:

```
document.getElementById("header");
```

Properties

- `defaultStatus` - This is the default message that is loaded into the status bar when the window loads.
- `opener` The object that caused the window to open.
- `status` - The status bar is the bar on the lower left side of the browser and is used to display temporary messages
- `length` - The number of frames that the window contains.

Methods

- `alert("message")` - The string passed to the alert function is displayed in an alert dialog box.

- open("URLname","Windowname",["options"]) - A new window is opened with the name specified by the second parameter.
- close() - This function will close the current window or the named window.
- confirm("message") The string passed to the confirm function is displayed in the confirm dialog box.
- prompt("message","defaultmessage") - A prompt dialog box is displayed with the message passed as the prompt question or phrase.

BROWSER OBJECT/NAVIGATOR OBJECT

It is used to obtain information about client browser.

Properties

- appName- Returns Browser Name
- appVersion- Returns Browser Version
- appUserAgent- It Returns User Agent
- plugins- It will display Plugins.
- mimeTypeypes – It will Return Mime type supported by browser

FORM OBJECT:

Properties

- action - The action attribute of the Top of Form element
- length - Gives the number of form controls in the form
- method- The method attribute of the Top of Form element
- name - The name attribute of the Top of Form element
- target - The target attribute of the Top of Form element

Methods

- reset()- Resets all form elements to their default values
- submit()- Submits the form

Properties of Form Elements

The following table lists the properties of form elements

- checked - Returns true when checked or false when not

- form - Returns a reference to the form in which it is part of
- length - Number of options in the <select> element.
- name - Accesses the name attribute of the element
- selectedIndex - Returns the index number of the currently selected item
- value - the value attribute of the element or content of a text input

STRING OBJECT:

String The string object allows you to deal with strings of text.

Properties

- length - The number of characters in the string.

Methods:

- charAt(index) - Returns a string containing the character at the specified location.
- indexOf(pattern) - Returns -1 if the value is not found and returns the index of the first character of the first string matching the pattern in the string.
- indexOf(pattern, index) - Returns -1 if the value is not found and returns the index of the first character of the first string matching the pattern in the string. Searching begins at the index value in the string.
- lastIndexOf(pattern) - Returns -1 if the value is not found and returns the index of the first character of the last string matching the pattern in the string.
- lastIndexOf(pattern, index) - Returns -1 if the value is not found and returns the index of the first character of the last string matching the pattern in the string. Searching begins at the index value in the string.
- split(separator) - Splits a string into substrings based on the separator character.
- substr(start, length) - Returns the string starting at the "start" index of the string Continuing for the specified length of characters unless the end of the string is found first.
- substring(start, end) - Returns the string starting at the "start" index of the string and ending at "end" index location, less one.
- toLowerCase() - Returns a copy of the string with all characters in lower case.
- toUpperCase() - Returns a copy of the string with all characters in upper case.

MATH OBJECT:

The Math object allows you to perform mathematical tasks.

Properties:

- E - Euler's constant
- LN2 - Natural log of the value 2
- LN10 - Natural log of the value 10
- LOG2E - The base 2 log of euler's constant (e).
- LOG10E - The base 10 log of euler's constant (e).
- PI - 3.1428 - The number of radians in a 360 degree circle (there is no other circle than a 360 degree circle) is 2 times PI.
- SQRT1_2 - The square root of one half.
- SQRT2 - The square root of 2.

Methods:

- abs(a) - Returns the absolute value of a.
- sin(a) - Returns the angle in radians that has a sine of the passed value.
- ceil(x) - Rounds up the value of "a" to the next integer. If the value is already a whole number, the return value will be the same as the passed value.
- cos(a) - Returns the cosine of "a" specified in radians. To convert radians to degrees, divide by 2π and multiply by 360.
- exp(a) - Returns Euler's constant to the power of the passed argument. This is the exponential power.
- floor(a) - Rounds the passed value down to the next lowest integer. If the passed value is already an integer the returned value is the same as the passed value.
- log(a) - This function is the opposite of "exp()" returning the natural log of the passed value.
- max(a,b) - Returns the larger value of a or b.
- min(a,b) - Returns the lower value of a or b.
- pow(a,b) - Takes the value of a to the power b.
- random() - Returns a random number between 0 and 1.
- round(a) - Returns the value of a to the nearest integer. If the values decimal value is .5 or greater the next highest integer value is returned otherwise the next lowest integer is returned.

- `sin(a)` - Returns the sine of "a" specified in radians. To convert radians to degrees, divide by 2π and multiply by 360.
- `sqrt(a)` - Returns the square root of a.
- `tan(a)` - Returns the tangent of a value in radians which is the value of the sine divided by the cosine.

ARRAY OBJECT:

The Array object is used to store multiple values in a single variable.

Properties:

- `length` - Sets or returns the number of elements in an array

Methods:

- `concat()` - Joins two or more arrays, and returns a copy of the joined arrays
- `indexOf()` - Search the array for an element and returns its position
- `join()` - Joins all elements of an array into a string
- `lastIndexOf()` - Search the array for an element, starting at the end, and returns its position
- `pop()` - Removes the last element of an array, and returns that element
- `push()` - Adds new elements to the end of an array, and returns the new length
- `reverse()` - Reverses the order of the elements in an array
- `shift()` - Removes the first element of an array, and returns that element
- `slice()` - Selects a part of an array, and returns the new array
- `sort()` - Sorts the elements of an array
- `splice()` - Adds/Removes elements from an array
- `toString()` - Converts an array to a string, and returns the result.

DATE OBJECT:

The Date object is used to work with dates and times

- `getDate()` - Get the day of the month. It is returned as a value between 1 and 31.
- `getDay()` - Get the day of the week as a value from 0 to 6
- `getHours()` - The value returned is 0 through 23.
- `getMinutes()` - The value returned is 0 through 59.
- `getMonth()` - Returns the month from the date object as a value from 0 through 11.
- `getSeconds()` - The value returned is 0 through 59.

- `getTime()` - The number of milliseconds since January 1, 1970.
- `getFullYear()` - Returns the numeric four digit value of the year.
- `setDate(value)` - Set the day of the month in the date object as a value from 1 to 31.
- `setHours(value)` - Set the hours in the date object with a value of 0 through 59.
- `setMinutes(value)` - Set the minutes in the date object with a value of 0 through 59.
- `setMonth(value)` - Set the month in the date object as a value of 0 through 11.
- `setSeconds(value)` - Set the seconds in the date object with a value of 0 through 59.
- `setTime(value)` - Sets time on the basis of number of milliseconds since January 1, 1970.
- `setYear(value)` - Set the year in the date instance as a 4 digit numeric value.

8. EVENT HANDLING:

JavaScript is an Event Driven System

Event:

An Event is “any change that the user makes to the state of the browser”

There are 2 types of events that can be used to trigger script:

1. Window Events
 1. Window Events, which occurs when
 - A page loads or unloads
 - Focus is being moved to or away from a window or frame
 - After a period of time has elapsed
 2. User Events, which occur when the user interacts with elements in the page using mouse or a keyboard.

Event Handlers:

Event handlers are Javascript functions which you associate with an HTML element as part of its definition in the HTML source code.

Syntax: `<element attributes eventAttribute="handler">`

Attribute	Description
Onblur	The input focus is moved from the object
Onchange	The value of a field in a form has been changes by the user by entering or deleting data

OnClick	Invoked when the user clicked on the object.
Ondblclick	Invoked when the user clicked twice on the object.
Onfocus	Input focus is given to an element
Onkeydown	Invoked when a key was pressed over an element.
Onkeypress	Invoked when a key was pressed over an element then released.
Onkeyup	Invoked when a key was released over an element.
Onload	When a page is loaded by the browser
Onmousedown	The cursor moved over the object and mouse/pointing device was pressed down.
Onmousemove	The cursor moved while hovering over an object.
Onmouseout	The cursor moved off the object
onmouseover	The cursor moved over the object (i.e. user hovers the mouse over the object).
Onmouseup	The mouse/pointing device was released after being pressed down.
Onmove	A window is moved, maximized or restored either by the user or by the script
Onresize	A window is resized by the user or by the script
onmousewheel	Invoked when the mouse wheel is being rotated.
Onreset	When a form is reset
Onselect	Invoked when some or all of the contents of an object is selected. For example, the user selected some text within a text field.
Onsubmit	User submitted a form.
Onunload	User leaves the Page

Examples:

```

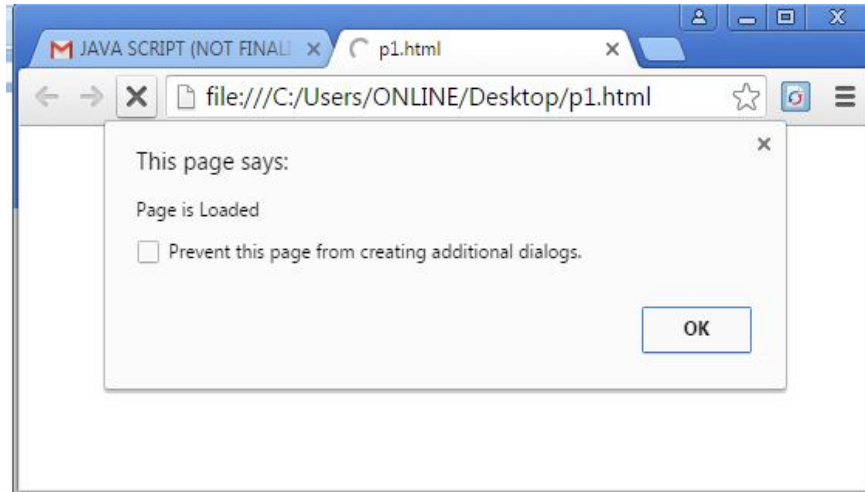
1. <html>
  <head>
    <script language="javascript">
      function fun()
      {
        alert("Page is Loaded");
      }
    </script>
  </head>
  <body onload="fun()">

```

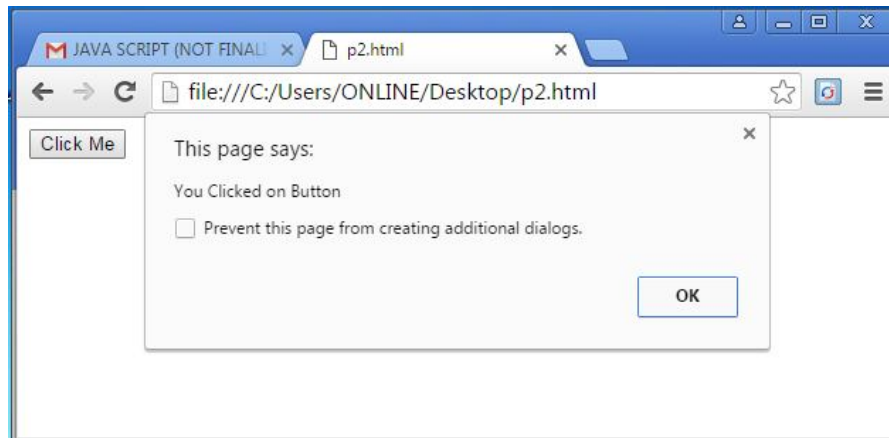
```
</body>
```

```
</html>
```

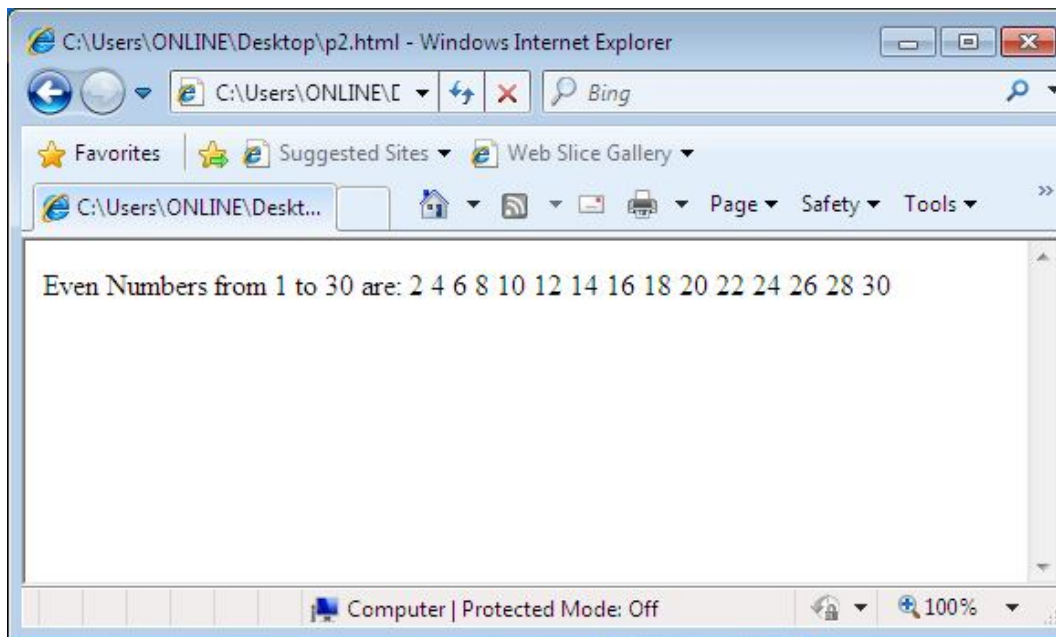
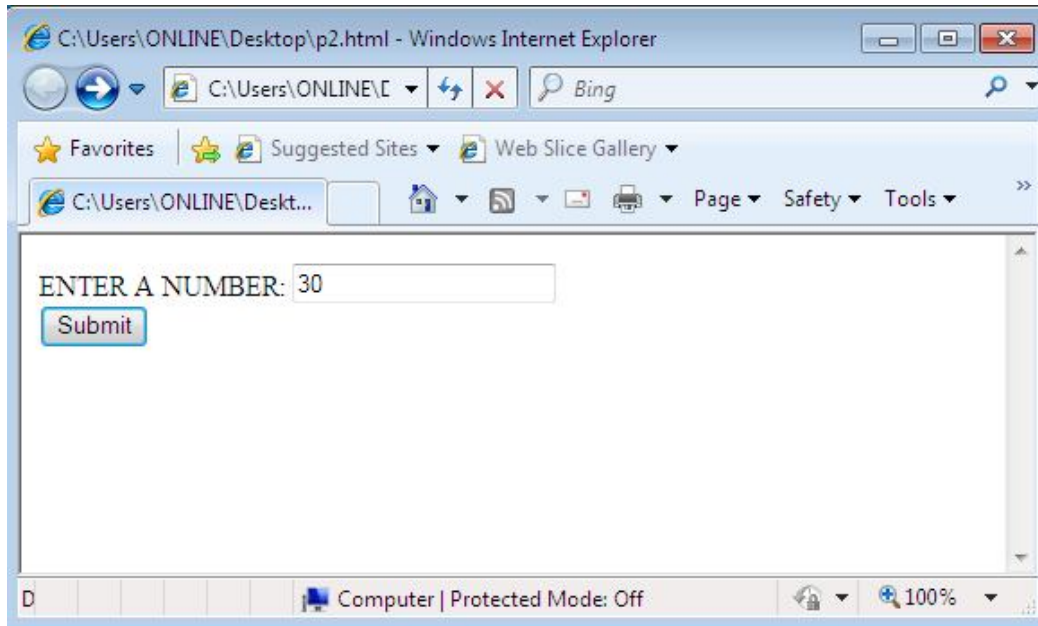
Output:



```
2. <html>
   <head>
     <script language="javascript">
       function fun()
       {
         alert("You Clicked on Button");
       }
     </script>
   </head>
   <body>
     <input type="button" value="Click Me" onClick="fun()">
   </body>
</html>
```

Output:

```
3. <html>
  <head>
  <script language="javascript">
    function fun1()
    {
      n=parseInt(f1.t1.value);
      document.writeln("Even Numbers from 1 to "+n+" are:");
      for(i=1;i<=n;i++)
      {
        if(i%2==0)
          document.write(i+" ");
      }
    }
  </script>
  </head>
  <body>
    <form name="f1" onSubmit="fun1()">
      <label>ENTER A NUMBER:</label>
      <input type="text" name="t1">
      <br>
      <input type="submit" value="Submit">
    </form>
  </body>
</html>
```

Output:**9. DHTML WITH JAVASCRIPT:**

- It refers to the technique of making web pages dynamic by client-side scripting to manipulate the document content and presentation
- Web pages can be made more lively, dynamic or interactive by DHTML techniques.

- DHTML is **not** a markup language or a software tool.
- DHTML involves the following aspects.
 - HTML - For designing static web pages
 - JAVASCRIPT - For browser scripting
 - CSS (Cascading Style Sheets) - For style and presentation control
 - DOM(Document Object Model) - An API for scripts to access and manipulate the web page as a document.

So, **DHTML = HTML + CSS + JAVASCRIPT + DOM**

- **HTML Vs DHTML**

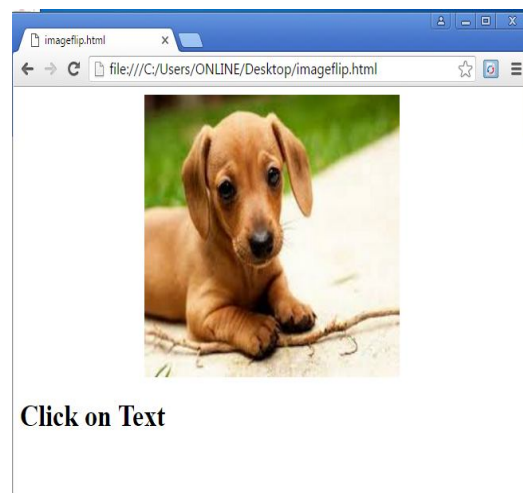
<u>HTML</u>	<u>DHTML</u>
1. It is used to create static web pages.	1. Used to create dynamic web pages.
2. Consists of simple HTML tags.	2. Made up of HTML tags+CSS+javascript+DOM
3. It is a markup language.	3. It is a technique to make web pages dynamic through client-side programming.
4. Do not allow to alter the text and graphics on the web page unless web page gets changed.	4. DHTML allows you to alter the text and graphics of the web page without changing the entire web page.
5. Creation of HTML web pages is simple.	5. Creation of DHTML web pages is complex.
6. Web pages are less interactive.	6. Web pages are more interactive.
7. HTML sites will be slow upon client-side technologies.	7. DHTML sites will be fast enough upon client-side technologies.

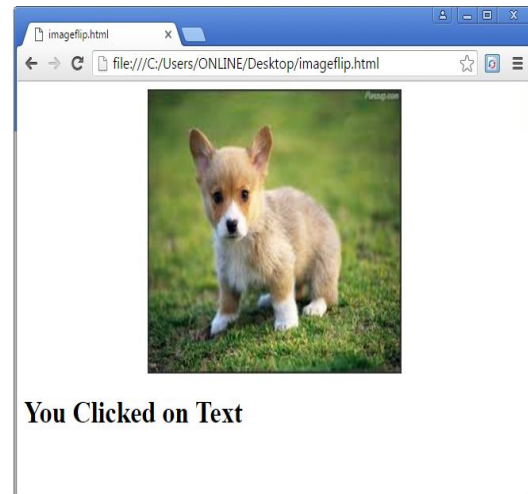
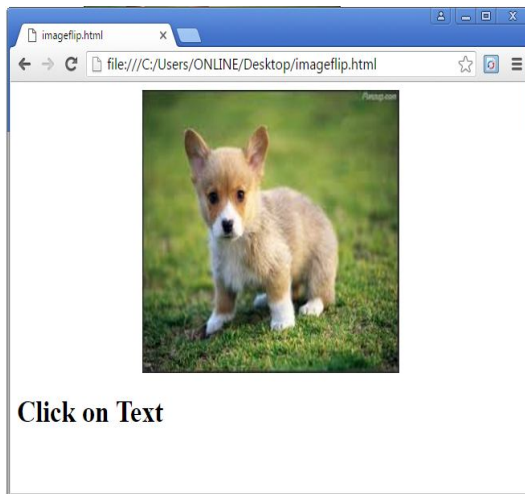
Example:

```
<html>
<head>
<script language="javascript">
    function img1()
    {
        i1.src="image2.jpg";
    }
    function img2()
    {
```

```
        i1.src="image1.jpg";
    }
    function fun1()
    {
        h11.innerText="You Clicked on Text";
    }
</script>
</head>
<body>
    <center>
        
    </center>
    <h1 id="h11" onclick="fun1()">Click on Text</h1>
</body>
</html>
```

Output:





UNIT-II
Assignment-Cum-Tutorial Questions
SECTION-A

Objective Questions

1. _____ tag is an extension to HTML that can enclose any number of JavaScript statements. []
A. <SCRIPT> B. <BODY> C. <HEAD> D. <TITLE
2. Which of the following best describes JavaScript? []
A. a low-level programming language.
B. a scripting language precompiled in the browser.
C. a compiled scripting language.
D. an object-based scripting language.
3. Is it possible to place nested functions in JavaScript? [True/False]
4. Is JavaScript a case-sensitive language? [True/False]
5. What is the output of this strict equality operator? 5 === "5" is []
A. True b.False
6. When the form elements like <button>, <input>, <select>, <textarea> loses their input focus: Which of the following event fires. []
A. onfocus B. onblur C. onclick D. ondblclick
7. We can declare all type of variables in JavaScript with the keyword _ []
A. obj B. jvar C. var D. None of these
8. Browser object is also called as _____ Object. []
9. What is the correct syntax for referring to an external script called " abc.js"?
A. <script href=" abc.js"> B. <script name=" abc.js">
C. <script src=" abc.js"> D. None of the above
10. How to create a Date object in JavaScript? []
A. dateObjectName = new Date([parameters])
B. dateObjectName.new Date([parameters])
C. dateObjectName := new Date([parameters])
D. dateObjectName Date([parameters])
11. What is the correct JavaScript syntax to write "Hello World"? []
A. System.out.println("Hello World")
B. println ("Hello World")
C. document.write("Hello World")
D. response.write("Hello World")
12. <script>
 document.write(navigator.appCodeName);
</script> []

- A. get code name of the browser of a visitor
 B. set code name of the browser of a visitor
 C. get the version of the browser
 D. None of the above
13. `<script language="javascript">` []
 `function x()`
 `{`
 `document.write(2+5+"8");`
 `}`
 `</script>`
 A. 258 B. Error C. 7 D. 78
14. `<script type="text/javascript">` []
 `var s = "9123456 or 80000?";`
 `var pattern = /\d{4}/;`
 `var output = s.match(pattern);`
 `document.write(output);`
 `</script>`
 A. 9123 B. 91234 C. 80000 D. None of the above
15. Which is the correct way to write a JavaScript array? []
 A. `var txt = new Array(1:"tim",2:"kim",3:"jim")`
 B. `var txt = new Array:1=("tim")2=("kim")3=("jim")`
 C. `var txt = new Array("tim","kim","jim")`
 D. `var txt = new Array="tim","kim","jim"`

SECTION-B

SUBJECTIVE QUESTIONS

- Define JavaScript and Describe Primitive Data Types that JavaScript uses.
- What is a function? Explain how parameters are passed to a function in JavaScript.
- Define a variable. Explain different Scoping rules associated with variables in JavaScript with example.
- Illustrate various control statements available with JavaScript.
- Write about the properties and methods of the following JavaScript Objects.
 a) Document b) Form c) Window d) Browser e) Math
- Describe an Event. Explain how events are handled in JavaScript with an Example.
- Distinguish between HTML with DHTML.
- Write a JavaScript that reads an integer and determines and displays whether it is an odd or even number?
- Create a JavaScript code to Print all numbers from 1 to 100 except multiples of 3.
- Write a JavaScript to check whether given two numbers are equal or not. If not, Display the Largest & Smallest among those two.

11. Write a JavaScript that reads an Integer and determine whether it is Prime Number or not
12. Develop a JavaScript that reads an Integer and print its factorial.
13. Write a JavaScript which reads a number given and displays the output in words (Eg:- Given 123, Output should be ONE TWO THREE)
14. Develop a JavaScript program to validate Login form consisting of username and password (use regular expressions).
15. Create a JavaScript which has event handlers for the buttons "red", "blue", "green", "yellow" and "orange" which must produce messages stating the chosen favorite color and applies it as a background color.

SECTION-C

QUESTIONS AT THE LEVEL OF GATE

1. _____ tag is an extension to HTML that can enclose any number of Javascript statements. **UGC-NET DEC 2015**

A. <SCRIPT> B.<BODY> C. <HEAD> D. <TITLE>

2. Javascript and Java has similar name because _____ is/are true.

UGC-NET DEC 2015

- (a) Javascripts syntax is loosely based on Java's syntax
(b) Javascript is stripped down version of Java
(c) Java and Javascript are originated from Island of Java

Codes:

A. Only B.(a), (b) and (c) C. (a) and (b) D. (b) and (c)

Web Technologies

UNIT – 3

SYLLABUS:

Basic building blocks, Validating XML Documents using DTD and XML Schemas, XML DOM, XML Parsers- DOM and SAX, XSLT, using CSS with XML.

Objective:

To develop real time web applications.

Learning Outcomes:

Students will be able to

- Understand the basic building blocks of XML and will be able to store Data in XML files.
- Differentiate between XML and HTML
- Define DTD and Schema for a XML file.
- Differentiate between different types of XML parsers and can use them to Validate XML documents.
- Transform XML documents into various other formats using XSLT.
- Use CSS rules with XML.

INTRODUCTION TO XML:

- XML stands for eXtensible Markup Language
- XML is a markup language much like HTML
- XML was designed to store and transport data
- XML was designed to be self-descriptive
- XML is a W3C Recommendation
- The first XML version 1.0, published in 1998.
- HTML limits you to use only fixed number of tags, where as XML allows to create new tags.
- For example, you are developing website for a college, then you have tags like <SNO>, <STUDENTNAME>, <DOB> etc.
- HTML, XML languages are derived from Standard Generalized Markup Language (SGML).
- XML and related technologies and those are: SGML, XSL, W3C, SOAP, XSLT, DOM, SAX
- SGML, Standard Generalized Markup Language is basis for all markup languages.
- XSL, eXtensible Stylesheet Language is a combination of XML and Style Sheets.
- XSLT, XSL transformations provides rules for transformations from one XML to another.
- SOAP, Simple Object Access Protocol is communication protocol for Internet to XML documents and it provides notifications for events.
- SAX, Simple API for XML, predefined application package interface.
- DOM, Document Object Model, is another XML Parser.

Uses of XML:

- Easy to organize the document
- Tags or document elements are reusable
- It simplifies data sharing

- Better environment for data transfer
- The XML document is language natural. That means a Java program can generate an XML document and this document can be parsed by Perl.
- XML files are independent of an operating system.
- It simplifies data availability

Applications of XML:

- Electronic Commerce (popularly known as E-Commerce)
- Financial Funds Transfer
- Multimedia Messages and Messaging exchange

Differences between XML and HTML:

XML	HTML
1. It is used to store the data.	1. It is used to present the content.
2. It supports user defined tags.	2. It supports only predefined tags in html.
3. XML separates content from presentation.	3. HTML specifies presentation
4. XML allows users to create new tags	4. HTML doesn't allow users to create new tags
5. You can generate new mark up languages using XML	5.No such possibility.
6. XML is case sensitive	6. HTML is not case sensitive
7. Root element is user defined and only one root element allowed.	7.Root element is <HTML>

XML Features:

- XML allows the user to define his own tags and his own document structure.
- XML document is a pure information wrapped in XML tags.
- XML is a text based language, plain text files can be used to share data.

- XML provides a software and hardware independent way of sharing data.

Elements and Attributes:

In XML 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.

- The syntax of writing any element for opening tag is <element name>
- The syntax of writing any closing element for closing tag is </element name>
- An empty tag can be defined by putting a / (forward slash) before closing bracket.
- A space or a tab character is not allowed in the element name or in attribute name.

Basic Structure / Syntax of an XML Document :

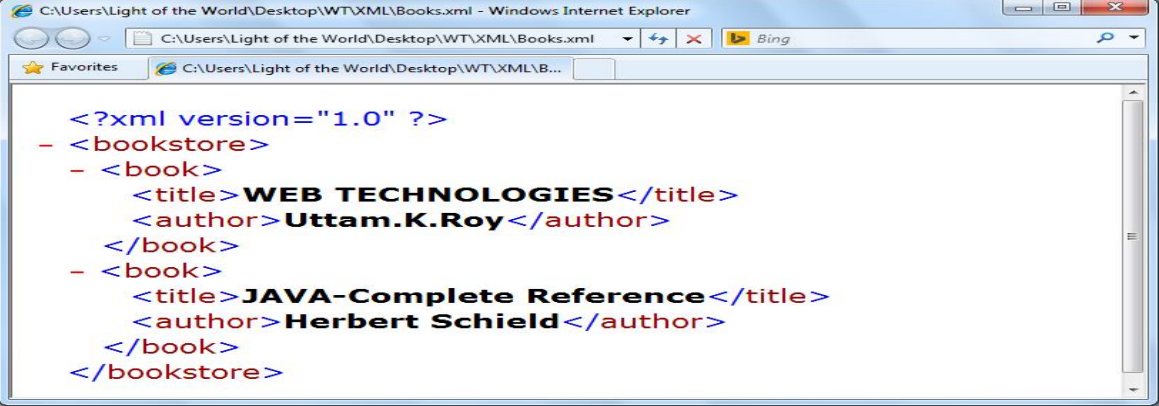
```
<?xml version="1.0"?>
<root>
  <child>
    <subchild>
      </subchild>
    </child>
  </root>
```


Example :

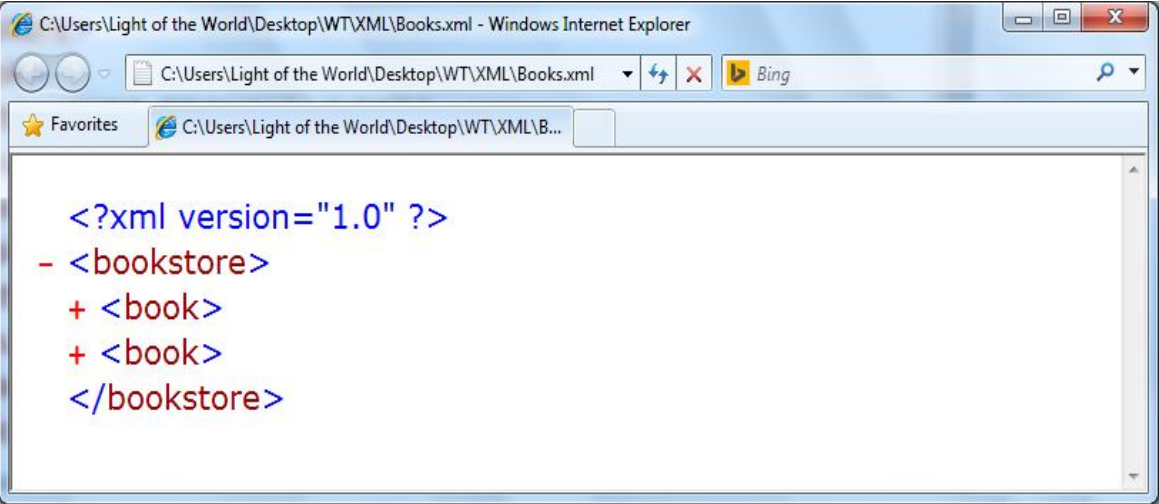
Books.xml

```
<?xml version="1.0"?>
<bookstore>
  <book>
    <title> WEB TECHNOLOGIES </title>
    <author>Uttam.K.Roy </author>
  </book>
  <book>
    <title> JAVA-Complete Reference </title>
    <author>Herbert Schildt </author>
  </book>
</bookstore>
```

If the above Books.xml is opened using any of the browsers (Example, Internet Explorer), following is the output.



The screenshot shows the XML content rendered in a tree view in Internet Explorer. The root element is <?xml version="1.0" ?>. Below it is the <bookstore> element, which contains two <book> elements. The first <book> element has a <title>WEB TECHNOLOGIES</title> and a <author>Uttam.K.Roy</author>. The second <book> element has a <title>JAVA-Complete Reference</title> and a <author>Herbert Schield</author>.



The screenshot shows the XML content rendered in a collapsed tree view in Internet Explorer. The root element is <?xml version="1.0" ?>. Below it is the <bookstore> element, which contains two <book> elements. The <book> elements are collapsed, indicated by plus signs (+).

1. Basic Building Blocks

Building block means which all element or part that make a xml document.

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

- Elements
- Attributes
- Entities
- PCDATA
- CDATA

Elements

Elements are the **main building blocks** of both XML and HTML documents. Examples of HTML elements are "body" and "table". Examples of XML elements could be "note" and "message". Elements can contain text, other elements, or be empty. Examples of empty HTML elements are "hr", "br" and "img".

Examples:

```
<body>some text</body>
<message>some text</message>
```

Attributes

Attributes provide **extra information about elements**.

Attributes are always placed inside the opening tag of an element. Attributes always come in name/value pairs. The following "img" element has additional information about a source file:

```

```

Entities

Some characters have a special meaning in XML, like the less than sign (<) that defines the start of an XML tag.

Most of you know the HTML entity: " ". This "no-breaking-space" entity is used in HTML to insert an extra space in a document. Entities are expanded when a document is parsed by an XML parser.

The following entities are predefined in XML:

Entity Reference	Character
<	<
>	>
&	&
"	"
'	'

PCDATA

PCDATA means parsed character data.

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

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

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

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

CDATA

CDATA means character data.

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

2. Validating an XML file :

An XML file can be validated using the following specifications.

1. DTD (Document type definition)
2. XML Scheme.

1. DOCUMENT TYPE DEFINITION(DTD):

- The document type definition used to define the basic building block of any 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.
- Many developers recommend writing DTDs for the XML applications.
- DTD standards are defined by the W3C.

A DTD may contain the following:

- Name of the root element
- Reference to an external DTD
- Element declaration
- Entity declaration

Occurrence indicators in DTD

Operator	Syntax	Description
None	a	Exactly one occurrence of a
*	a*	Zero or more occurrences of a
+	a+	One or more occurrences of a
?	a?	Zero or one occurrence of a

There are two ways of writing DTDs

1. Internal DTD
2. External DTD

Internal DTD

The DTD can be embedded directly in the XML document as a part of it.

The general syntax for an internal DTD is

```
<!DOCTYPE root-element [
    <!-- doctype declaration -->
]>
```

The keyword DOCTYPE specifies that a DTD is to be used by the document.

The following rules must be followed:

- The keyword DOCTYPE must be in upper case (Well formedness constraint).

- The document type declaration must appear before the first element in the document(Well formedness constraint).
- The name following the word DOCTYPE (root-element in this case) must match with the name of the root-element(Top-level element) in the xml document.

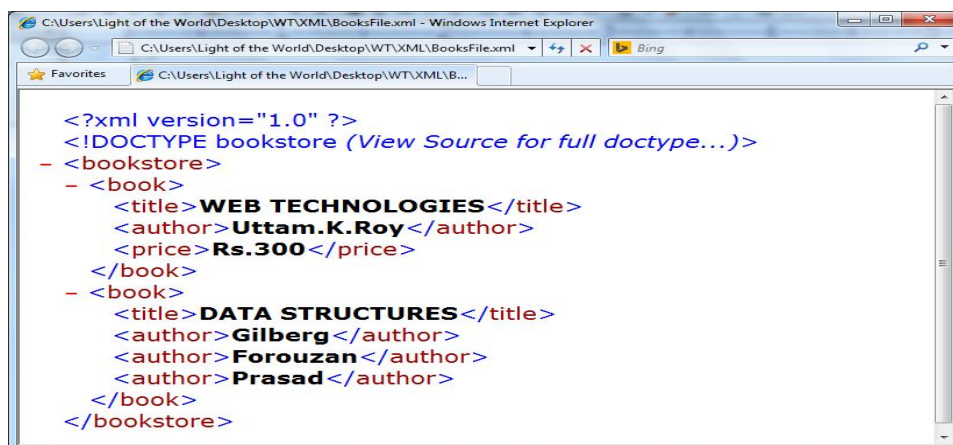
Example:

BooksFile.xml

```
<?xml version="1.0"?>
<!DOCTYPE bookstore [
  <!ELEMENT bookstore (book+)>
  <!ELEMENT book (title,auther+,price?)>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT auther (#PCDATA)>]
>
<bookstore>
  <book>
    <title>WEB TECHNOLOGIES</title>
    <author>Uttam.K.Roy </author>
    <price>Rs.300 </price>
  </book>

  <book>
    <title>DATA STRUCTURES</title>
    <author>Gilberg </author>
    <author>Forouzan</author>
    <author>Prasad</author>
  </book>
</bookstore>
```

If the above BooksFile.xml is opened using a web browser like Internet Explorer, the following is the result.



```
<?xml version="1.0" ?>
<!DOCTYPE bookstore (View Source for full doctype...)>
- <bookstore>
- <book>
  <title>WEB TECHNOLOGIES</title>
  <author>Uttam.K.Roy</author>
  <price>Rs.300</price>
</book>
- <book>
  <title>DATA STRUCTURES</title>
  <author>Gilberg</author>
  <author>Forouzan</author>
  <author>Prasad</author>
</book>
</bookstore>
```

External DTD

In this type, an external DTD file is created and its name must be specified in the corresponding XML file.

The following example illustrates the use of external DTD.

Step 1: Creation of DTD file [Validatebooks.dtd]

```
<!DOCTYPE bookstore [  
    <!ELEMENT bookstore (book+)>  
    <!ELEMENT book (title,author+)>  
    <!ELEMENT title (#PCDATA)>  
    <!ELEMENT author (#PCDATA)>  
>
```

Step 2: Creation of XML document [BooksFile.xml]

```
<?xml version="1.0"?>  
<bookstore>  
    <book>  
        <title>WEB TECHNOLOGIES</title>  
        <author>Uttam.K.Roy </author>  
        <price>Rs.300 </price>  
    </book>  
  
    <book>  
        <title>DATA STRUCTURES</title>  
        <author>Gilberg </author>  
        <author>Forouzan</author>  
        <author>Prasad</author>  
    </book>  
</bookstore>
```

XML SCHEMA

- 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.

Syntax :

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

Example :

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

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.

Example :

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

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.

```

<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>

```

Difference between DTD and XSD

	DTD	XSD
1)	DTD stands for Document Type Definition .	XSD stands for XML Schema Definition.
2)	DTDs are derived from SGML syntax.	XSDs are written in XML.
3)	DTD doesn't support datatypes .	XSD supports datatypes for elements and attributes.
4)	DTD doesn't support namespace .	XSD supports namespace .
5)	DTD doesn't define order for child elements.	XSD defines order for child elements.
6)	DTD is not extensible .	XSD is extensible .
7)	DTD is not simple to learn ..	XSD is simple to learn because you don't need to learn new language..
8)	DTD provides less control on XML structure.	XSD provides more control on X

3. XML DOM:

- DOM Stands for **Document Object Model**.
- DOM is an Application Programming Interface (API).
- API is a set of data items and operations which can be used by the developers of Application programs.
- The W3C DOM Specification is divided into 3 parts:

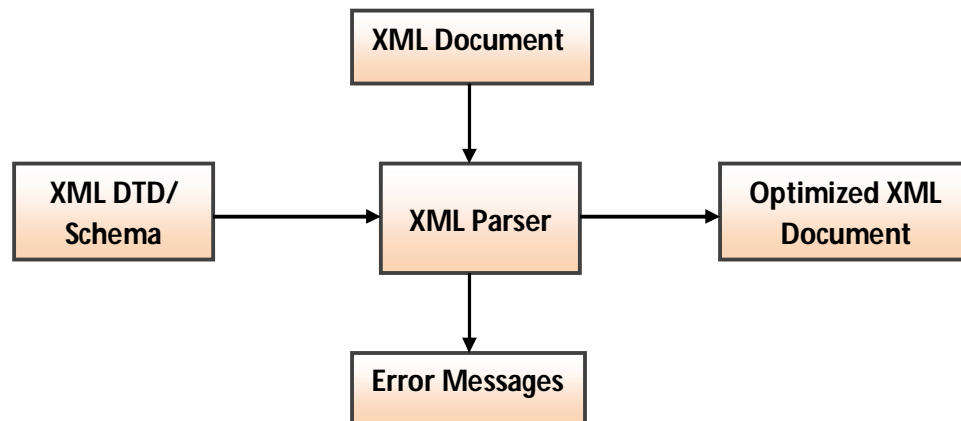
1. **Core DOM:** Defines standard/basic set of objects and interfaces for any structured documents.
 2. **HTML DOM:** Defines standard set of objects and interfaces for HTML documents.
 3. **XML DOM:** Defines standard set of objects and interfaces for XML documents only.
- “XML DOM is a set of Platform Independent and Language Neutral API which specifies the logical structure of XML documents and the ways in which they can be accessed and manipulated”.
 - XML DOM is a:
 1. A standard object model for XML
 2. A standard programming interface for XML
 3. Platform- and language-independent
 4. A W3C standard
 - XML DOM is for
 - Loading XML documents
 - Navigating and Searching through them
 - Accessing the elements of XML document
 - Deleting the elements of XML document and to
 - Modify the contents of existing elements
 - The DOM presents an XML document as a Tree-Structure (XML DOM tree) and stores the tree structure in memory.
 - In the tree structure
 - i. Each node of the tree, each XML element, is modelled as an Object
 - ii. This means that the node encompasses both data and behaviour, and the whole document can be seen as a single complex object.
 - iii. The XML DOM tree is a Hierarchical representation of an XML document.
 - Programming interfaces of DOM objects can be used with the help of properties and methods.

- The properties are meant for accessing the XML elements and Methods are used to perform some actions on the XML elements.
- **Various Properties are:**
 1. x.nodeName - the name of x
 2. x.nodeValue - the value of x
 3. x.parentNode - the parent node of x
 4. x.childNodes - the child nodes of x
 5. x.attributes - the attributes nodes of x
- **Various methods are:**
 1. x.getElementsByTagName(*name*) - get all elements with a specified tag name
 2. x.appendChild(*node*) - insert a child node to x
 3. x.removeChild(*node*) - remove a child node from x
 4. getAttribute(attribute-name) – returns values of an attribute
 5. setAttribute(attribute-name,attribute-value) – adds an attribute to an element

4. XML PARSERS:

- The primary goal of any XML processor/parser is to parse the given XML document.
- Parsers are used to check whether a given XML document
 - i. Is Well formed or not – It follows all of the syntax rules of XML.
 - ii. It obeys its own internal rules defined in the DTD or XML Schema
- The W3C recommendations for XML specify the external behaviour that parsers must have:

“A parser has to structure its output in a specific way, has to pass certain messages to applications, and has to handle specific types of Input”.
- Developers can use whatever language they want when implementing a parser.
- Java has rich source of in-built APIs for parsing the given XML document.



- Two models are commonly used for parsers:
 1. Tree based Parsers (DOM)
 2. Event based Parsers (SAX)

1. DOM Parser:

- A Tree based parser builds a static representation (tree structure) of the document and then parses it, so it requires more memory space.
- This tree may be updated by adding, removing or modifying the nodes at run time.
- DOM is a Tree based Parser, it works on DOM API
- DOM parser loads whole XML document into memory and constructs a tree structure of an XML document and used the tree structure to parse the document.
- Parsing XML file using DOM parser is quite fast if XML file is small but is not efficient for large XML files.
- DOM is a read-only parser that accesses the information of an XML document by interacting with tree nodes and using DOM API.

• Example:

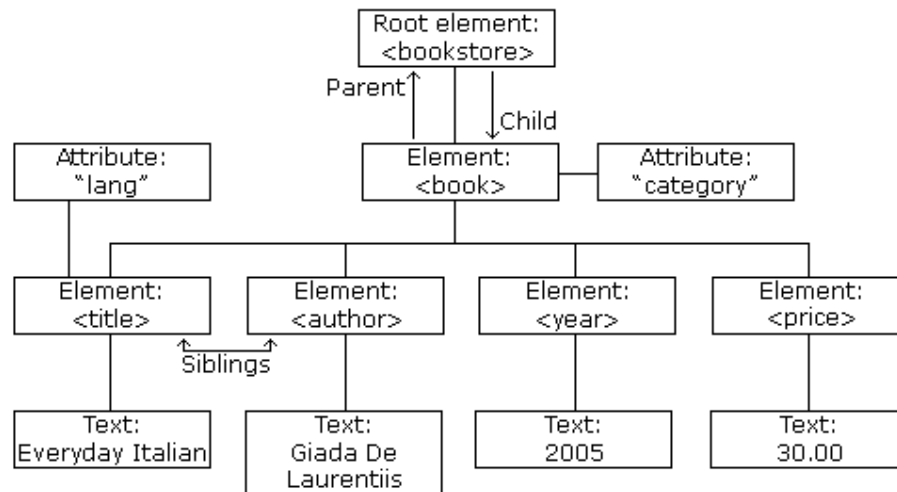
Book.xml:

```
<?xml version="1.0"?>
<bookstore>
  <book category="general">
    <title lang="italian">Everyday Italian </title>
    <author>Giada De Laurentiis</author>
```

```

    <year>2005</year>
    <price>30.00</price>
  </book>
</bookstore>

```



XML DOM Tree

Accessing elements defined in Book.xml using DOM API:

Syntax:

```
node.getElementsByTagName("tagname");
```

Example:

```

xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
xmlDoc.getElementsByTagName("author")[0].childNodes[0].nodeValue;
xmlDoc.getElementsByTagName("year")[0].childNodes[0].nodeValue;
xmlDoc.getElementsByTagName("price")[0].childNodes[0].nodeValue;
xmlDoc.getElementsByTagName("title")[0].getAttribute("lang");

```

Output:

```

Everyday Italian
Giada De Laurentiis
2005
30.00
Italian

```

2. SAX Parser:

- Event based parsers are also known as "Stream based Parsers".
- SAX stands for **S**imple **A**PI for **X**ML.

- SAX is an Event based parser that allows us to access the information of an XML document using sequence of events. It parses the XML file step by step so much suitable for large XML files.
- Because of event-driven nature of SAX, processing documents is generally faster than DOM parsers.
- DOM operates on the document as a whole, SAX parsers operate on each piece of the XML document sequentially.
- SAX parsers do not have to build large static models of the document in memory and are much space efficient.
- These parsers are read only and cannot be used to manipulate (insert, delete & update) the contents of an XML document.
- This model is used when parsing XML data across the network between applications and is widely used by Java programmers.
- It is recommended to use SAX parse for parsing large XML files because it doesn't require loading the whole XML document into memory.
- **Working:**
 - SAX parser fires an event when it encounters an opening tag of an element or an attribute.
 - The user defines a number of callback methods that will be called when events occur during parsing. The SAX events include:
 - XML Text nodes
 - XML Element Starts and Ends
 - XML Processing Instructions
 - XML Comments
- For accessing elements in the XML file it used SAX API which is a part of Java API.
- **Checking the well-formed nature of an XML document using SAX API:**

```
import java.io.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
public class SAXParser
{
```

```
public static void main(String args[])
{
    try
    {
        System.out.print("Enter name of XML document:");
        BufferedReader br=new BufferedReader(new
            InputStreamReader(System.in));
        String f=br.readLine();
        File fi=new File(f);
        if(fi.exists())
        {
            try
            {
                XMLReader
r=XMLReaderFactory.createXMLReader();
                r.parse(f);
                System.out.println("It is Well Formed");
            }
            catch(Exception e)
            {
                System.out.println("It is Not Well Formed");
            }
        }
    }
    catch(Exception e)
    {
        System.out.println(e);
    }
}
```

Output:

Enter name of XML document: Book.xml
It is Well Formed

➤ **DIFFERENCES BETWEEN SAX & DOM PARSERS:**

SAX Parser	DOM Parser
<ul style="list-style-type: none"> • Uses Stream based/Event based Parsing approach. 	<ul style="list-style-type: none"> • Uses tree based parsing approach
<ul style="list-style-type: none"> • Loads small parts of XML file in memory. 	<ul style="list-style-type: none"> • Loads the whole XML document in memory before processing.
<ul style="list-style-type: none"> • Does not create any internal (tree) structure for the given XML file. 	<ul style="list-style-type: none"> • A DOM parser creates a tree structure in memory from an Input document and then waits for requests from clients.
<ul style="list-style-type: none"> • It is much space efficient. 	<ul style="list-style-type: none"> • More space is required in case of big input document.
<ul style="list-style-type: none"> • Uses SAX API which is a part of Java API. 	<ul style="list-style-type: none"> • Uses DOM API.
<ul style="list-style-type: none"> • SAX is read only parser. 	<ul style="list-style-type: none"> • DOM is read-write parser.
<ul style="list-style-type: none"> • We cannot insert or delete elements. 	<ul style="list-style-type: none"> • We can insert, delete & manipulate elements and attributes.
<ul style="list-style-type: none"> • Traversal can be done only from top to bottom. 	<ul style="list-style-type: none"> • Traversal can be done in any direction.
<ul style="list-style-type: none"> • Efficient for parsing large XML files and faster than DOM. 	<ul style="list-style-type: none"> • Faster if the size of XML file is small and not suitable for large XML files.

5. XSLT:

- XSL stands for **EX**tensible **S**tylesheet **L**anguage, and is a style sheet language for XML documents.
- XSL consists of four parts:
 - XSLT - a language for transforming XML documents
 - XPath - a language for navigating in XML documents
 - XSL-FO - a language for formatting XML documents
 - XQuery - a language for querying XML documents
- XSLT stands for **XSL** Transformations.

- XSLT is used to transform an XML document into another XML document, or another type of document that is recognized by a browser, like HTML and XHTML.
- XSLT uses XPath to find information in an XML document. XPath is used to navigate through elements and attributes in XML documents.
- In the transformation process, XSLT uses XPath to define parts of the source document that should match one or more predefined templates. When a match is found, XSLT will transform the matching part of the source document into the result document.
- **Some Important XSLT Elements:**

Element	Description
apply-templates	Applies a template rule to the current element or to the current element's child nodes
attribute	Adds an attribute
choose	Used in conjunction with <when> and <otherwise> to express multiple conditional tests
copy	Creates a copy of the current node (without child nodes and attributes)
element	Creates an element node in the output document
pi	Processing Instruction is inserted into the output
for-each	Loops through each node in a specified node set
if	Contains a template that will be applied only if a specified condition is true
stylesheet	Defines the root element of a style sheet
template	Rules to apply when a specified node is matched
value-of	Extracts the value of a selected node
when	Specifies an action for the <choose> element

- **Example:**
Book.xml:

```
<?xml version="1.0" ?>
<?xml-stylesheet href="book.xsl" type="text/xsl" ?>
<catalog>
```



```

<bookdetails>
  <title>XML bible</title>
  <author>watson</author>
  <isbn>A121</isbn>
  <publication>TMH</publication>
  <edition>2</edition>
  <price>50</price>
</bookdetails>
<bookdetails>
  <title>HTML Bible</title>
  <author>chris bates</author>
  <isbn>B222</isbn>
  <publication>Pearson</publication>
  <edition>5</edition>
  <price>39</price>
</bookdetails>
</catalog>

```

Book.xsl:

```

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
  <xsl:template match="/">
    <html>
      <body>
        <table border="2" bordercolor="black" cellspacing="0" align="center"
style="font-family:calibri;text-align:center">
          <tr style="background-color:gray;text-transform:uppercase;">
            <th>title</th>
            <th>author</th>
            <th>isbn number</th>
            <th>publication</th>
            <th>edition</th>
            <th>price</th>
          </tr>
          <xsl:for-each select="catalog/bookdetails">
            <tr>
              <td><xsl:value-of select="title" /></td>
              <td>
                <xsl:value-of select="author" /></td>
              <td><xsl:value-of select="isbn" /></td>
              <td><xsl:value-of select="publication" /></td>

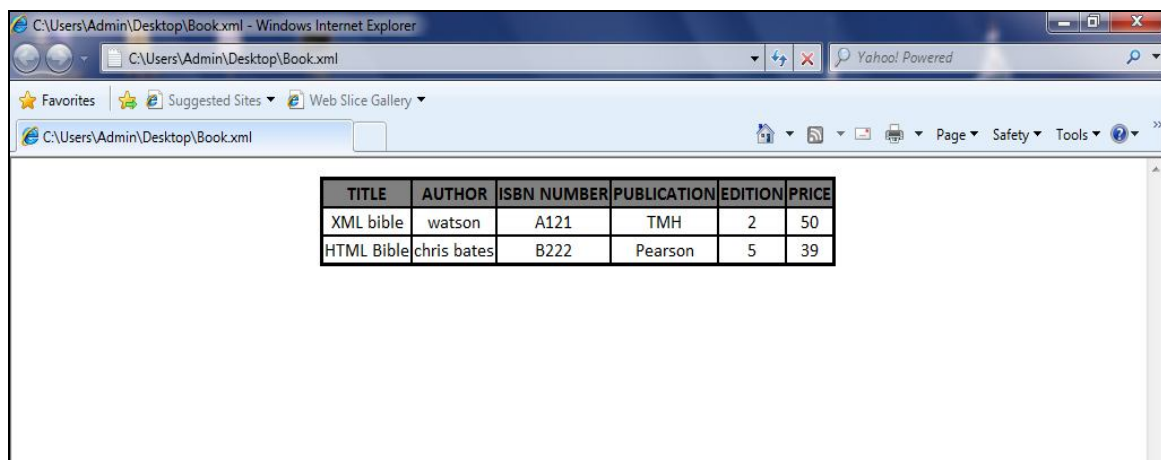
```

```

<td><xsl:value-of select="edition" /></td>
<td><xsl:value-of select="price" /></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>

```

Output: Book.xml



6. USING CSS WITH XML:

- XML files can be combined with CSS. When displayed in a browser the XML files will be displayed just as if it were HTML files.
- XML Document can be presented with effective Styles.

Example:

Book.css:

```

title
{
    Color:red;
    Text-decoration:underline;
    Font-size:20pt;
}
author
{
    Color:blue;
    Text-decoration:overline;
}

```

```
        Font-size:20pt;
    }
    price
    {
        Color:green;
        Font-family:calibri;
        Font-size:20pt;
    }
```

Book.xml:

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/css" href="Book.css"?>
<catalog>
<bookdetails>
    <title>XML bible</title>
    <author>watson</author>
    <isbn>A121</isbn>
    <publication>TMH</publication>
    <edition>2</edition>
    <price>50</price>
</bookdetails>
<bookdetails>
    <title>HTML Bible</title>
    <author>chris bates</author>
    <isbn>B222</isbn>
    <publication>Pearson</publication>
    <edition>5</edition>
    <price>39</price>
</bookdetails>
</catalog>
```

Output:



Example 2:

Ocean.XML

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

```
<!-- XML demonstration -->
```

```
<?xml-stylesheet type="text/css" href="style9.css"?>
```

```
<!DOCTYPE planet>
```

```
<planet>
```

```
<ocean>
```

```
<name>Arctic</name>
```

```
<area>13,000</area>
```

```
<depth>1,200</depth>
```

```
</ocean>
```

```
<ocean>
```

```
<name>Atlantic</name>
```

```
<area>87,000</area>
```

```
<depth>3,900</depth>
```

```
</ocean>
```

```
<ocean>
```

```
<name>Pacific</name>
```

```
<area>180,000</area>
```

```
<depth>4,000</depth>
```

```
</ocean>
```

```
<ocean>
```

```
<name>Indian</name>
```

```
<area>75,000</area>
```

```
<depth>3,900</depth>
```

```
</ocean>
```

```
<ocean>
```

```
<name>Southern</name>
```

```
<area>20,000</area>
```

```
<depth>4,500</depth>
</ocean>
```

```
</planet>
```

Style9.css

```
planet:before {
  display: block;
  width: 8em;
  font-weight: bold;
  font-size: 200%;
  content: "Oceans";
  margin: -.75em 0px .25em -.25em;
  padding: .1em .25em;
  background-color: #cdf;
}
```

```
planet {
  display: block;
  margin: 2em 1em;
  border: 4px solid #cdf;
  padding: 0px 1em;
  background-color: white;
}
```

```
ocean {
  display: block;
  margin-bottom: 1em;
}
```

```
name {
  display: block;
  font-weight: bold;
  font-size: 150%;
}
```

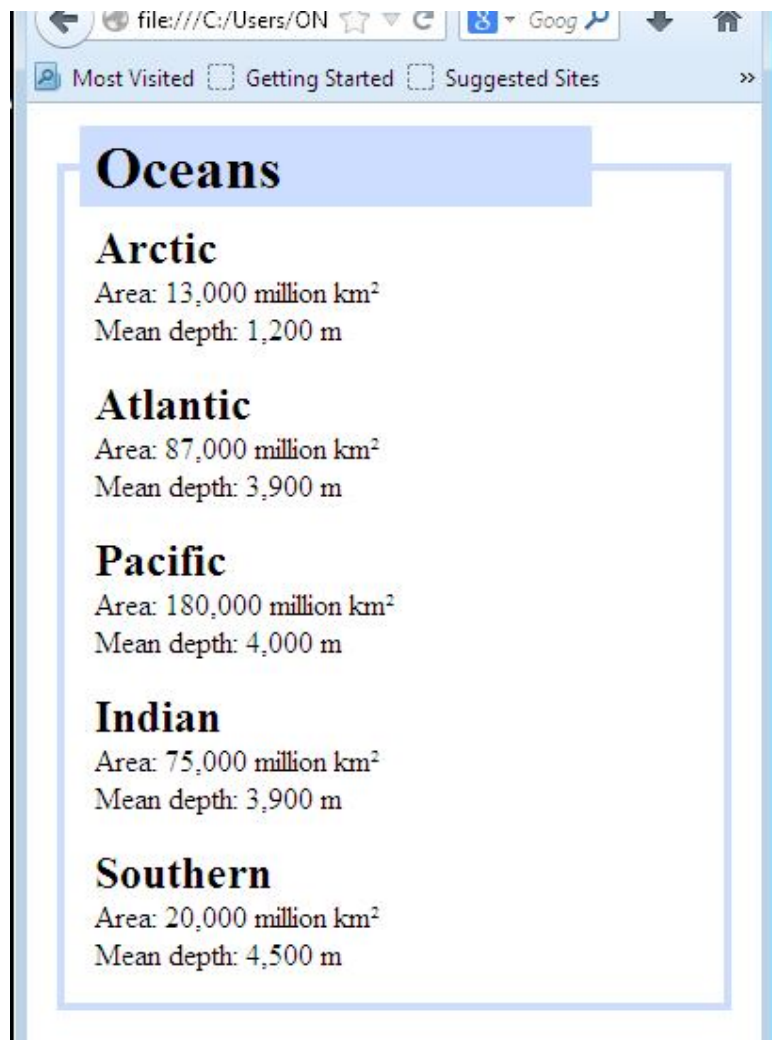
```
area {
  display: block;
}
```

```
area:before {
  content: "Area: ";
}
```

```
area:after {
  content: " million km\B2";
}
```

```
}  
depth {  
  display: block;  
}  
depth:before {  
  content: "Mean depth: ";  
}  
depth:after {  
  content: " m";  
}
```

Output:



UNIT-III
Assignment-Cum-Tutorial Questions
SECTION-A

Objective Questions

1. What does XML stand for? []
A. eXtra Modern Link B. eXtensible Markup Language
C. Example Markup Language D. X-Markup Language
2. Well formed XML document means []
A. it contains a root element
B. it contain an element
C. it contains one or more elements
D. must contain one or more elements and root element must contain all other elements
3. XML uses the features of []
A. HTML B. XHTML C. VML D. SGML
4. An entity in XML is a _____. []
A. class B. logical information
C. simple information D. flexible information storage unit
5. There is a way of describing XML data, how? []
A. XML uses a DTD to describe the data
B. XML uses XSL to describe data
C. XML uses a description node to describe data
D. Both A and C.
6. What does DTD stand for? []
A. Direct Type Definition B. Document Type Definition
C. Do The Dance D. Dynamic Type Definition
7. The use of a DTD in XML development is: []
A. required when validating XML documents
B. no longer necessary after the XML editor has been customized
C. used to direct conversion using an XSLT processor
D. a good guide to populating a templates to be filled in when generating an XML document automatically.
8. Parameter 'entities' can appear in []
A. XML file B. DTD file C. XSL file D. Both A and B
9. The XML DOM object is []
A. Entity B. Entity Reference
C. Comment Reference D. Comment Data

10. What is an advantage of XML compared to HTML? []
A. XML works on more platforms.
B. XML is suited to using Web pages as front ends to databases.
C. XML was designed for portable phones.
D. XML is simpler to learn than HTML.
11. The attribute used to define a new namespace is []
A. XMLNS B. XmlNameSpace C. Xmlns D. XmlNs
12. XSL stands for []
A) Extensible Style sheet Language
B) Extensible Style Language
C) Exclusive Stylesheet Language
D) Exclusive Style Language
13. Which of the following XML documents are well-formed? []
A. `<firstElement>some text goes here
<secondElement>another text goes here</secondElement>
</firstElement>`
B. `<firstElement>some text goes here</firstElement>
<secondElement> another text goes here</secondElement>`
C. `<firstElement>some text goes here
<secondElement> another text goes here</firstElement>
</secondElement>`
D. `</firstElement>some text goes here
</secondElement>another text goes here
<firstElement>`
14. Which of the following XML fragments are well-formed? []
A. `<myElement myAttribute="someValue"/>`
B. `<myElement myAttribute=someValue/>`
C. `<myElement myAttribute='someValue'/>`
D. `<myElement myAttribute="someValue' />`
15. The syntax for writing default values for element is []
A. `<xsd:element name="max" type="xsd:integer" value=" 100" />`
B. `<xsd:element name="max" type="xsd:integer" fixValue=" 100" />`
C. `<xsd:element name="max" type="xsd:integer" default=" 100" />`
D. `<xsd:element name="max" type="xsd:integer" defaultval=" 100" />`
16. The XSL formatting object use to hold the content of the label of a list item is []
A. list-block B. list item

C. list-item-body

D. list-item-label

17. In XSLT style sheet we have syntax to match elements with id as (if id is " change") []
- A. <xsl:template match=" id('change')" >
 - B. <xsl:template match=" (change)">
 - C. <xsl:template match=" change">
 - D. <xsl:template match-id="Change">

SECTION-B

SUBJECTIVE QUESTIONS

1. What is XML? Explain the various building blocks of XML file.
2. Explain the basic structure of an XML document and Differentiate XML & HTML.
3. What is Document Type Definition (DTD)? Explain in detail Internal and External DTDs with examples.
4. Define an XML schema. Show how an XML schema can be created with an example.
5. Explain various types of XML Schema data types used.
6. Differentiate DOM and SAX XML Parsers.
7. Explain the various XSL elements in detail with Example.
8. Write the use of CSS in XML Document. Explain with an Example
9. Write down Internal and External DTDs for the following XML file

Students.xml:

```
<students>

  <student roll="1">
    <firstname> James </firstname>
    <lastname> Watson </lastname>
    <year> 3 </year>
    <courses>
      <course id="1">
        <name> Advanced Java </name>
      </course>
      <course id="2">
        <name> Web Technologies </name>
      </course>
    </courses>
  </student>
</students>.
```

10. Write a DTD for the XML document which has the Employee details with following fields arranged in a tabular format. Also assume the values of each field.

Employee Id	Name	Department	Designation	Sal
-------------	------	------------	-------------	-----

11. Design an XML Schema for hospital information management system.

12. Design XSL Sheet by using following XSL elements.

- (i) template
- (ii) stylesheet
- (iii) for-each
- (iv) value-of
- (vi) choose.

13. Design an XML Schema for the following XML Document

```
<?xml version="1.0"?>
<bookstore>
  <book>
    <title>WEB TECHNOLOGIES</title>
    <author>Uttam.K.Roy </author>
    <price>Rs.300 </price>

  </book>
  <book>
    <title>DATA STRUCTURES</title>
    <author>Gilberg </author>
    <author>Forouzan</author>
    <author>Prasad</author>
  </book>
</bookstore>
```

14. Design XML Schema for the following XML Document.

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


UNIT - 4

Web Servers and Servlets

SYLLABUS:

Tomcat web server,
Introduction to Servlets, Lifecycle of a Servlet,
JSDK,
The Servlet API- javax.servlet Package, javax.servlet.http package
Reading Servlet parameters,
Reading Initialization parameters,
Using Cookies-Session Tracking.

Objective:

- To develop real time web applications.
- To get acquainted with skills for creating websites and web apps through learning various technologies like HTML, CSS, JavaScript, XML and Servlets,

Learning Outcomes:

Students will be able to

- Learn how to install Tomcat web server,
 - Know about Servlets and Lifecycle of a servlet.
 - Define JSDK.
 - Differentiate between javax.servlet Package, javax.servlet.http package
 - Know how to read Servlet parameters and Initialization parameters, Design cookies and session tracking.

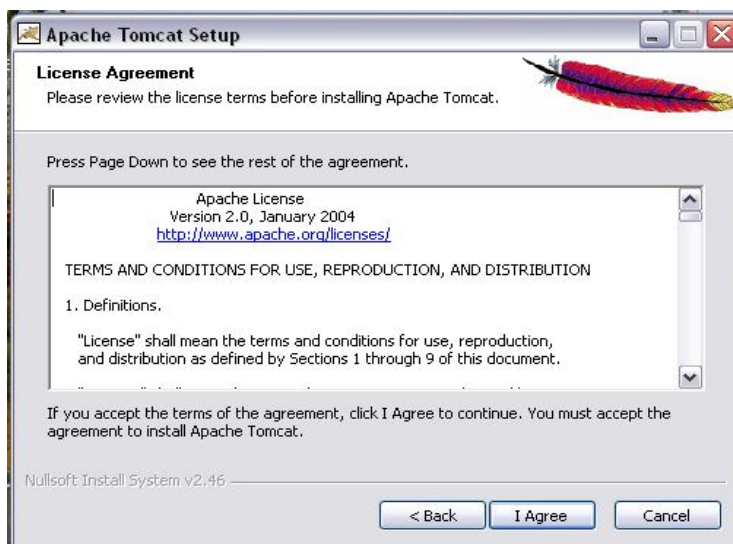
1. Tomcat web server:

- **Apache Tomcat**, often referred to as **Tomcat**.
- It is an open-source web server developed by the Apache Software Foundation (ASF).
- Tomcat implements several Java EE specifications including Java Servlet, Java Server Pages(JSP), Java EL, and Web Socket, and provides a "pure Java" HTTP web server environment in which Java code can run.

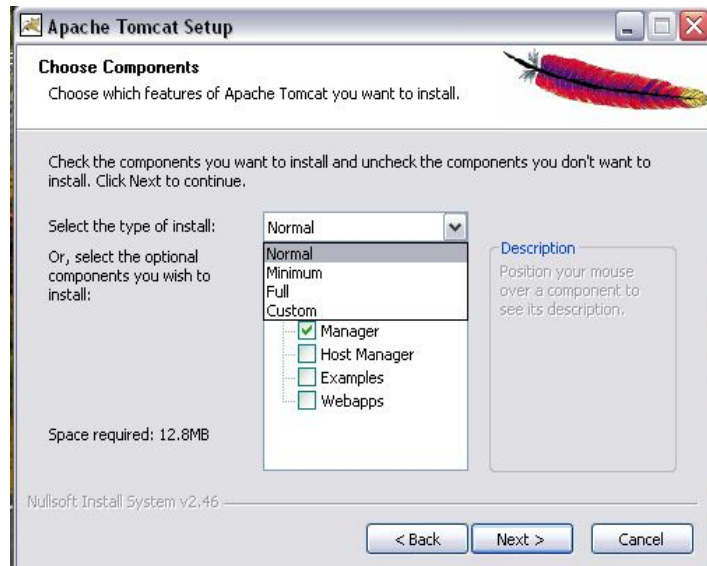
Installing APACHE TOMCAT:-



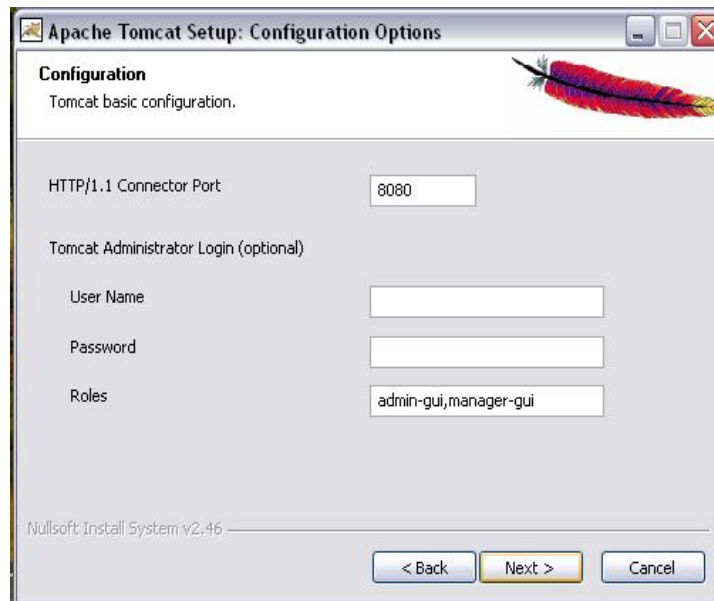
Accepting License Agreement:



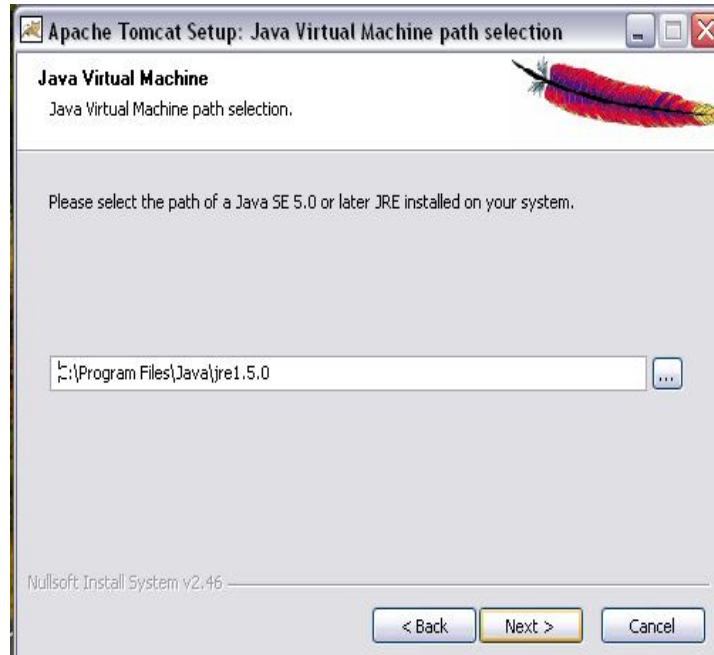
Choosing Components to be installed:



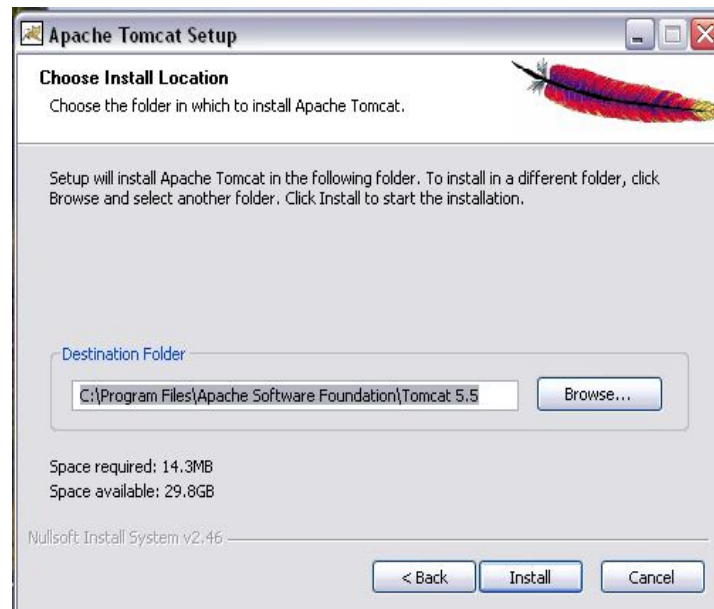
Configuring Port Number, Username & Password:



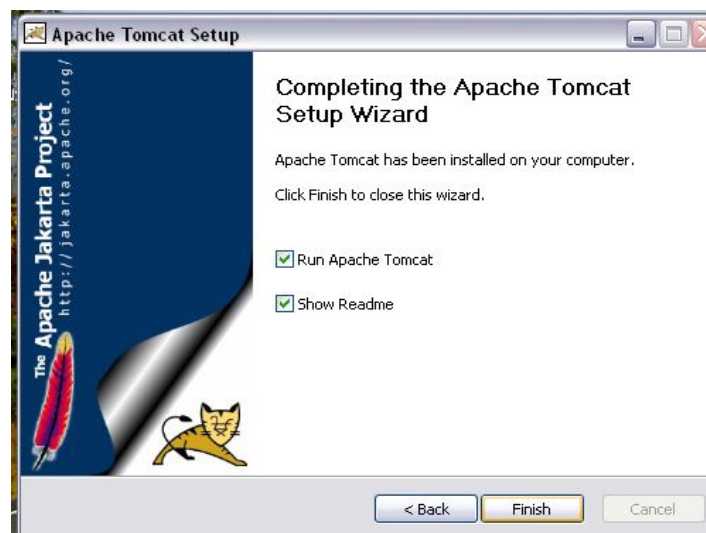
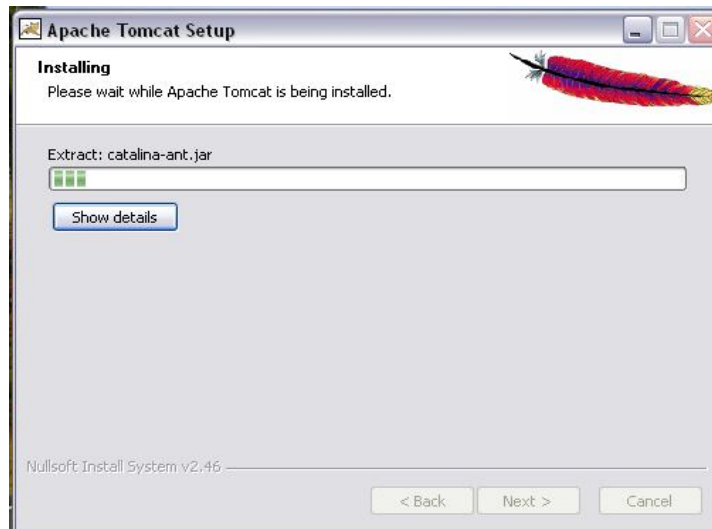
Choosing the JVM:-



Choosing the Installation location:



Installation Progress:

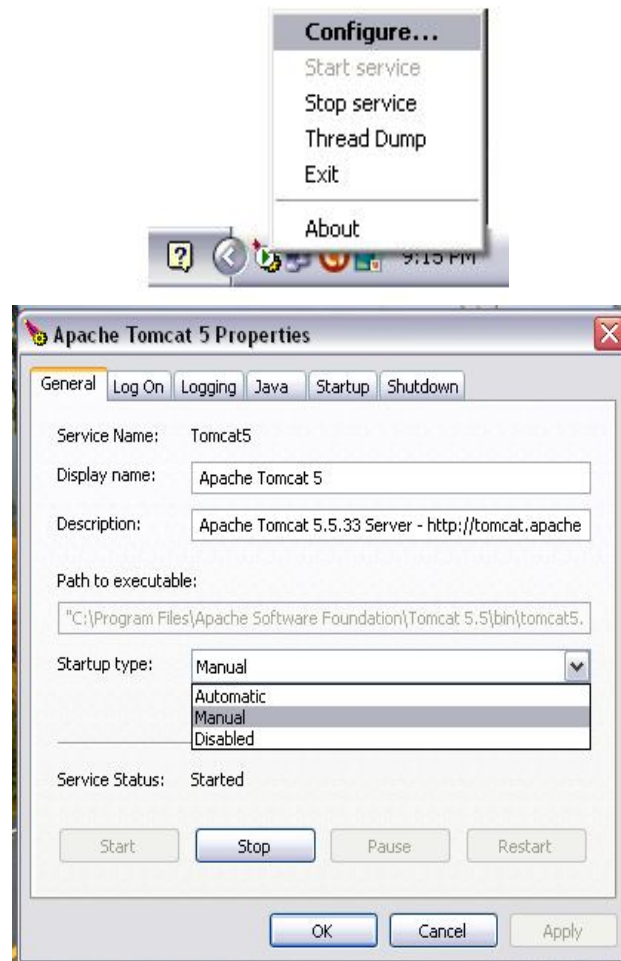


Starting TOMCAT Service:



Configuring TOMCAT:

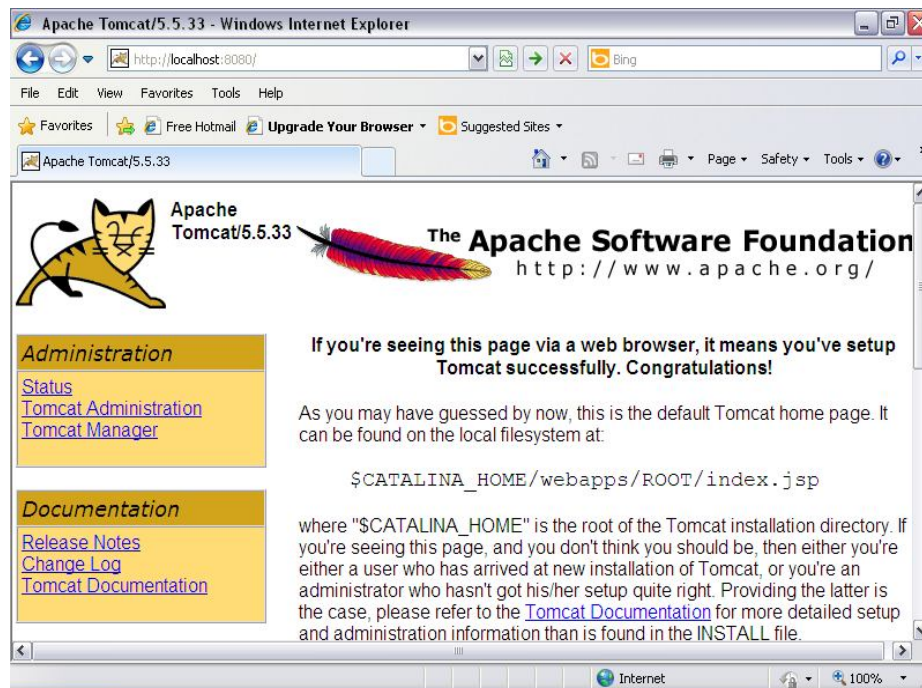
This can be done in 2 ways



Testing TOMCAT:

To test the server - assuming you're running Tomcat on the same machine as the browser and that you're using the default port for Tomcat (8080) - open a browser and enter the following URL in the Location/Address field: *http://localhost:8080/*

The Tomcat main page is shown in the browser and you can now run all servlet and JSP examples bundled with Tomcat to make sure everything works.



TOMCAT Directory Structure:

The TOMCAT Working Directory consists of following folders:

- Bin--- Contains startup, shutdown scripts and other files
 - Startup.bat
 - Shutdown.bat
 - Catalina.bat
- Common--- Contains common classes for servlet and JSP applications
- Conf--- Contains server configuration files including server.xml and global web.xml
- Logs--- Contains TOMCAT log files
- Server--- Contains server's active file
- Shared--- Contains classes and resources that must be shared across all web applications
- Temp--- Directory used by JVM for temporary files
- Webapps--- Contains servlet and JSP applications to be served by TOMCAT
- Work--- Contains files and directories created by JSP container. This directory holds the servlet equivalent of JSP
- Notice.file--- Contains general info about the product.
- Running.txt--- Contains details about how to get started with TOMCAT
- License.txt--- Contains license for the product

WEB SERVER:

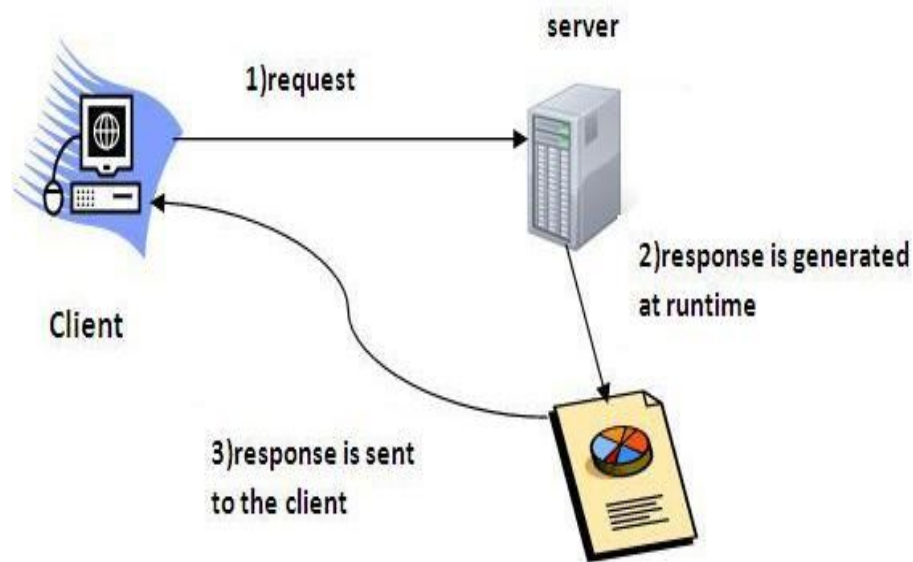
A Web server is a computer or virtual machine used to run web applications. The primary function of a web server is to store, process and deliver web pages to clients. The communication between client and server takes place using the Hypertext Transfer Protocol (HTTP). Pages delivered are most frequently HTML documents, which may include images, style sheets and scripts in addition to text content. A user agent, commonly a web browser, initiates communication by making a request for a specific resource using HTTP and the server responds with the content of that resource or an error message if unable to do so.

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. Some of them are:

- **Apache Tomcat Server**
- ***Microsoft's Internet Information Services (IIS) Windows Server***
- **Oracle Weblogic Server**
- **Google Web Server**

2. Introduction to Servlets:

- Servlet is a technology i.e. used to create web application.
- Servlet is an API that provides many interfaces and classes including documentations.
- Servlet is an interface that must be implemented for creating any servlet.
- Servlet is a class that extend the capabilities of the servers and respond to the incoming request. It can respond to any type of requests.
- Servlet is a web component that is deployed on the server to create dynamic web page.

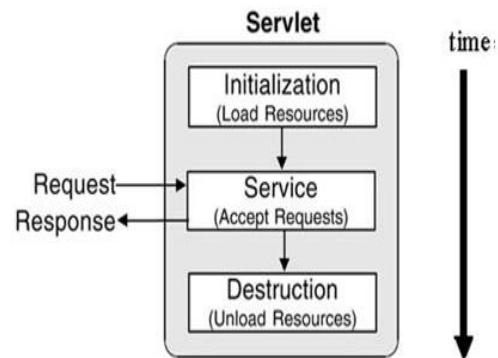


3. Lifecycle of a Servlet:

A servlet life cycle can be defined as the entire process from its creation till the destruction. The web container maintains the life cycle of a servlet instance. This life cycle is expressed in the API by the `init`, `service`, and `destroy` methods of the `javax.servlet.Servlet` interface that all servlets must implement directly or indirectly through the `GenericServlet` or `HttpServlet` abstract classes.

The following are the paths followed by a servlet

- The servlet is initialized by calling the **`init ()`** method.
- The servlet calls **`service()`** method to process a client's request.
- The servlet is terminated by calling the **`destroy()`** method.
- Finally, servlet is garbage collected by the garbage collector of the JVM.



init(): The web container calls the init method only once after creating the servlet instance. It is called when the servlet is first created, and not called again for each user request. The init method is used to initialize the servlet.

```

public void init(ServletConfig config) throws ServletException
{
    //code
}

```

service(): The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service method. If servlet is initialized, it calls the service method.

```

public void service(ServletRequest request, ServletResponse response) throws
ServletException, IOException
{
    //code
}

```

destroy(): The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc.

```

public void destroy()
{
    //code
}

```

4. JSDK:

- Java Servlet Development Kit contains the packages that are needed to create servlets.
- A utility known as a ServletRunner is also included which enables us to test some of the servlets that are created.
- JSDK can be downloaded from sun Microsystems website at java.sun.com.
- The following are the basic steps to build and test a Servlet:
 1. Create and compile servlet source code
 2. Start the ServletRunner Utility
 3. Start a Web browser and Request the servlet.

5. The Servlet API:

Consists of two packages

1. **javax.servlet** : This package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol.
2. **javax.servlet.http** : This package contains interfaces and classes that are responsible for http requests only.

javax.servlet package

Interface	
<u>Servlet</u>	Defines methods that all servlets must implement.
<u>ServletConfig</u>	A servlet configuration object used by a servlet container to pass information to a servlet during initialization.
<u>ServletContext</u>	Defines a set of methods that a servlet uses to communicate with its servlet container, for example, to get the MIME type of a file, dispatch requests, or write to a log file.
<u>ServletRequest</u>	Defines an object to provide client request information to a servlet.

<u>ServletResponse</u>	Defines an object to assist a servlet in sending a response to the client.
<u>SingleThreadModel</u>	Deprecated. As of Java Servlet API 2.4, with no direct replacement.

Class	
<u>GenericServlet</u>	Defines a generic, protocol-independent servlet.
<u>ServletInputStream</u>	Provides an input stream for reading binary data from a client request, including an efficient readLine method for reading data one line at a time.
<u>ServletOutputStream</u>	Provides an output stream for sending binary data to the client.
<u>ServletException</u>	Defines a general exception a servlet can throw when it encounters difficulty.
<u>UnavailableException</u>	Defines an exception that a servlet or filter throws to indicate that it is permanently or temporarily unavailable.

Servlet

Methods
<ul style="list-style-type: none"> • Void <u>destroy()</u> Called by the servlet container to indicate to a servlet that the servlet is being taken out of service. • <u>ServletConfig</u> <u>getServletConfig()</u> Returns a <u>ServletConfig</u> object, which contains initialization and startup parameters for this servlet.
<ul style="list-style-type: none"> • String <u>getServletInfo()</u> Returns information about the servlet, such as author, version, and copyright. • Void <u>init</u>(<u>ServletConfig</u> config) Called by the servlet container to indicate to a servlet that the servlet is being placed into service. • Void <u>service</u>(<u>ServletRequest</u> req, <u>ServletResponse</u> res) Called by the servlet container to allow the servlet to respond to a request.

ServletConfig

Method

- String getInitParameter(java.lang.String name)
Returns a String containing the value of the named initialization parameter, or null if the parameter does not exist.
- Enumeration getInitParameterNames()
Returns the names of the servlet's initialization parameters as an Enumeration of Stringobjects, or an empty Enumeration if the servlet has no initialization parameters.
- ServletContext getServletContext()
Returns a reference to the ServletContext in which the caller is executing.
- String getServletName()
Returns the name of this servlet instance.

- **javax.servlet.http package:**

Interfaces:

There are many interfaces in javax.servlet.http package. They are as follows:

1. HttpServletRequest
2. HttpServletResponse
3. HttpSession
4. HttpSessionListener
5. HttpSessionAttributeListener
6. HttpSessionBindingListener
7. HttpSessionActivationListener
8. HttpSessionContext (deprecated now)

Classes:

There are many classes in javax.servlet.http package. They are as follows:

1. HttpServlet
2. Cookie
3. HttpServletRequestWrapper
4. HttpServletResponseWrapper
5. HttpSessionEvent
6. HttpSessionBindingEvent
7. HttpUtils (deprecated now)

Reading servlet parameters /Reading Form Data:

- HTML Form uses <form> tag to work with forms.
- <form> tag uses either **GET** method or **POST** method.
- If the form uses **GET** method in the form tag, then use **doGet()** method in the servlet.
- If the form uses **POST** method in the form tag, then use **doPost()** method in the servlet.
- Instead of adding request parameters to the end of the URL, it is also possible to POST them as actual data.

Servlets handles form data using the following methods.

- **getParameter():** to get the value of a form parameter.
- **getParameterValues():** 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.

Example:

Write a servlet program to read user 's name from a web page and greet with user's name.

Index.html

```
<html>
<body>
<form      action="http://localhost:4040/demo/greet"
method="get">
    Enter your name : <input type="text" name="uname">
    <br> <input type="submit" value="Greet me">
</form>
</body>
</html>
```

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class DemoServlet extends HttpServlet
{
    public void
doGet(HttpServletRequest req, HttpServletResponse
res) throws ServletException, IOException
    {
        res.setContentType("text/html");
        PrintWriter pw=res.getWriter();
        String name=req.getParameter("uname");
        pw.println("Welcome "+name);
    }
}
```

web.xml

```
<?xml version="1.0"?>
<web-app>
<servlet>
<servlet-name>MyServlet</servlet-name>
<servlet-class>DemoServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>MyServlet</servlet-name>
<url-pattern>/greet</url-pattern>
</servlet-mapping>
</web-app>
```

Initialization Parameters :

- Using init parameters, It is not needed to edit the servlet file. This concept provides facility to modify web.xml file to supply init parameters.
- 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.

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.

Methods of ServletConfig interface

1. **public String getInitParameter(String name)**:Returns the parameter value for the specified parameter name.
2. **public Enumeration getInitParameterNames()**:Returns an enumeration of all the initialization parameter names.
3. **public String getServletName()**:Returns the name of the servlet.
4. **public ServletContext getServletContext()**:Returns an object of ServletContext.

To get the object of ServletConfig,use the following method of Servlet interface.

getServletConfig()

Syntax to provide the initialization parameter for a servlet

```
<web-app>
  <servlet>
    .....
    <init-param>
      <param-name>parametername</param-
name>
      <param-value>parametervalue</param-
value>
    </init-param>
    .....
  </servlet>
</web-app>
```

The **<init-param>** is used to specify the initialization parameter for a servlet.

1. .java file (This is our servlet file)
2. .xml file (We mention init parameters here)

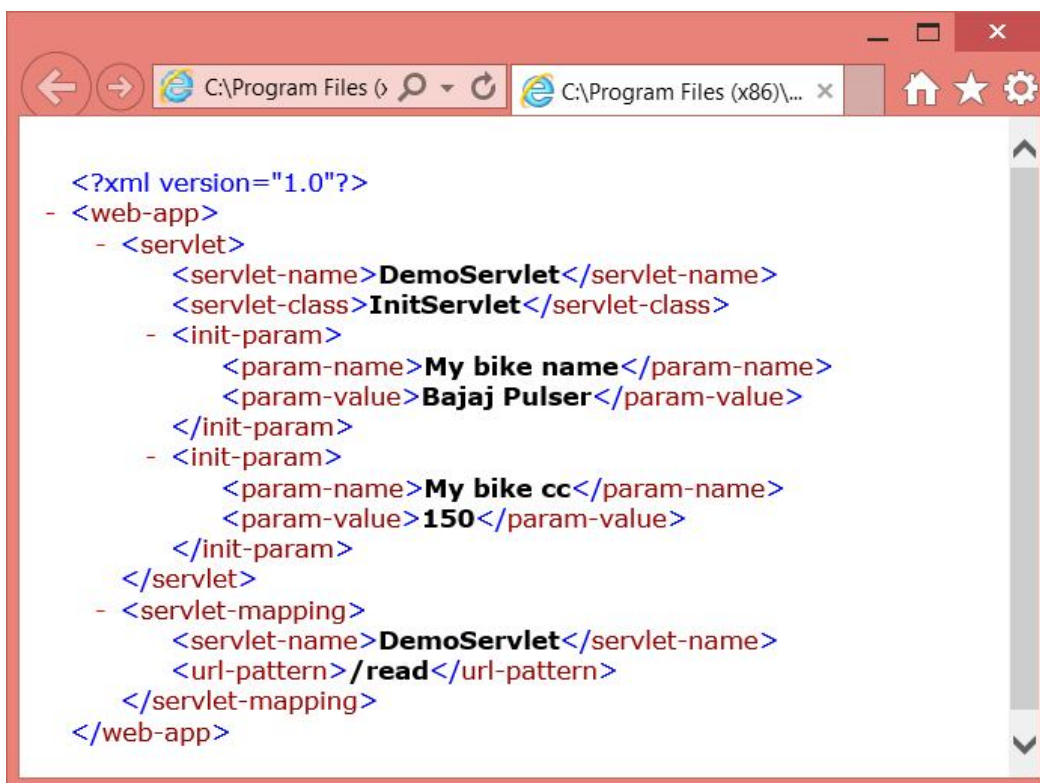
Java file must be compiled to generate a .class file.

Example:

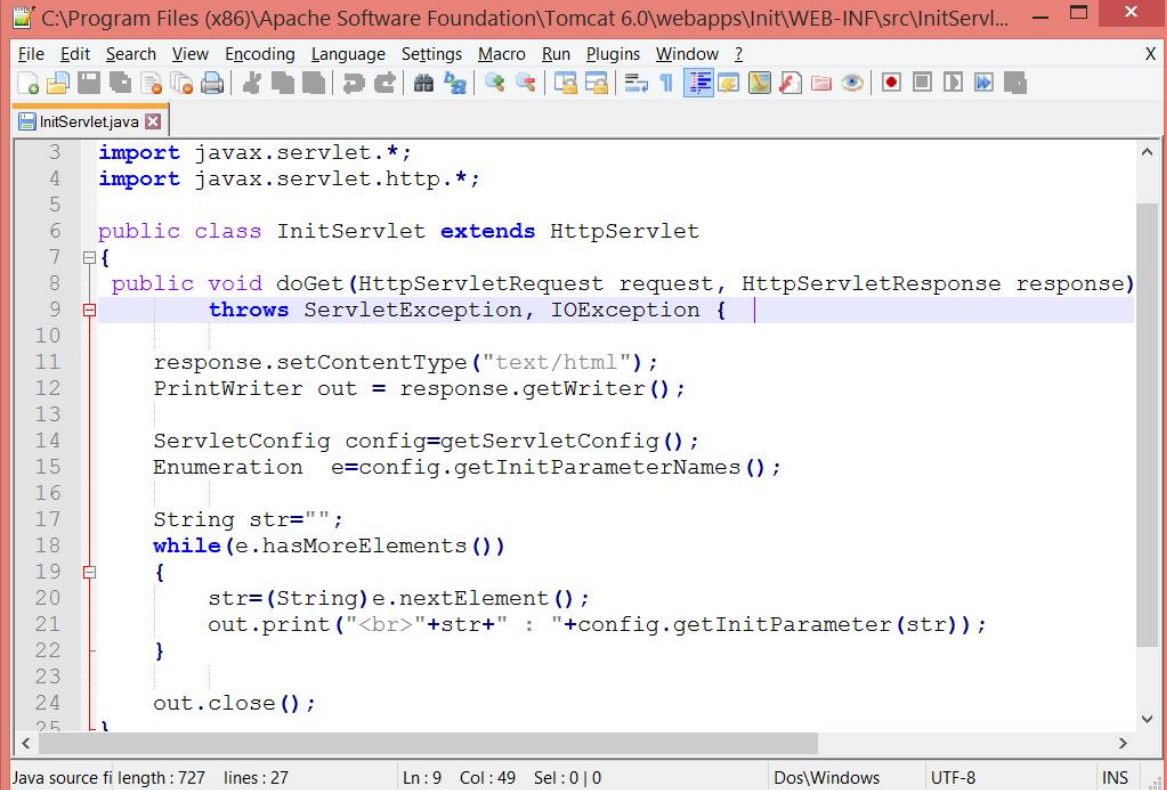
We need two files in this example.

1. InitServlet.java
2. web.xml

Compile **InitServlet.java** file to generate **InitServlet.class** and place it in classes folder.



```
<?xml version="1.0"?>
- <web-app>
  - <servlet>
    <servlet-name>DemoServlet</servlet-name>
    <servlet-class>InitServlet</servlet-class>
  - <init-param>
    <param-name>My bike name</param-name>
    <param-value>Bajaj Pulsar</param-value>
  </init-param>
  - <init-param>
    <param-name>My bike cc</param-name>
    <param-value>150</param-value>
  </init-param>
</servlet>
- <servlet-mapping>
  <servlet-name>DemoServlet</servlet-name>
  <url-pattern>/read</url-pattern>
</servlet-mapping>
</web-app>
```



```
3 import javax.servlet.*;
4 import javax.servlet.http.*;
5
6 public class InitServlet extends HttpServlet
7 {
8     public void doGet(HttpServletRequest request, HttpServletResponse response)
9         throws ServletException, IOException {
10
11         response.setContentType("text/html");
12         PrintWriter out = response.getWriter();
13
14         ServletConfig config=getServletConfig();
15         Enumeration e=config.getInitParameterNames();
16
17         String str="";
18         while(e.hasMoreElements())
19         {
20             str=(String)e.nextElement();
21             out.print("<br>" +str+ " : "+config.getInitParameter(str));
22         }
23
24         out.close();
25 }
```

Java source file length : 727 lines : 27 Ln: 9 Col: 49 Sel: 0|0 Dos\Windows UTF-8 INS



Session Tracking:

Session simply means a particular interval of time.

Session Tracking is a way to maintain state (data) of a user. It is also known as **session management** in servlet.

- Http protocol is a stateless 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.

Using Cookies:

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

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

Here are a few facts to know about cookies:

1. Cookies are domain specific i.e. a domain cannot read or write to a cookie created by another domain. This is done by the browser for security purpose.
2. Cookies are browser specific. Each browser stores the cookies in a different location. The cookies are browser specific and so a cookie created in one browser(e.g in Google Chrome) will not be accessed by another browser(Internet Explorer/Firefox).
3. Most of the browsers store cookies in text files in clear text. So it's not secure at all and no sensitive information should be stored in cookies.
4. Most of the browsers have restrictions on the length of the text stored in cookies. It is 4096(4kb) in general but could vary from browser to browser.
5. Some browsers limit the number of cookies stored by each domain(20 cookies). If the limit is exceeded, the new cookies will replace the old cookies.

6. Cookies can be disabled by the user using the browser properties. So unless you have control over the cookie settings of the users (for e.g. intranet application), cookies should not be used.
7. Cookie names are case-sensitive.

Advantage of Cookies

1. Simplest technique of maintaining the state.
2. Cookies are maintained at client side.
3. Occupies less memory, do not require any server resources and are stored on the user's computer so no extra burden on server.
4. We can configure cookies to expire when the browser session ends (session cookies) or they can exist for a specified length of time on the client's computer (persistent cookies).
5. Cookies persist a much longer period of time than Session state.

Disadvantage of Cookies

1. It will not work if cookie is disabled from the browser.
2. Only textual information can be set in Cookie object.
3. Several limitations exist on the size of the cookie text(4kb in general), number of cookies(20 per site in general), etc.
4. Cookies will not work if the security level is set to high in the browser.
5. Users can delete cookies.
6. Users browser can refuse cookies,so your code has to anticipate that possibility.
7. Complex type of data not allowed (e.g. dataset etc). It allows only plain text (i.e. cookie allows only string content)

Useful Methods of Cookie class

There are given some commonly used methods of the Cookie class.

Method	Description
public void setMaxAge(int expiry)	Sets the maximum age of the cookie in seconds.

<code>public String getName()</code>	Returns the name of the cookie. The name cannot be changed after creation.
<code>public String getValue()</code>	Returns the value of the cookie.
<code>public void setName(String name)</code>	changes the name of the cookie.
<code>public void setValue(String value)</code>	changes the value of the cookie.

Other methods required for using Cookies

For adding cookie or getting the value from the cookie, we need some methods provided by other interfaces. They are:

1. **public void addCookie(Cookie ck):**method of HttpServletResponse interface is used to add cookie in response object.
2. **public Cookie[] getCookies():**method of HttpServletRequest interface is used to return all the cookies from the browser.

Statements to create Cookies.

```
Cookie  
c=new Cookie("user","cse");//creating cookie object  
response.addCookie(c);//adding cookie in the response
```

Statements to get cookies :

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

UNIT-IV
Assignment-Cum-Tutorial Questions
SECTION-A

Objective Questions

1. Which html tag can be used to send the request to servlet? []
A. <action> B. <form> C. <input> D. <submit>
2. Which interface contain servlet life-cycle methods? []
A. HttpServlet C. GenericServlet
B. ServletRequest D. Servlet
3. When init() method of servlet gets called? []
A. The init() method is called when the servlet is first created.
B. The init() method is called whenever the servlet is invoked.
C. Both of the above.
D. None of the above.
4. The init parameter name and value pairs that are defined in web.xml file are handled by []
A. ServletConfig object C. ServletContext object
B. ServletRequest object D. ServletResponse object
5. When init parameters are read by container []
A. When Container starts C. When doGet/doPost is called
B. When Servlet is initialized D. When constructor called
6. What type of servlets use these methods doGet(), doPost(),doHead, doDelete(), doTrace)? []
A. GenericServlet B. HttpServlet C.Both A & B D.None
7. Which of the following way can be used to keep track of previous client request
A. Using cookies C. Using hidden form fields []
B. Using URL rewriting D. All of the above
8. Which of the following code is used to get session in servlet? []
A. new Session() C. response.getSession()
B. request.getSession() D. None of the above

9. The life cycle of a servlet is managed by []
A) servlet context B) servlet container
C) the supporting protocol D) All of the above
10. Which method is used to specify before any lines that uses the PrintWriter?
A) setPageType() B) setContextType() []
C) setContentType() D) setResponseType()
11. Where does the Web Application 'Deployment Descriptor' for application be placed as per the Servlet API Specification []
A) /WEB-INF/wdd.xml B) /WEB-INF/web.xml
C) /WEB-INF/lib/web.xml D) /WEB-INF/classes/web.xml
12. To get the servlet environment information _____object is used.[]
A) ServletConfig object is used B) ServletException object is used
C) ServletContext object is used D) ServletContainer object is used
13. When doPost() method of servlet gets called? []
A) A POST request results from an HTML form that specifically lists POST as the method
B) The service() method checks the HTTP request type as POST and calls doPost() method
C) Both of the above D) None of the above
14. Which of the following code can be used to redirect user to different url location []
A) request.sendRedirect(location) B) response.sendRedirect(location)

C) header.sendRedirect(location) D) None of the above
15. Which of the following can be used to add specified cookie to the response and can be used to write a cookie
[]
A) request.addCookie(cookie) B) Header.addCookie(cookie)
C) response.addCookie(cookie) D) Session.addCookie(cookie)

16. The `getSession()` method with 'true' as its parameter will return the appropriate session object when []
- A) the session is completed
 - B) the session does not exist
 - C) the session object is passed to another method
 - D) the session is existing

SECTION-B

SUBJECTIVE QUESTIONS

1. Define Servlet? Explain Servlet Life Cycle Mechanism.
2. Explain with an example how to create and compile the Servlet source code.
3. Explain the importance of Deployment Descriptor(web.xml) in Servlet web application.
4. Explain the functionality of `javax.servlet` package for web servers. Discuss about various classes and interfaces of the package.
5. Explain the functionality of `javax.servlet.http` package for web servers. Discuss about various classes and interfaces of the package.
6. Explain with an example how to read Initialization Parameters stored in web.xml file using ServletConfig Interface.
7. Discuss the use of cookie. Develop and explain a servlet that illustrates the use of cookies.
8. What is the importance of session tracking? Explain how sessions are created with an example?
9. Design a simple servlet that displays a message "HELLO WORLD!" in bold type in the browser display area.
10. A). Create a Servlet that displays the current date and time
B). Write a Servlet that handles HTTP POST request
11. Write a Servlet that greets the user by name. Accept the username through a HTML form
12. Design a Servlet that performs simple page redirection to another location (use `response.sendRedirect(newURL)`)
13. Create a Login form (userid & password) using HTML.

- a. Read the values entered by the user
 - b. Authenticate the user with the values available in web.xml file as Init-paramerters
14. Create a Login form (userid & password) using HTML.
 - a. Read the values entered by the user & store them in Cookie
 - b. Authenticate the user with the values available in Cookies
15. Implement a simple page hit counter using HttpSession

Web Technologies
UNIT – 5
JSP (Java Server Pages)

SYLLABUS:

The Problem with Servlet.
The Anatomy of a JSP Page,
Generating Dynamic Content -Using Scripting Elements, Directives.
Implicit JSP Objects,
Declaring Variables and Methods,
Passing Control and Data between Pages,
Sharing Session and Application Data.

Objectives:

- To develop real time web applications.
- To get acquainted with skills for creating websites and web apps through learning various technologies like HTML, CSS, JavaScript, XML and Servlets and JSP,

Learning Outcomes:

Students will be able to

- Know the problems with servlet
- Learn the Anatomy of JSP page
- Know how to generate dynamic content using scripting elements.
- Identify Implicit JSP Objects, variables and methods declaration.
- Know how to share session and application data.

Problems with Servlets

- Servlets need a special "servlet container" to run servlets.
- Servlets need a Java Runtime Environment on the server to run servlets.
- For developing Servlet based application, knowledge of java as well as HTML code is necessary.

- The servlet has to do various tasks such as acceptance of request, processing of request, handling of business logic and generation of response.
- In many Java servlet-based applications, processing the request and generating the response are both handled by a single servlet class.

An example servlet looks like this:

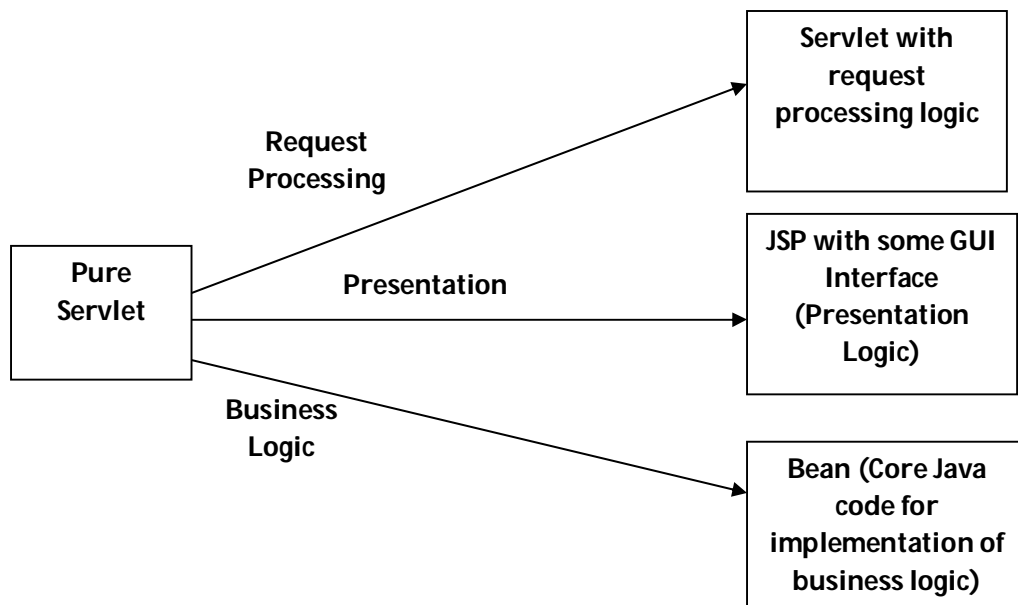
```
public class OrderServlet extends HttpServlet
{
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter( );
        if (isOrderInfoValid(request)) {
            saveOrderInfo(request);
            out.println("<html>");
            out.println(" <head>");
            out.println(" <title>Order Confirmation</title>");
            out.println(" </head>");
            out.println(" <body>");
            out.println(" <h1>Order Confirmation</h1>");
            renderOrderInfo(request);
            out.println(" </body>");
            out.println("</html>");
        }
    }
}
```

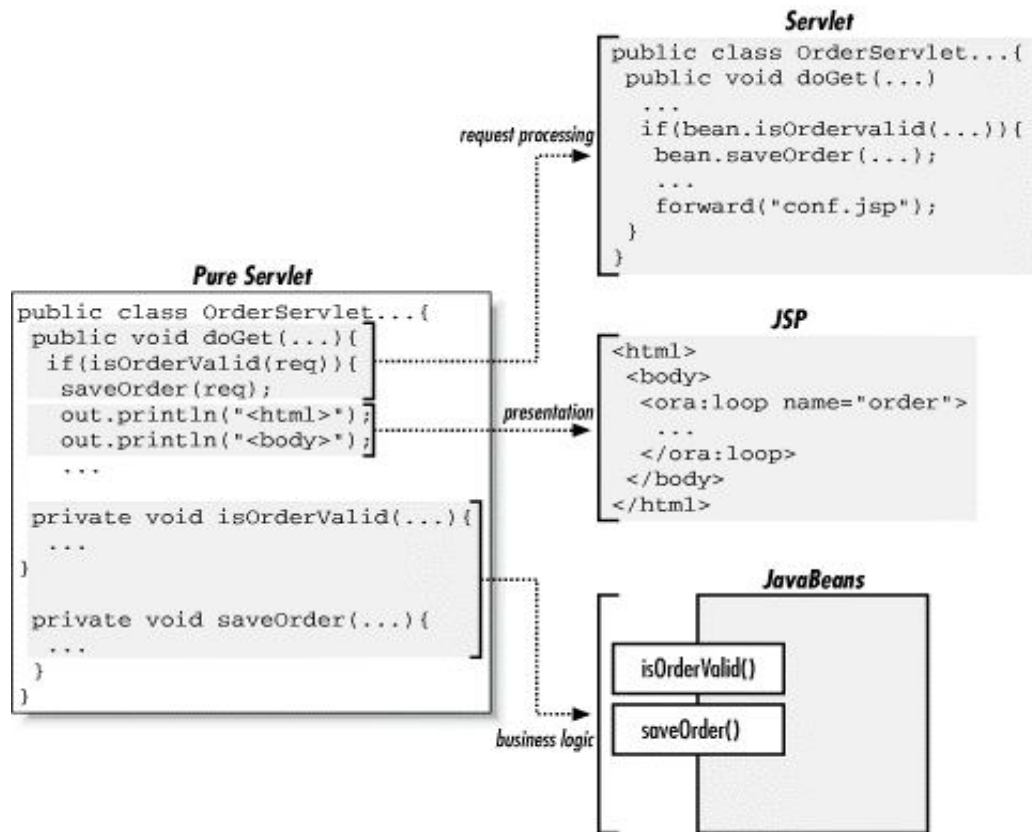
The pure servlet-based approach still has a few problems:

- Detailed Java programming knowledge is needed to develop and maintain all aspects of the application, since the processing code and the HTML elements are lumped together.
- Changing the look and feel of the application, or adding support for a new type of client (such as a WML client), requires the servlet code to be updated and recompiled.
- 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 that is time-consuming and error-prone

JSP lets you solve these problems “by separating the request processing and business logic code from the presentation”, as illustrated in Figure 1.1. Instead of embedding HTML in the code, you place all static HTML in JSP pages, 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 servlet programmers, and the business logic can be handled by JavaBeans and Enterprise JavaBeans (EJB) components.

Figure 1.1. Separation of request processing, business logic, and presentation





Advantages of JSP:

- Separating the request processing and business logic from presentation makes it possible to divide the development tasks among people with different skills. Java programmers implement the request processing and business logic pieces, web page authors implement the user interface, and both groups can use best-of-breed development tools for the task at hand. The result is a *much more productive development process*.
- It also makes it *possible to change different aspects of the application independently*, such as changing the business rules without touching the user interface.
- This model has clear benefits even for a web page author *without programming skills* who is working alone. A page author can develop web applications with many dynamic features, using generic Java components provided by open source projects or commercial companies.
- It provides a very powerful and flexible mechanism to produce dynamic web pages.
- Dynamic contents can be handled using JSP because JSP allows scripting and element based programming.

- JSP allows creating and using our own custom tag libraries. Hence any application specific requirements can be satisfied using custom tag libraries. This helps the developer to develop any kind of application.
- JSP is an essential component of J2EE. Hence using JSP it is possible to develop simple as well as complex applications.
- In JSP we can directly embed java code into html code but in servlet is not possible.
- JSP page is automatically compiled but servlet will manually redeploy.
- In jsp implicit objects are presents which is we can implement directly into jsp pages but in servlet there are no implicit objects.

Introduction to JSP

JSP Overview

- JavaServer Pages (JSP) is a server-side programming technology that enables the creation of dynamic, platform-independent method for building Web-based applications.
- JavaServer Pages (JSP) is a technology for developing web pages that support dynamic content which helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with <% and end with %>.
- JSP is a specification and not a product.Hence developers can develop variety of applications and add up to performance and quality of software products.It is essential component of J2EE.
- A JavaServer Pages component is a type of Java servlet that is designed to fulfill the role of a user interface for a Java web application. Web developers write JSPs as text files that combine HTML or XHTML code, XML elements, and embedded JSP actions and commands.
- Using JSP, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.
- JSP tags can be used for a variety of purposes, such as retrieving information from a database or registering user preferences, accessing JavaBeans components, passing control between pages and sharing information between requests, pages etc.

Why Use JSP?

- Performance is significantly better because JSP allows embedding Dynamic Elements in HTML Pages itself.

- JSP allows to separate the presentation logic and business logic
- JSP are always compiled before it's processed by the server
- JavaServer Pages are built on top of the Java Servlets API, so like Servlets, JSP also has access to all the powerful Enterprise Java APIs, including JDBC, EJB, JAXP etc.
- JSP pages can be used in combination with servlets that handle the business logic, the model supported by Java servlet template engines.

Anatomy of a JSP page

JSP page is simply a regular web page with JSP elements for generating the parts of the page that differ for each request.

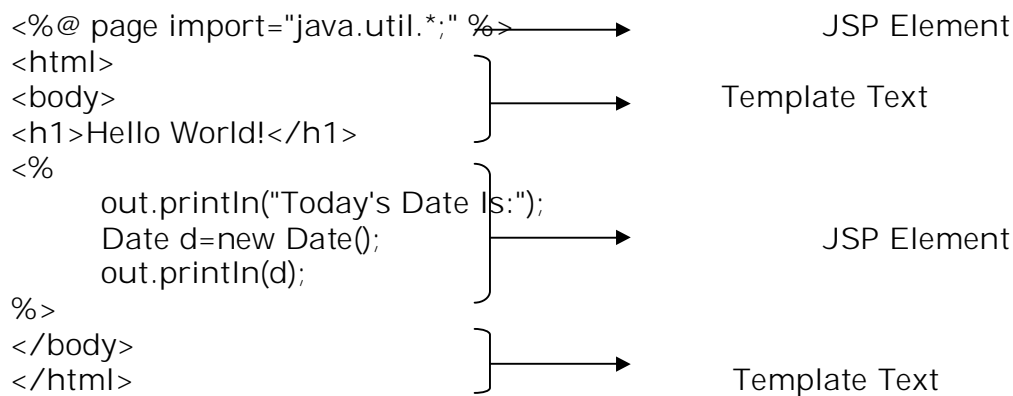
The JSP Page consists of 2 parts:

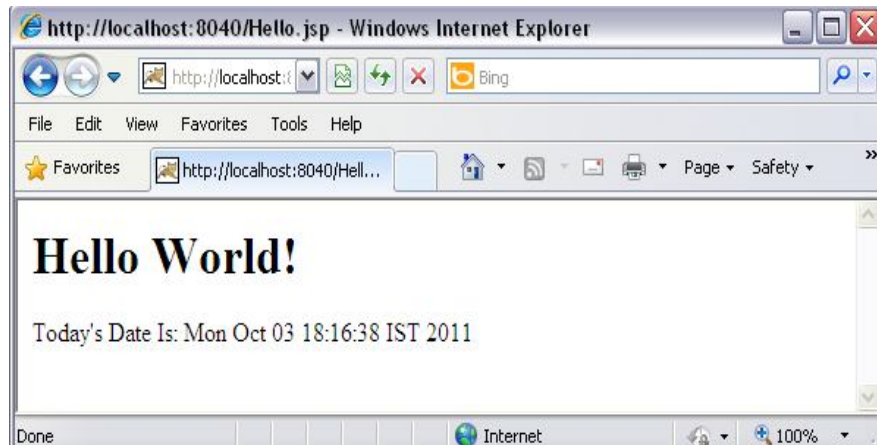
- Template text
- JSP Elements

i) Template Text:

Everything in the page that is not a JSP element is called *template text*. Template text can really be any text: HTML, WML, XML, or even plain text. Template text is always passed straight through to the browser.

Figure 1.2. Template text and JSP elements



OUTPUT:

When a JSP page request is processed, the template text and the dynamic content generated by the JSP elements are merged, and the result is sent as the response to the browser.

ii) JSP Elements:

There are three types of elements with Java Server Pages:

Directive,
Action, and
Scripting Elements.

(Additional Elements are **JSP Implicit Objects**)

The Directive elements, shown in [Table 1.1](#), are used to specify information about the page itself that remains the same between page requests, for example, the scripting language used in the page, whether session tracking is required, and the name of a page that should be used to report errors, if any.

Table 1.1 Directive elements

Element	Description
<% page ... %>	Defines page-dependent attributes, such as scripting language, error page, and buffering requirements
<% include ... %>	Includes a file during the translation phase
<% taglib ... %>	Declares a tag library, containing custom actions, used in the page

Action elements typically perform some action based on information that is required at the exact time the JSP page is requested by a client. An action element can, for instance, access parameters sent with the request to do a database lookup. It can also dynamically generate HTML, such as a table filled

with information retrieved from an external system. The JSP specification defines a few standard action elements, listed in Table 1.2, and includes a framework for developing custom action elements. A custom action element can be developed by a programmer to extend the JSP language.

Table 1.2 Action Elements

Element	Description
<code><jsp:useBean></code>	Makes a JavaBeans component available in a page
<code><jsp:getProperty></code>	Gets a property value from a JavaBeans component and adds it to the response
<code><jsp:setProperty></code>	Sets a JavaBeans property value
<code><jsp:include></code>	Includes the response from a servlet or JSP page during the request processing phase
<code><jsp:forward></code>	Forwards the processing of a request to a servlet or JSP page
<code><jsp:param></code>	Adds a parameter value to a request handed off to another servlet or JSP page using <code><jsp:include></code> or <code><jsp:forward></code>
<code><jsp:plugin></code>	Generates HTML that contains the appropriate client browser-dependent elements (OBJECT or EMBED) needed to execute an Applet with the Java Plugin software

Scripting elements, shown in Table 1.3, allow you to add small pieces of code to a JSP page, such as an if statement to generate different HTML depending on a certain condition. Like actions, they are also executed when the page is requested. Scripting elements must be used with extreme care: if you embed too much code in your JSP pages, you will end up with the same kind of maintenance problems as with servlets embedding HTML.

1. Scriptlets
2. Expressions
3. Declarations

Table 1.3 Scripting elements

Element	Description
<code><% ... %></code>	Scriptlet, used to embed scripting code.
<code><%= ... %></code>	Expression, used to embed Java expressions when the result shall be added to the response. Also used as runtime action attribute values.
<code><%! ... %></code>	Declaration, used to declare instance variables and methods in the JSP page implementation class.

JSP Processing/ Life Cycle of JSP:

A JSP page cannot be sent as-is to the browser; all JSP elements must first be processed by the server. This is done by turning the JSP page into a servlet, and then executing the servlet.

JSP Container:

Just as a web server needs a servlet container to provide an interface to servlets, the server needs a JSP container to process JSP pages. The JSP container is often implemented as a servlet configured to handle all requests for JSP pages. In fact, these two containers - a servlet container and a JSP container - are often combined into one package under the name *web container*.

JSP Processing is done in 2 phases:

- i) Translation Phase
- ii) Request Processing Phase

i) Translation Phase:

A JSP container is responsible for *converting the JSP page into a servlet* (known as the *JSP page implementation class*) and *compiling the servlet*. These two steps form the *translation phase*. The JSP container automatically initiates the translation phase for a page when the first request for the page is received. The translation phase can also be initiated explicitly; this is referred to as *precompilation* of a JSP page.

When a JSP container receives a jsp request, it checks for the jsp's servlet instance. If no servlet instance is available, then, the container creates the servlet instance using following stages.

- Translation
- Compilation
- Loading
- Instantiation
- Initialization

Translation - In this step the JSP page is translated into the corresponding Servlet.

Compilation - Once the JSP page has been translated into the corresponding Servlet, the next obvious step is to compile that Servlet.

Loading & Instantiation - As is the case with any compiled class (.class file), this servlet class also needs to be loaded into the memory before being used. The default class loader of the Container will load this class. Once the class is loaded, an instance of this class gets created.

Initialization: JspPage interface contains the jspInit() method, which is used by the JSP container to initialize the newly created instance. This jspInit() method is just like the init() method of the Servlet and it's called only once during the entire life cycle of a JSP/Servlet.

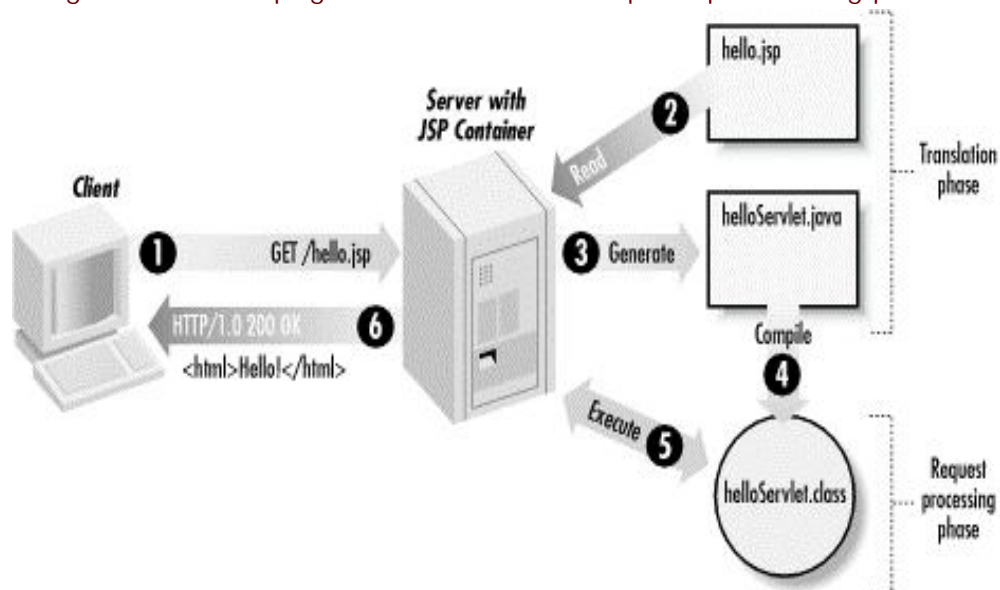
ii) Request Processing Phase:

The JSP container is also responsible for invoking the JSP page implementation class to process each request and generate the response. This is called the *request processing phase*.

`_jspService()` is the method which is called every time the JSP is requested to serve a request. This method normally executes in a separate thread of execution and the main JSP thread keeps waiting for other incoming requests. Every time a request arrives, the main JSP thread spawns a new thread and passes the request (incoming request) and response (new) objects to the `_jspService()` method which gets executed in the newly spawned thread.

The two phases are illustrated in Figure 1.3.

Figure 1.3. JSP page translation and request processing phases



As long as the JSP page remains unchanged, any subsequent processing goes straight to the request processing phase (i.e., it simply executes the class file).

When the JSP page is modified, it goes through the translation phase again before entering the request processing phase. So in a way, a JSP page is really just another way to write a servlet without having to be a Java programming expert.

And, except for the translation phase, a JSP page is handled exactly like a regular servlet: it's loaded once and called repeatedly, until the server is shut down.

By virtue of being an automatically generated servlet, a JSP page inherits all of the advantages of servlets: platform and vendor independence, integration, efficiency, scalability, robustness, and security.

Life Cycle Methods:

- i). The `jspInit()`- The container calls the `jspInit()` to initialize the servlet instance. It is called before any other method, and is called only once for a servlet instance.

```
public void jspInit(){
    // Initialization code...
}
```

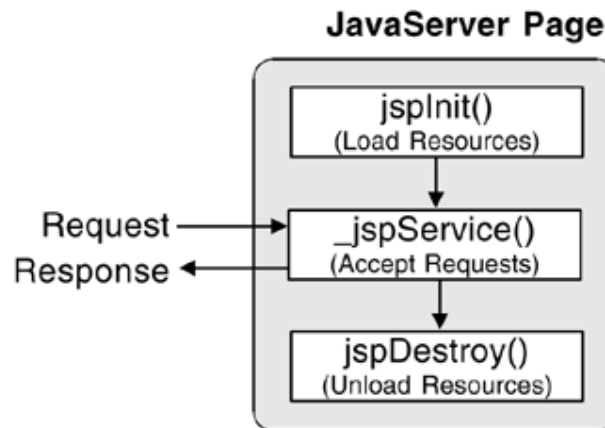
- ii). The `_jspService()`- The container calls the `_jspService()` for each request, passing it the request and the response objects.

```
void _jspService(HttpServletRequest request,
    HttpServletResponse response)
{
    // Service handling code...
}
```

- iii). The `jspDestroy()`- The container calls this when it decides to take the instance out of service. It is the last method called in the servlet instance.

```
public void jspDestroy()
{
    // cleanup code goes here.
}
```


Figure 1.4 Life Cycle Methods of JSP



JSP Application Design with MVC:

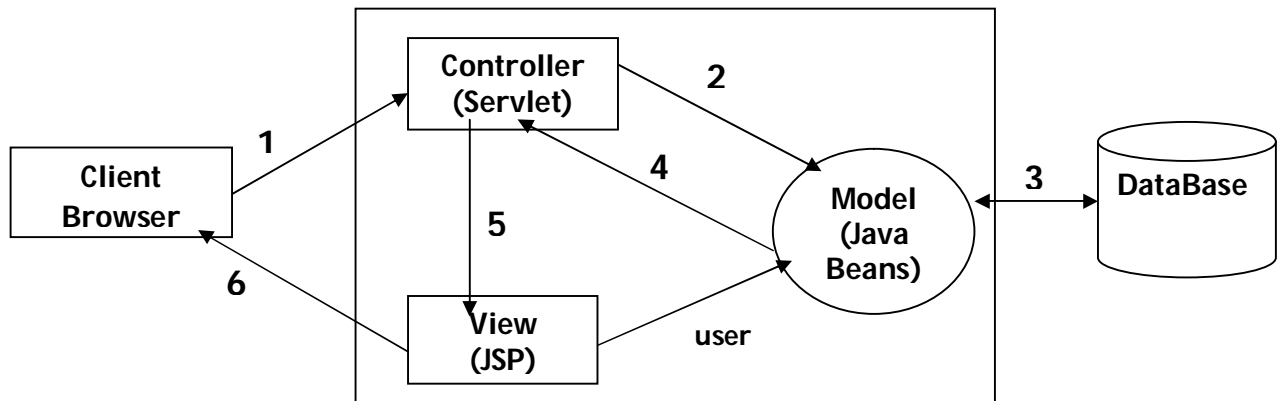
Model View Controller is a design model suitable for both simple and complex applications.

The MVC model was first described by Xerox in a number of papers published in the late 1980s in conjunction with the Smalltalk language. But this model has since been used for GUI applications developed in all popular programming languages. The basic idea is to separate the application data and business logic, the presentation of the data, and the interaction with the data into distinct entities labeled the Model, the View, and the Controller, respectively.

In a server application, we commonly classify the parts of the application as: business logic, presentation, and request processing together. The key point of using MVC is to separate components into three distinct units: the Model, the View, and the Controller.

- 1) **MODEL:** In MVC terms, the Model corresponds to business logic and data, *Business logic* is the term used for the manipulation of an application's data, i.e., customer, product, and order information. An application data structure and logic (the Model) is typically the most stable part of an application, while the presentation of that data (the View) changes fairly often.
- 2) **VIEW:** The View corresponds to the presentation logic; *Presentation* refers to how the application is displayed to the user, i.e., the position, font, and size. It may have little or no programming logic at all.
- 3) **CONTROLLER:** The Controller corresponds to the request processing. And finally, *request processing* is what ties the business logic and presentation parts, so as to perform some operation.

Figure 1.5 MVC Architecture



1. Request
2. Invoke Business Logic
3. Retrieve/ store data
4. Get the result and set it in appropriate scope
5. Forward to view
6. Response

Advantages:

- This design makes a flexible design, where multiple presentations (Views) can be provided and easily modified,
- Changes in the business rules or physical representation of the data (the Model) can be made without touching any of the user interface code.

Generating Dynamic Content:

- JSP is all about generating dynamic content: content that differs based on user input, time of day, the state of an external system, or any other runtime conditions.
- JSP provides you with lots of tools for generating this content.
- Dynamic content can be generated using all JSP Elements

**standard actions,
custom actions,
JavaBeans, and
scripting elements
Directives**

JSP Page Showing the Current Date and Time (date.jsp)

```
<%@ page language="java" contentType="text/html" %>
<html>
  <body bgcolor="white">

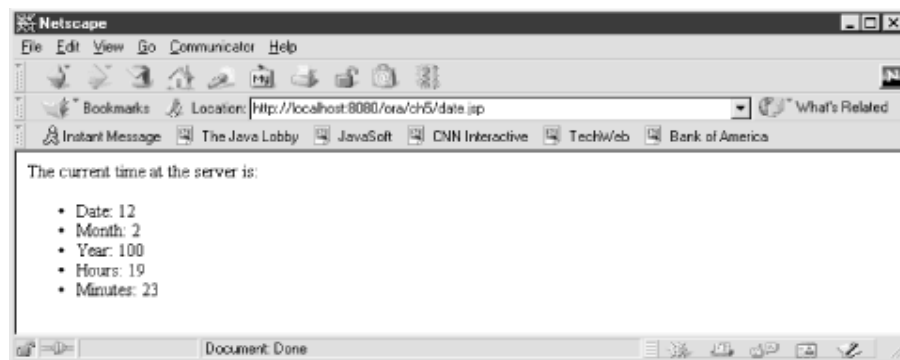
    <jsp:useBean id="clock" class="java.util.Date" />

    The current time at the server is:
    <ul>
      <li>Date: <jsp:getProperty name="clock" property="date" />
      <li>Month: <jsp:getProperty name="clock" property="month" />
      <li>Year: <jsp:getProperty name="clock" property="year" />
      <li>Hours: <jsp:getProperty name="clock" property="hours" />
      <li>Minutes: <jsp:getProperty name="clock" property="minutes" />
    </ul>

  </body>
</html>
```

The *date.jsp* page displays the current date and time.

Output of *date.jsp* example



Using Scripting Elements ,Directive, Action, Scriptlet:

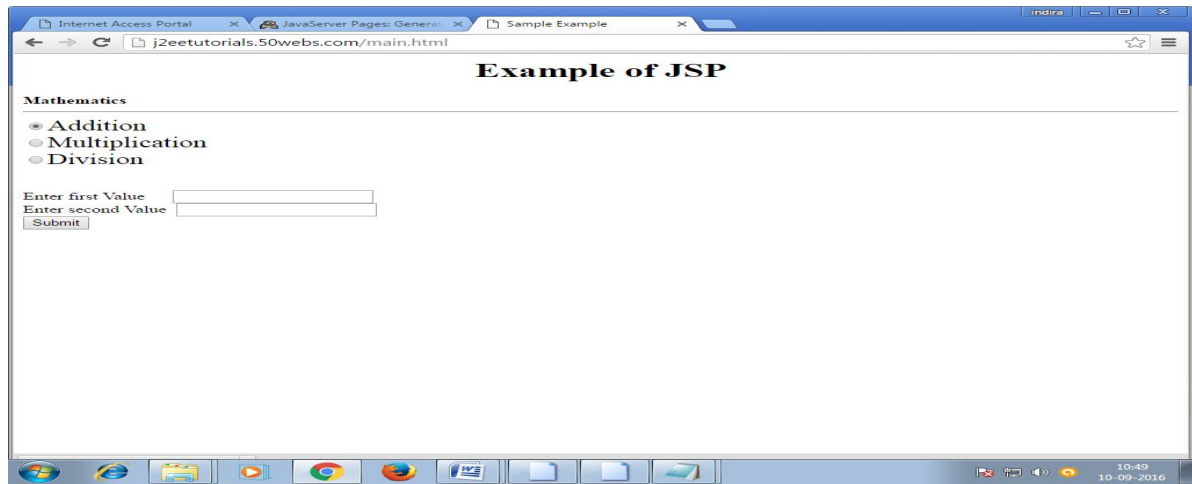
Using JSP Directives

- Directives are used to specify attributes of the page itself, primarily those that affect how the page is converted into a Java servlet.
- There are three JSP directives:

[page](#), [include](#), and [taglib](#).

- JSP pages typically start with a [page](#) directive that specifies the scripting

Output:



JSP Scripting Elements

- Expressions
 - Format: `<%= expression %>`
- Scriptlets
 - Format: `<% code %>`
- Declarations
 - Format: `<%! code %>`

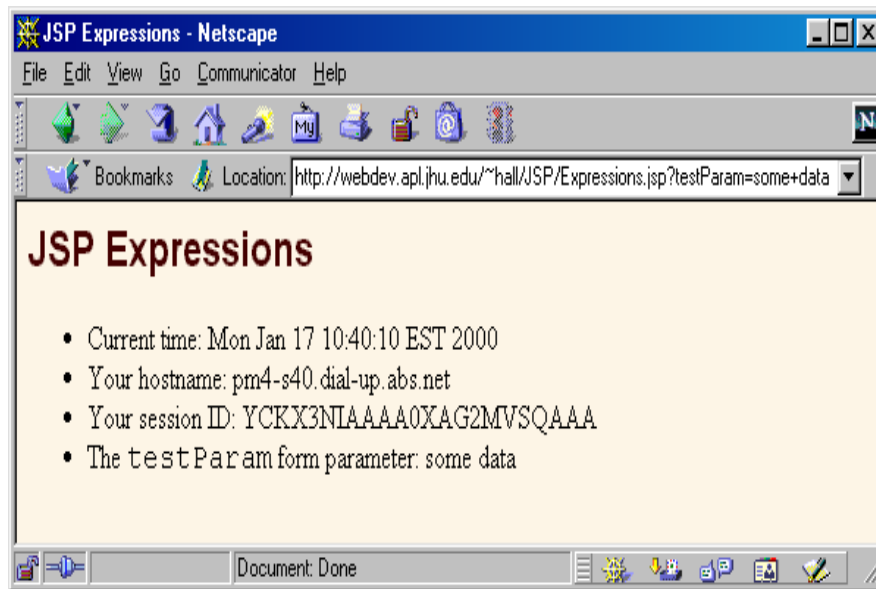
JSP Expressions

- Format
 - `<%= Java Expression %>`
- Result
 - expression placed in `_jspService` inside `out.print`
- Examples
 - Current time: `<%= new java.util.Date() %>`
 - `<%= request.getRemoteHost() %>`
- XML-compatible syntax
 - `<jsp:expression>Java Expression</jsp:expression>`

Example:

```
<BODY>
<H2>JSP Expressions</H2>
<UL>
  <LI>
    Current time: <%= new java.util.Date() %>
  <LI>
    Your hostname: <%= request.getRemoteHost() %>
  <LI>
    Your session ID: <%= session.getId() %>
  <LI>
    The <CODE>testParam</CODE> form parameter:
    <%= request.getParameter("testParam") %>
</UL>
</BODY>
```

Output:



Predefined Variables

- request
 - The HttpServletRequest (1st argument to service/doGet)
- response
 - The HttpServletResponse (2nd arg to service/doGet)
- out
 - The Writer (a buffered version of type JspWriter) used to send output to the client
- session

- The HttpSession associated with the request (unless disabled with the session attribute of the page directive)
- application
 - The ServletContext (for sharing data) as obtained via `getServletContext()`.

JSP Scriptlets

- Format
 - `<% Java Code %>`
- Result
 - Code is inserted verbatim into servlet's `_jspService`
- Example
 - `<%`
String queryData = request.getQueryString();
out.println("Attached GET data: " + queryData);
`%>`
 - `<% response.setContentType("text/plain"); %>`
- XML-compatible syntax
`<jsp:scriptlet>Java Code</jsp:scriptlet>`

JSP Declarations

- Format
 - `<%! Java Code %>`
- Result
 - Code is inserted verbatim into servlet's class definition, outside of any existing methods
- Examples
 - `<%! private int someField = 5; %>`
 - `<%! private void someMethod(...) {...} %>`
- XML-compatible syntax

- `<jsp:declaration>Java Code</jsp:declaration>`

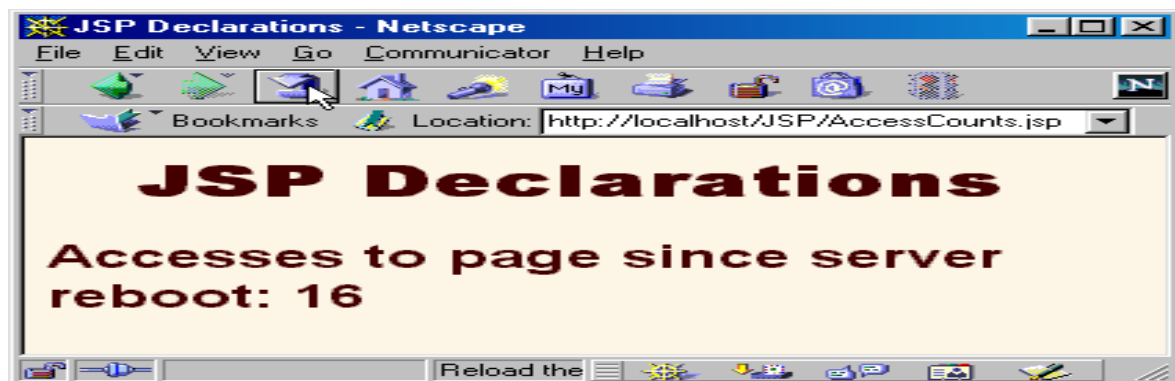
Example:

```
<HTML><HEAD><TITLE>JSP Declarations</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>

<BODY>
<H1>JSP Declarations</H1>

<%= private int accessCount = 0; %>
<H2>Accesses to page since server reboot:
<%= ++accessCount %></H2>

</BODY>
</HTML>
```

Output:

Declaring Variables and Methods

The **JSP declaration tag** is used *to declare fields and methods*.

The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet.

So it doesn't get memory at each request.

Syntax of JSP declaration tag

The syntax of the declaration tag is as follows:

1. `<%! field or method declaration %>`

Difference between JSP Scriptlet tag and Declaration tag

Jsp Scriptlet Tag	Jsp Declaration Tag
The jsp scriptlet tag can only declare variables not methods.	The jsp declaration tag can declare variables as well as methods.
The declaration of scriptlet tag is placed inside the <code>_jspService()</code> method.	The declaration of jsp declaration tag is placed outside the <code>_jspService()</code> method.

Example of JSP declaration tag that declares field

In this example of JSP declaration tag, we are declaring the field and printing the value of the declared field using the jsp expression tag.

index.jsp

1. `<html>`
 2. `<body>`
 3. `<%! int data=50; %>`
 4. `<%= "Value of the variable is:"+data %>`
 5. `</body>`
 6. `</html>`
-

Example of JSP declaration tag that declares method

In this example of JSP declaration tag, we are defining the method which returns the cube of given number and calling this method from the jsp expression tag. But we can also use jsp scriptlet tag to call the declared method.

index.jsp

1. `<html>`
2. `<body>`
3. `<%!`
4. `int cube(int n){`
5. `return n*n*n*;`
6. `}`
7. `%>`
8. `<%= "Cube of 3 is:"+cube(3) %>`
9. `</body>`
10. `</html>`

Implicit JSP Objects:

JSP Implicit Objects are the Java objects that the JSP Container makes available to developers in each page and developer can call them directly without being explicitly declared.

JSP Implicit Objects are also called pre-defined variables.

JSP supports nine Implicit Objects which are listed below:

Object	Description
Request	This is the HttpServletRequest object associated with the request.
Response	This is the HttpServletResponse object associated with the response to the client.
Out	This is the PrintWriter object used to send output to the client.
Session	This is the HttpSession object associated with the request.
Application	This is the ServletContext object associated with application context.
Config	This is the ServletConfig object associated with the page.
pageContext	This encapsulates use of server-specific features like higher performance JspWriters .
Page	This is simply a synonym for this , and is used to call the methods defined by the translated servlet class.

Exception	The Exception object allows the exception data to be accessed by designated JSP.
-----------	---

The request Object:

The request object is an instance of a `javax.servlet.http.HttpServletRequest` object. Each time a client requests a page the JSP engine creates a new object to represent that request.

The request object provides methods to get HTTP header information including form data, cookies, HTTP methods etc.

The response Object:

The response object is an instance of a `javax.servlet.http.HttpServletResponse` object. Just as the server creates the request object, it also creates an object to represent the response to the client.

The response object also defines the interfaces that deal with creating new HTTP headers. Through this object the JSP programmer can add new cookies or date stamps, HTTP status codes etc.

The out Object:

The out implicit object is an instance of a `javax.servlet.jsp.JspWriter` object and is used to send content in a response.

The initial `JspWriter` object is instantiated differently depending on whether the page is buffered or not. Buffering can be easily turned off by using the `buffered='false'` attribute of the page directive.

The `JspWriter` object contains most of the same methods as the `java.io.PrintWriter` class. However, `JspWriter` has some additional methods designed to deal with buffering. Unlike the `PrintWriter` object, `JspWriter` throws `IOExceptions`.

Following are the important methods which we would use to write boolean, char, int, double, object, String etc.

Method	Description
<code>out.print(dataType dt)</code>	Print a data type value
<code>out.println(dataType dt)</code>	Print a data type value then terminate the line with new line character.
<code>out.flush()</code>	Flush the stream.

The session Object:

The session object is an instance of `javax.servlet.http.HttpSession` and behaves exactly the same way that session objects behave under Java Servlets.

The session object is used to track client session between client requests.

The application Object:

The application object is direct wrapper around the `ServletContext` object for the generated Servlet and in reality an instance of a `javax.servlet.ServletContext` object.

This object is a representation of the JSP page through its entire lifecycle. This object is created when the JSP page is initialized and will be removed when the JSP page is removed by the `jspDestroy()` method.

By adding an attribute to application, you can ensure that all JSP files that make up your web application have access to it.

The config Object:

The config object is an instantiation of `javax.servlet.ServletConfig` and is a direct wrapper around the `ServletConfig` object for the generated servlet.

This object allows the JSP programmer access to the Servlet or JSP engine initialization parameters such as the paths or file locations etc.

The following config method is the only one you might ever use, and its usage is trivial:

```
config.getServletName();
```

This returns the servlet name, which is the string contained in the <servlet-name> element defined in the WEB-INF\web.xml file

The pageContext Object:

The pageContext object is an instance of a javax.servlet.jsp.PageContext object. The pageContext object is used to represent the entire JSP page.

This object is intended as a means to access information about the page while avoiding most of the implementation details.

This object stores references to the request and response objects for each request. The application, config, session, and out objects are derived by accessing attributes of this object.

The pageContext object also contains information about the directives issued to the JSP page, including the buffering information, the errorPageURL, and page scope.

The PageContext class defines several fields, including PAGE_SCOPE, REQUEST_SCOPE, SESSION_SCOPE, and APPLICATION_SCOPE, which identify the four scopes. It also supports more than 40 methods, about half of which are inherited from the javax.servlet.jsp.JspContext class.

One of the important methods is **removeAttribute**, which accepts either one or two arguments. For example, pageContext.removeAttribute ("attrName") removes the attribute from all scopes, while the following code only removes it from the page scope:

```
pageContext.removeAttribute("attrName", PAGE_SCOPE);
```

The page Object:

This object is an actual reference to the instance of the page. It can be thought of as an object that represents the entire JSP page.

The page object is really a direct synonym for the **this** object.

The exception Object:

The exception object is a wrapper containing the exception thrown from the previous page. It is typically used to generate an appropriate response to the error condition.

Passing Control and Data between JSP pages, Requests:

- Sometimes, we need to handover the control to another page, with the necessary data passed to it.

Passing Control & Data

- Sometimes, a jsp page wants to pass the control to another server-side program for further processing.
- This is done by using `<jsp:forward>` action element.

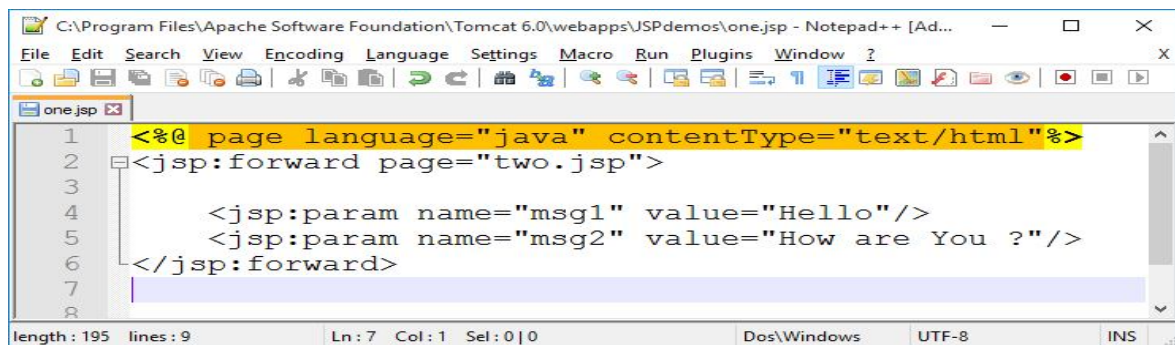
```
<jsp:forward  
page="targetPageURL" />
```

- Once the jsp engine encounters the `<jsp:forward>` action, it stops the processing of the current page and starts the processing of the target page specified by the **page** attribute. The rest of the original page is never processed further.
- The **HttpServletRequest** and **HttpServletResponse** objects are passed to the target page. So, the target page can access all information passed to the original page.
- We can pass additional parameters to the target page using the `<jsp:param>` action element along with the `<jsp:forward>`

```
<jsp:param name="parameterName"  
value="parameterValue" />
```

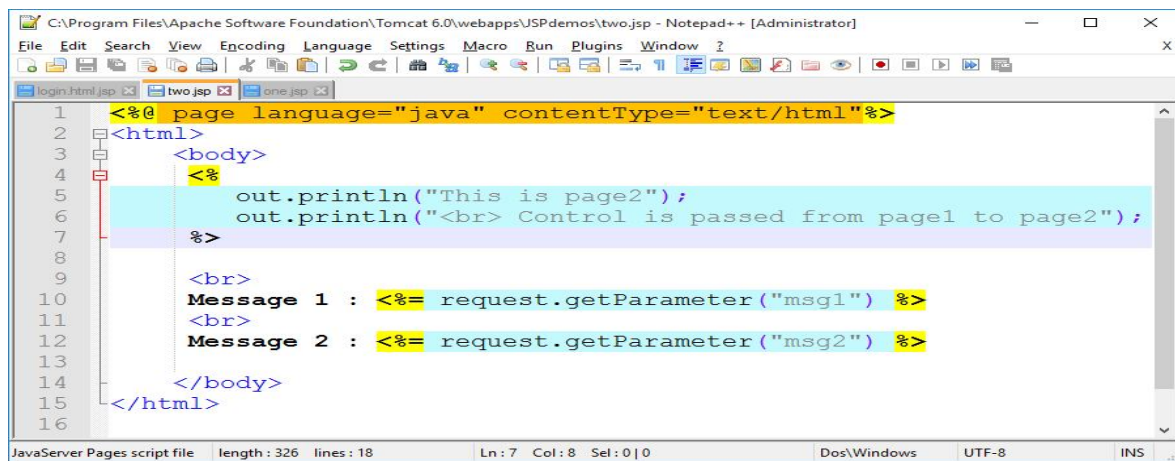
- The target page can access these parameters in the same way as the original parameters using the **getParameter()** method or by using `#{param.parameterName}` .

Example :



```
1 <%@ page language="java" contentType="text/html"%>
2 <jsp:forward page="two.jsp">
3
4     <jsp:param name="msg1" value="Hello"/>
5     <jsp:param name="msg2" value="How are You ?"/>
6 </jsp:forward>
7
8
```

length : 195 lines : 9 Ln : 7 Col : 1 Sel : 0 | 0 Dos\Windows UTF-8 INS

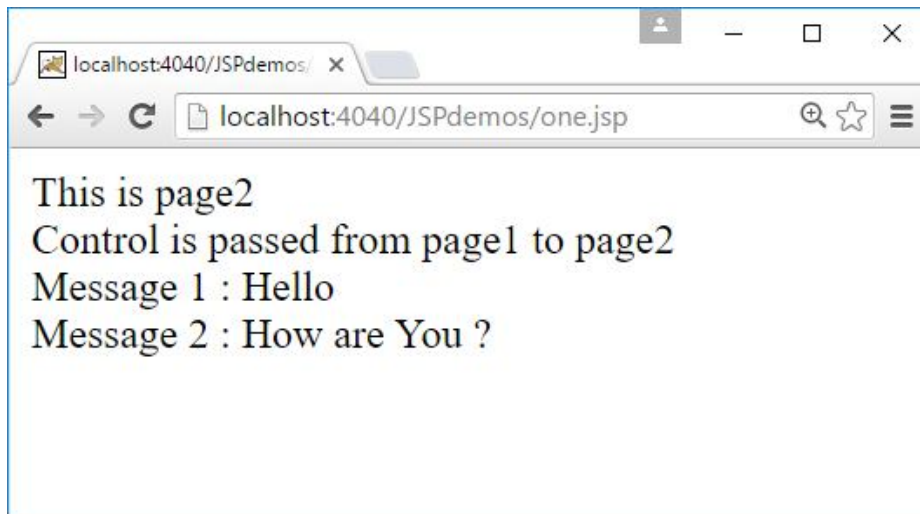


```
1 <%@ page language="java" contentType="text/html"%>
2 <html>
3     <body>
4         <%
5             out.println("This is page2");
6             out.println("<br> Control is passed from page1 to page2");
7         %>
8
9         <br>
10        Message 1 : <%= request.getParameter("msg1") %>
11        <br>
12        Message 2 : <%= request.getParameter("msg2") %>
13
14    </body>
15 </html>
16
```

JavaServer Pages script file length : 326 lines : 18 Ln : 7 Col : 8 Sel : 0 | 0 Dos\Windows UTF-8 INS

Now access the page **one.jsp** through the tomcat server as,

<http://localhost:4040/JSPdemos/one.jsp>



Sharing Session and Application Data

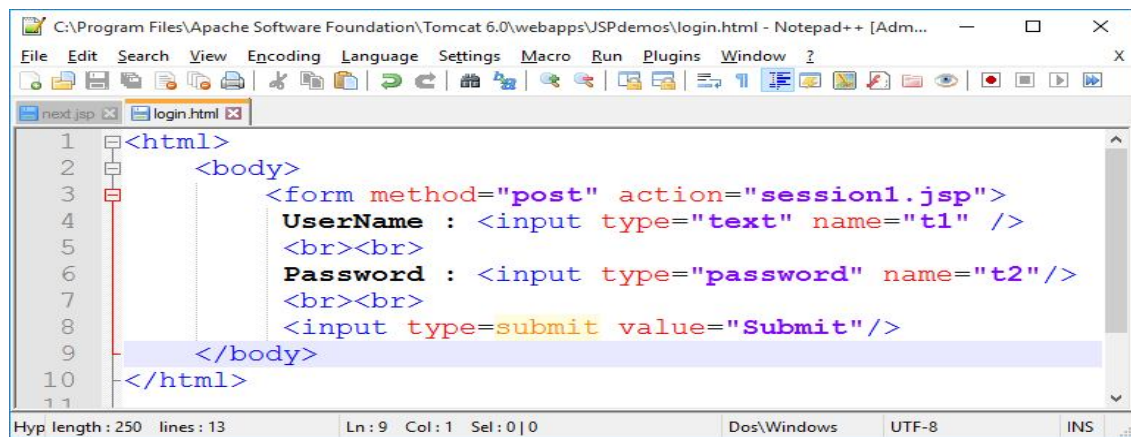
Sharing Session data

- The objects having request scope are available in all pages across a single request. However, sometimes objects should be shared among multiple requests.

Example :

- Online examination system allows different users. Different users have different sets of data such as expiry time, number of questions answered and so on.
- JSP pages requested from the same browser must share the same user name that was provided during the login procedure. This type of information can be shared through session scope.
- The objects having session scope are available to all pages requested by the same browser.
- The implicit object **session** is one such object.
- This **session** object can be used to store and retrieve other objects.
- So, objects may be shared across pages in the same session using this **session** object.

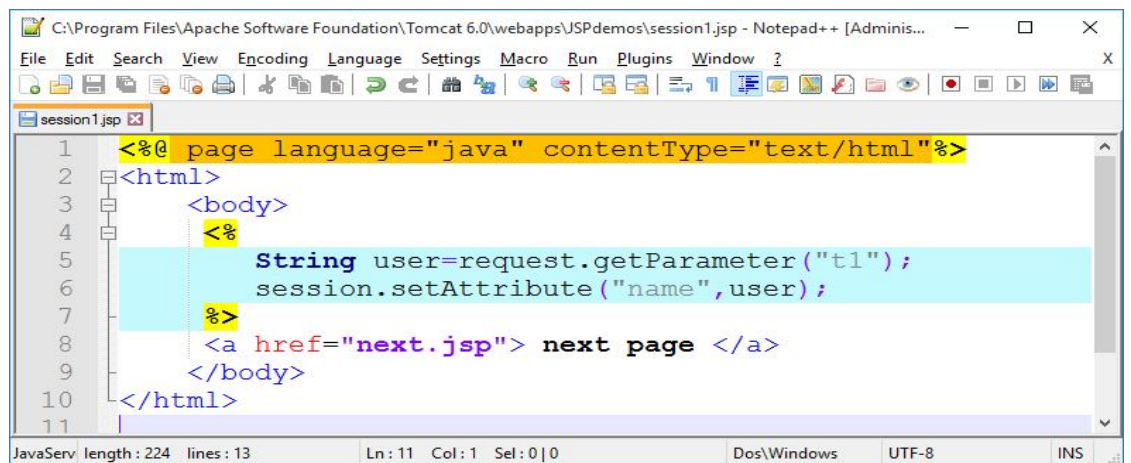
Example program :



```

C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\JSPdemos\login.html - Notepad++ [Adm...
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
next.jsp login.html
1 <html>
2 <body>
3 <form method="post" action="session1.jsp">
4   UserName : <input type="text" name="t1" />
5   <br><br>
6   Password : <input type="password" name="t2"/>
7   <br><br>
8   <input type="submit" value="Submit"/>
9 </body>
10 </html>
11
Hyp length: 250 lines: 13 Ln: 9 Col: 1 Sel: 0|0 Dos\Windows UTF-8 INS

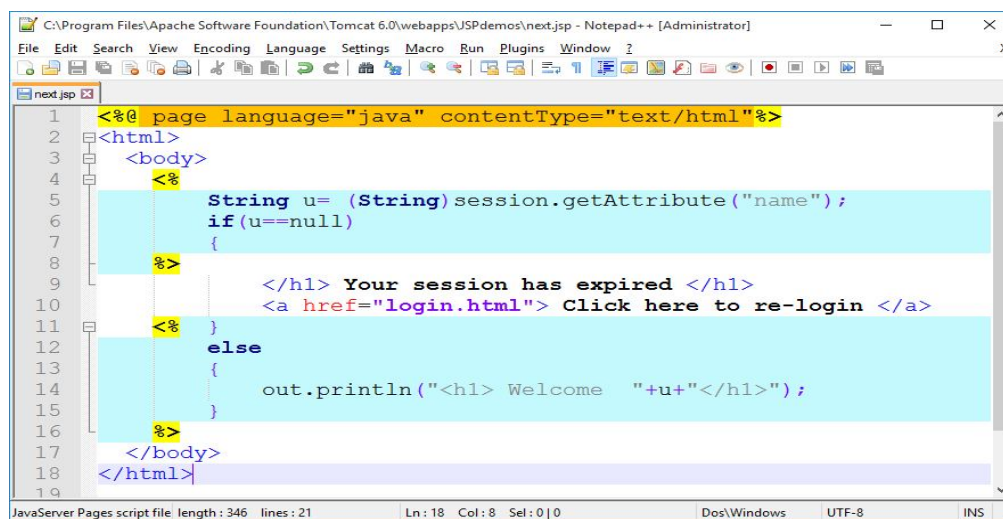
```



```

C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\JSPdemos\session1.jsp - Notepad++ [Adminis...
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
session1.jsp
1 <%@ page language="java" contentType="text/html"%>
2 <html>
3 <body>
4 <%
5   String user=request.getParameter("t1");
6   session.setAttribute("name",user);
7   %>
8   <a href="next.jsp"> next page </a>
9 </body>
10 </html>
11
JavaServ length: 224 lines: 13 Ln: 11 Col: 1 Sel: 0|0 Dos\Windows UTF-8 INS

```

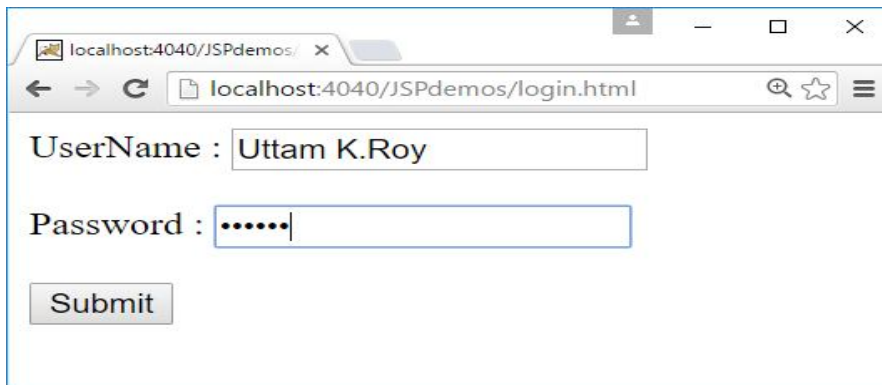


```

C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\JSPdemos\next.jsp - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
next.jsp
1 <%@ page language="java" contentType="text/html"%>
2 <html>
3 <body>
4 <%
5   String u= (String) session.getAttribute("name");
6   if(u==null)
7   {
8   <%
9     </h1> Your session has expired </h1>
10    <a href="login.html"> Click here to re-login </a>
11    %>
12   }
13   else
14   {
15     out.println("<h1> Welcome "+u+"</h1>");
16   }
17 </body>
18 </html>
19
JavaServer Pages script file: length: 346 lines: 21 Ln: 18 Col: 8 Sel: 0|0 Dos\Windows UTF-8 INS

```

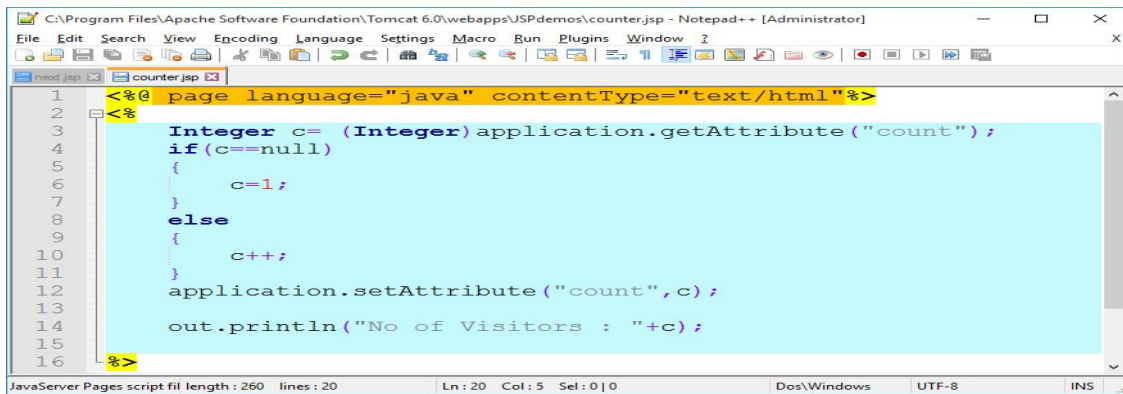
Now, run **login.html** through tomcat server.



Sharing Application Data :

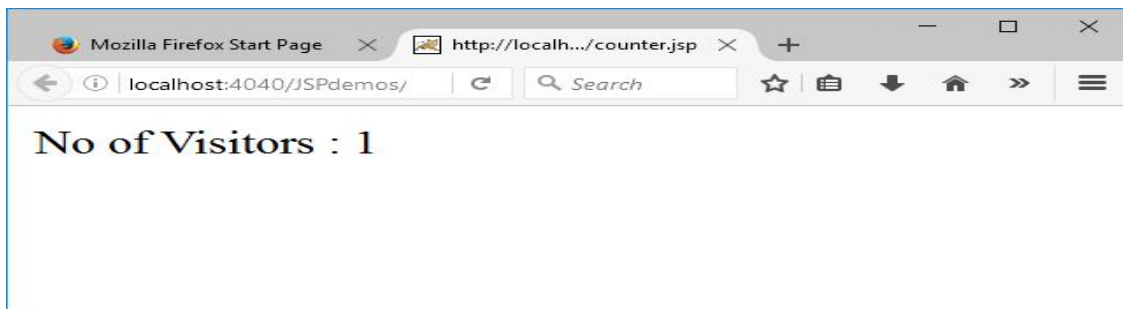
- The objects that have application scope are shared across all pages of an application that are requested through any browser.
- Implicit object used is **application**.

Example: The following example counts the number of visitors to a specific web site. The counter will be incremented for different users accessing the website through different browsers.



```
1 <%@ page language="java" contentType="text/html"%>
2 <%
3     Integer c= (Integer) application.getAttribute("count");
4     if(c==null)
5     {
6         c=1;
7     }
8     else
9     {
10        c++;
11    }
12    application.setAttribute("count",c);
13
14    out.println("No of Visitors : "+c);
15
16 %>
```

Access the web page through Mozilla browser.



Access the web page through Google Chrome.



UNIT-V
Assignment-Cum-Tutorial Questions
SECTION-A

Objective Questions

1. The problem with servlets is []
A) Processing the request and generating the response are both handled by a single servlet class
B) Processing the request and generating the response are both handled by a two servlet classes
C) Processing the request and generating the response are both handled by a three servlet classes
D) Processing the request and generating the response are both handled by a four servlet classes

2. A servlet container and a JSP container are often combined in one package under the name _____ []
A) Server B) JSTL C) JSP Container D) Web Container

3. Which of the following is not a jsp directive? []
A) include B) page C) scriptlet D) useBean

4. Match the following:
1. <%@ include %> [] A) Expression
2. <jsp:useBean> [] B) Scriptlet
3. <% ---- %> [] C) Directive
4. <%= --- %> [] D) Action

5. Which of the following method can be used to read a form parameter in JSP? []
A) request.getParameter() B) response.getParameter() []
C) request.getValue() D) response.getValue()

6. How many jsp implicit objects are there []
A) 8 B) 9 C) 10 D) 7

7. Which tag is used to execute java source code in JSP? []
A) <%! --- %> B) <% ---%> C) <%= --- %> D) <%@ --- %>

8. Which JSP Action tags is used to include the content of another resource it may be jsp, html or servlet? []
A) jsp:forward B) jsp:plugin C) jsp:include D) jsp:param

9. Which JSP tag is used to transfer processing to another JSP page? []

- A) <jsp:include> B) <jsp:forward> C) <jsp:redirect> D) <jsp:useBean>
10. Which of the following standard action element makes a Java Beans component available in a page? []
A) <jsp:setProperty> B) <jsp:include> C) <jsp:useBean> D) <jsp:param>
11. Match the following scopes:
1. Page A) accessible from the pages which are processing the same request
 2. request B) accessible from the pages which are in same session
 3. session C) accessible from the page in which they are created.
 4. application D) accessible from the pages which reside in same application.
13. The JSP implicit object 'request' is an instance of which class? []
A) HttpRequest B) ServletRequest C) Request D) HttpServletRequest
14. Name the implicit object available to JSP pages that may be used to access all the other implicit objects []
A) Page B) Session C) PageContext D) request
15. In <jsp:useBean> which two attributes is necessary []
A) id and scope B) id and class C) type and id D) class and type
16. What is the correct signature of _jspService() method of HttpJspPage class?
A) void _jspService(HttpServletRequest request, HttpServletResponse response)
B) void _jspService(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
C) void _jspService() []
D) void _jspService() throws ServletException, IOException
17. Which of the following is legal JSP syntax to print the value of i. Select the one correct answer []
A) <% int i = 1; %>
 <%= i; %>
B) <% int i = 1;
 i; %>
C) <%int i = 1%>
 <%= i %>
D) <% int i = 1; %>
 <%= i %>

18. What gets printed when the following JSP code is invoked in a browser.

```
<%= if(Math.random() < 0.5) %>
hello
<%= } else { %>
hi
<%= } %>
```

- A) The browser will print either hello or hi based upon the return value of random. []
- B) The string hello will always get printed.
- C) The string hi will always get printed.
- D) The JSP file will not compile

SECTION-B

SUBJECTIVE QUESTIONS

1. What is XML? Explain the various building blocks of XML file.
2. Explain the basic structure of an XML document and Differentiate XML & HTML.
3. What is Document Type Definition (DTD)? Explain in detail Internal and External DTDs with examples.
4. Define an XML schema. Show how an XML schema can be created with an example.
5. Explain various types of XML Schema data types used.
6. Differentiate DOM and SAX XML Parsers.
7. Explain the various XSL elements in detail with Example.
8. Write the use of CSS in XML Document. Explain with an Example
9. Write down Internal and External DTDs for the following XML file

Students.xml:

```
<students>
  <student roll="1">
    <firstname> James </firstname>
    <lastname> Watson </lastname>
    <year> 3 </year>
    <courses>
      <course id="1">
        <name> Advanced Java </name>
      </course>
      <course id="2">
        <name> Web Technologies </name>
      </course>
    </courses>
  </student>
```

</students>.

10. Write a DTD for the XML document which has the Employee details with following fields arranged in a tabular format. Also assume the values of each field.

Employee Id	Name	Department	Designation	Sal
-------------	------	------------	-------------	-----

11. Design an XML Schema for hospital information management system.

12. Design XSL Sheet by using following XSL elements.

- (i) template
- (ii) stylesheet
- (iii) for-each
- (iv) value-of
- (vi) choose.

13. Design an XML Schema for the following XML Document

```
<?xml version="1.0"?>
<bookstore>
  <book>
    <title>WEB TECHNOLOGIES</title>
    <author>Uttam.K.Roy </author>
    <price>Rs.300 </price>

  </book>
  <book>
    <title>DATA STRUCTURES</title>
    <author>Gilberg </author>
    <author>Forouzan</author>
    <author>Prasad</author>
  </book>
</bookstore>
```

14. Design XML Schema for the following XML Document.

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

UNIT – 6

DATABASE ACCESS

Syllabus

JDBC Drivers,

Database Programming using JDBC,

Studying Javax.sql.* package,

Accessing a database from a JSP Page and a Servlet page,

Introduction to struts.

Objectives:

- To develop real time web applications.
- To get acquainted with skills for creating websites and web apps through learning various technologies like HTML, CSS, JavaScript, XML and Servlets and JSP,

Learning Outcomes:

Students will be able to

- Know the types of JDBC Drivers
- Learn Database Programming using JDBC
- Aware the javax.sql.* Package.
- Know how to Accessing a database from a JSP Page and a Servlet page.
- Learn How to work with struts framework.

1. JDBC Drivers

JDBC drivers are divided into four types or levels. The **different types of jdbc drivers** are:

Type 1: JDBC-ODBC Bridge driver (Bridge)

Type 2: Native-API/partly Java driver (Native)

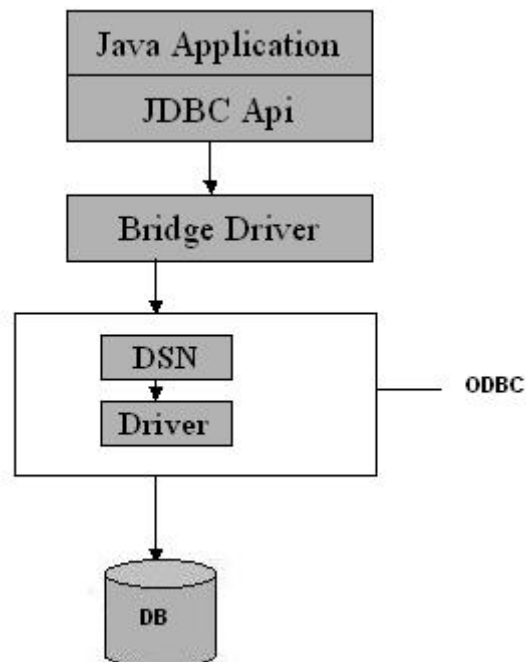
Type 3: AllJava/Net-protocol driver (Middleware)

Type 4: All Java/Native-protocol driver (Pure)

Type 1 JDBC Driver

JDBC-ODBC Bridge driver

The Type 1 driver translates all JDBC calls into ODBC calls and sends them to the ODBC driver. ODBC is a generic API. The JDBC-ODBC Bridge driver is recommended only for experimental use or when no other alternative is available.



ype 1: JDBC-ODBC Bridge

Advantage

The JDBC-ODBC Bridge allows access to almost any database, since the database's ODBC drivers are already available.

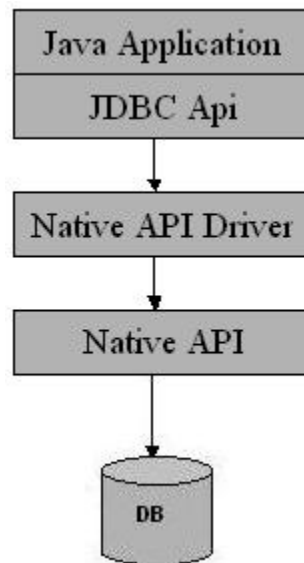
Disadvantages

1. Since the Bridge driver is not written fully in Java, Type 1 drivers are not portable.
2. A performance issue is seen as a JDBC call goes through the bridge to the ODBC driver, then to the database, and this applies even in the reverse process. They are the slowest of all driver types.
3. The client system requires the ODBC Installation to use the driver.
4. Not good for the Web.

Type 2 JDBC Driver

Native-API/partly Java driver

The distinctive characteristic of type 2 jdbc drivers are that Type 2 drivers convert JDBC calls into database-specific calls i.e. this driver is specific to a particular database. Some distinctive characteristic of type 2 jdbc drivers are shown below. Example: Oracle will have oracle native api.



Type 2: Native api/ Partly Java Driver

Advantage

The distinctive characteristic of type 2 jdbc drivers are that they are typically offer better performance than the JDBC-ODBC Bridge as the layers of communication (tiers) are less than that of Type 1 and also it uses Native api which is Database specific.

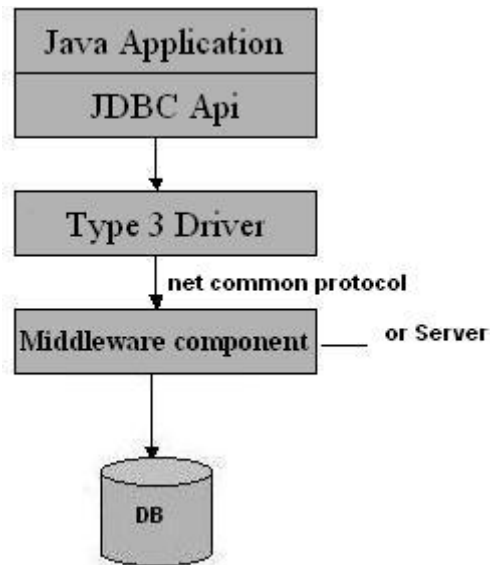
Disadvantage

1. Native API must be installed in the Client System and hence type 2 drivers cannot be used for the Internet.
2. Like Type 1 drivers, it's not written in Java Language which forms a portability issue.
3. If we change the Database we have to change the native api as it is specific to a database
4. Mostly obsolete now
5. Usually not thread safe.

Type 3 JDBC Driver

All Java/Net-protocol driver

Type 3 database requests are passed through the network to the middle-tier server. The middle-tier then translates the request to the database. If the middle-tier server can in turn use Type1, Type 2 or Type 4 drivers.



Type 3: All Java/ Net-Protocol Driver

Advantage

1. This driver is server-based, so there is no need for any vendor database library to be present on client machines.
2. This driver is fully written in Java and hence Portable. It is suitable for the web.
3. There are many opportunities to optimize portability, performance, and scalability.
4. The net protocol can be designed to make the client JDBC driver very small and fast to load.
5. The type 3 driver typically provides support for features such as caching (connections, query results, and so on), load balancing, and advanced system administration such as logging and auditing.
6. This driver is very flexible allows access to multiple databases using one driver.
7. They are the most efficient amongst all driver types.

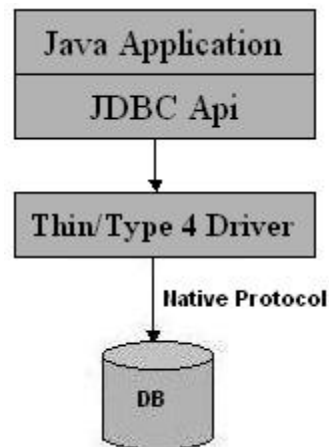
Disadvantage

It requires another server application to install and maintain. Traversing the recordset may take longer, since the data comes through the backend server.

Type 4 JDBC Driver

Native-protocol/all-Java driver

The Type 4 uses java networking libraries to communicate directly with the database server.



Type 4: Native-protocol/all-Java driver

Advantage

1. The major benefit of using a type 4 jdbc drivers are that they are completely written in Java to achieve platform independence and eliminate deployment administration issues. It is most suitable for the web.
2. Number of translation layers is very less i.e. type 4 JDBC drivers don't have to translate database requests to ODBC or a native connectivity interface or to pass the request on to another server, performance is typically quite good.
3. You don't need to install special software on the client or server. Further, these drivers can be downloaded dynamically.

Disadvantage

With type 4 drivers, the user needs a different driver for each database.

2. Database Programming using JDBC:

JDBC stands for **J**ava **D**atabase **C**onnectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.

- Making a connection to a database.
- Creating SQL or MySQL statements.
- Executing SQL or MySQL queries in the database.
- Viewing & Modifying the resulting records.

Fundamentally, JDBC is a specification that provides a complete set of interfaces that allows for portable access to an underlying database. Java can be used to write different types of executables, such as –

- Java Applications
- Java Applets
- Java Servlets
- Java ServerPages (JSPs)
- Enterprise JavaBeans (EJBs).

All of these different executables are able to use a JDBC driver to access a database, and take advantage of the stored data.

JDBC provides the same capabilities as ODBC, allowing Java programs to contain database-independent code.

JDBC Architecture

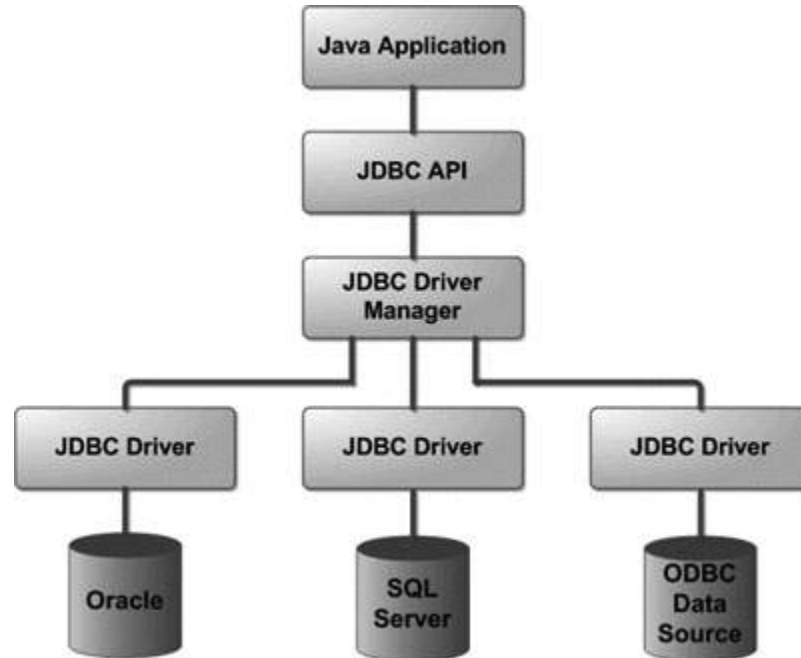
The JDBC API supports both two-tier and three-tier processing models for database access but in general, JDBC Architecture consists of two layers –

- **JDBC API:** This provides the application-to-JDBC Manager connection.
- **JDBC Driver API:** This supports the JDBC Manager-to-Driver Connection.

The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases.

The JDBC driver manager ensures that the correct driver is used to access each data source. The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases.

Following is the architectural diagram, which shows the location of the driver manager with respect to the JDBC drivers and the Java application –



Common JDBC Components

The JDBC API provides the following interfaces and classes –

- **DriverManager:** This class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication sub protocol. The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.
- **Driver:** This interface handles the communications with the database server. You will interact directly with Driver objects very rarely. Instead, you use DriverManager objects, which manages objects of this type. It also abstracts the details associated with working with Driver objects.
- **Connection:** This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.
- **Statement:** You use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.
- **ResultSet:** These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.
- **SQLException:** This class handles any errors that occur in a database application.

3. Studying Javax.sql.* package

Provides the API for server side data source access and processing from the Java™ programming language. This package supplements the java.sql package.

The javax.sql package provides for the following:

1. The DataSource interface as an alternative to the DriverManager for establishing a connection with a data source
2. Connection pooling and Statement pooling
3. Distributed transactions
4. Rowsets

Applications use the DataSource and RowSet APIs directly, but the connection pooling and distributed transaction APIs are used internally by the middle-tier infrastructure.

Interface Summary

Interface	Description
CommonDataSource	Interface that defines the methods which are common between DataSource, XADataSource and ConnectionPoolDataSource.
ConnectionEventListener	An object that registers to be notified of events generated by a PooledConnection object.
ConnectionPoolDataSource	A factory for PooledConnection objects.
DataSource	A factory for connections to the physical data source that this DataSource object represents.
PooledConnection	An object that provides hooks for connection pool management.
RowSet	The interface that adds support to the JDBC API for the JavaBeans™ component model.
RowSetInternal	The interface that a RowSet object implements in order to present itself to a RowSetReader or RowSetWriter object.
RowSetListener	An interface that must be implemented by a component that wants to be notified when a significant event happens in the life of a RowSet object.
RowSetMetaData	An object that contains information about the columns in a RowSet object.
RowSetReader	The facility that a disconnected RowSet object calls on to populate itself with rows of data.
RowSetWriter	An object that implements the RowSetWriter interface, called a <i>writer</i> .
StatementEventListener	An object that registers to be notified of events that occur on PreparedStatements that are in the Statement pool.
XAConnection	An object that provides support for distributed transactions.
XADataSource	A factory for XAConnection objects that is used internally.

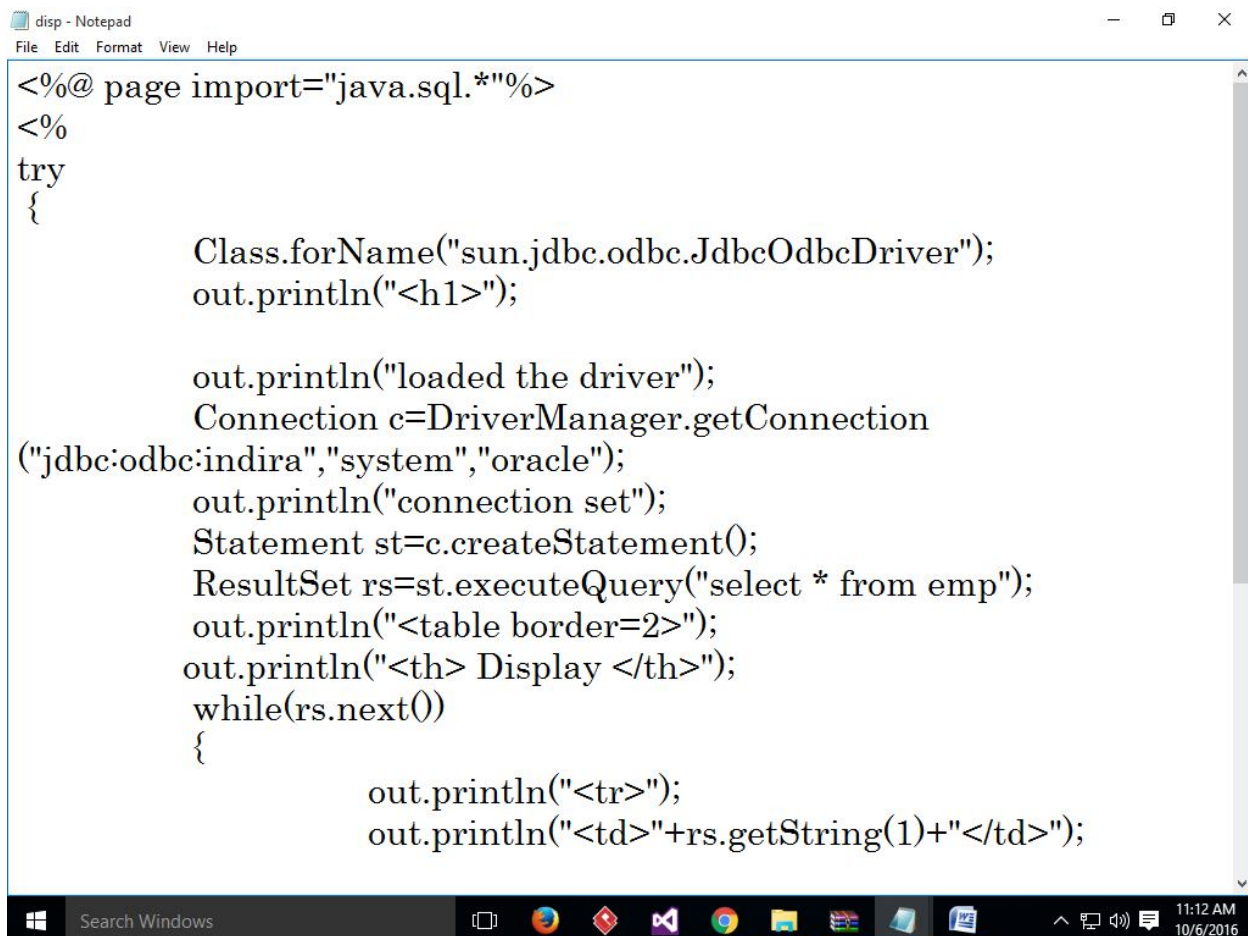
Class Summary

Class	Description
ConnectionEvent	An EVENT object that provides information about the source of a connection-related event.
RowSetEvent	An EVENT object generated when an event occurs to a RowSet object.
StatementEvent	A StatementEvent is sent to all StatementEventListeners which were registered with a PooledConnection.

4. Accessing a database from a JSP Page and a Servlet page:

Sample programs to access database from JSP:

- Write a JSP to display employee number and name from emp table



```
disp - Notepad
File Edit Format View Help
<%@ page import="java.sql.*"%>
<%
try
{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    out.println("<h1>");

    out.println("loaded the driver");
    Connection c=DriverManager.getConnection
("jdbc:odbc:indira","system","oracle");
    out.println("connection set");
    Statement st=c.createStatement();
    ResultSet rs=st.executeQuery("select * from emp");
    out.println("<table border=2>");
    out.println("<th> Display </th>");
    while(rs.next())
    {
        out.println("<tr>");
        out.println("<td>"+rs.getString(1)+"</td>");
    }
}
```

```
disp - Notepad
File Edit Format View Help

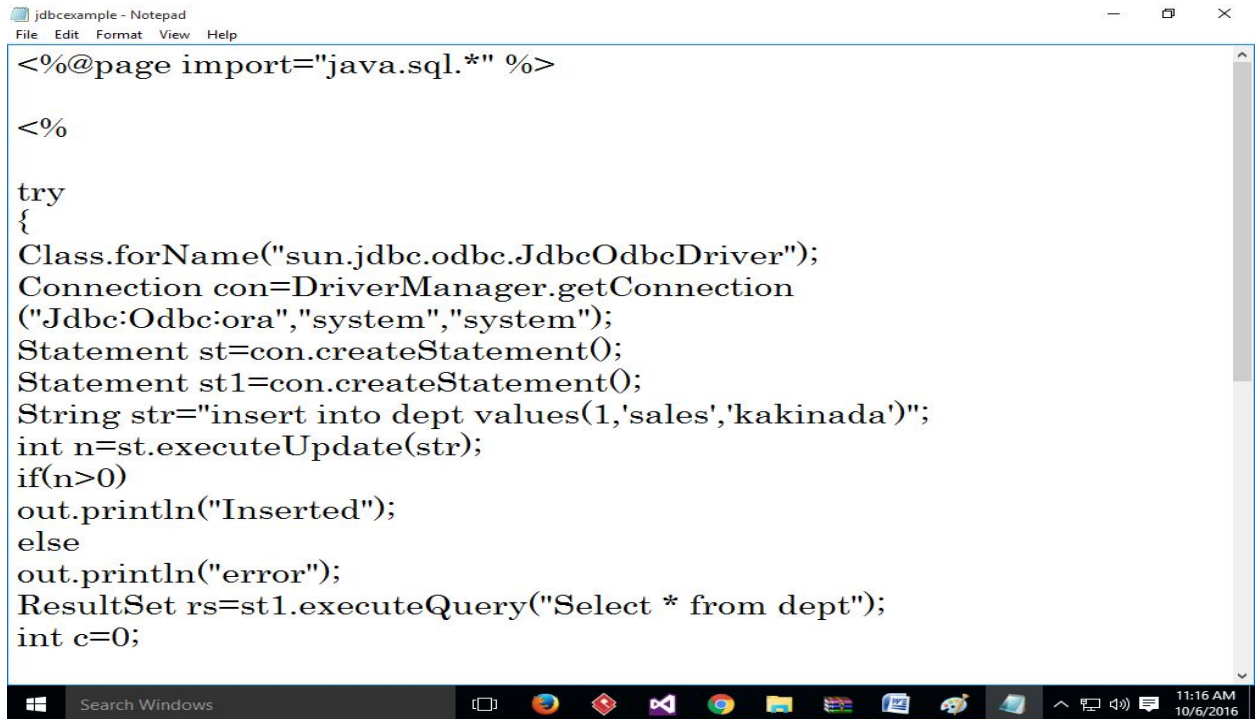
Statement st=c.createStatement();
ResultSet rs=st.executeQuery("select * from emp");
out.println("<table border=2>");
out.println("<th> Display </th>");
while(rs.next()
{
    out.println("<tr>");
    out.println("<td>"+rs.getString(1)+"</td>");
    out.println("<td>"+rs.getString(2)+"</td>");
    out.println("</tr>");
}
out.println("/<table>");
out.println("</h1>");
}
catch(Exception e)
{
    out.println(e.toString());
}
}
| %>
```

Output:

loaded the driver connection set /

Display	
1111	sai
1111	sai
2222	jhon
2222	jhon
123	manisha
123	manisha
123	manisha
123	manisha

- Write a JSP to Insert one record into dept table and Display them.

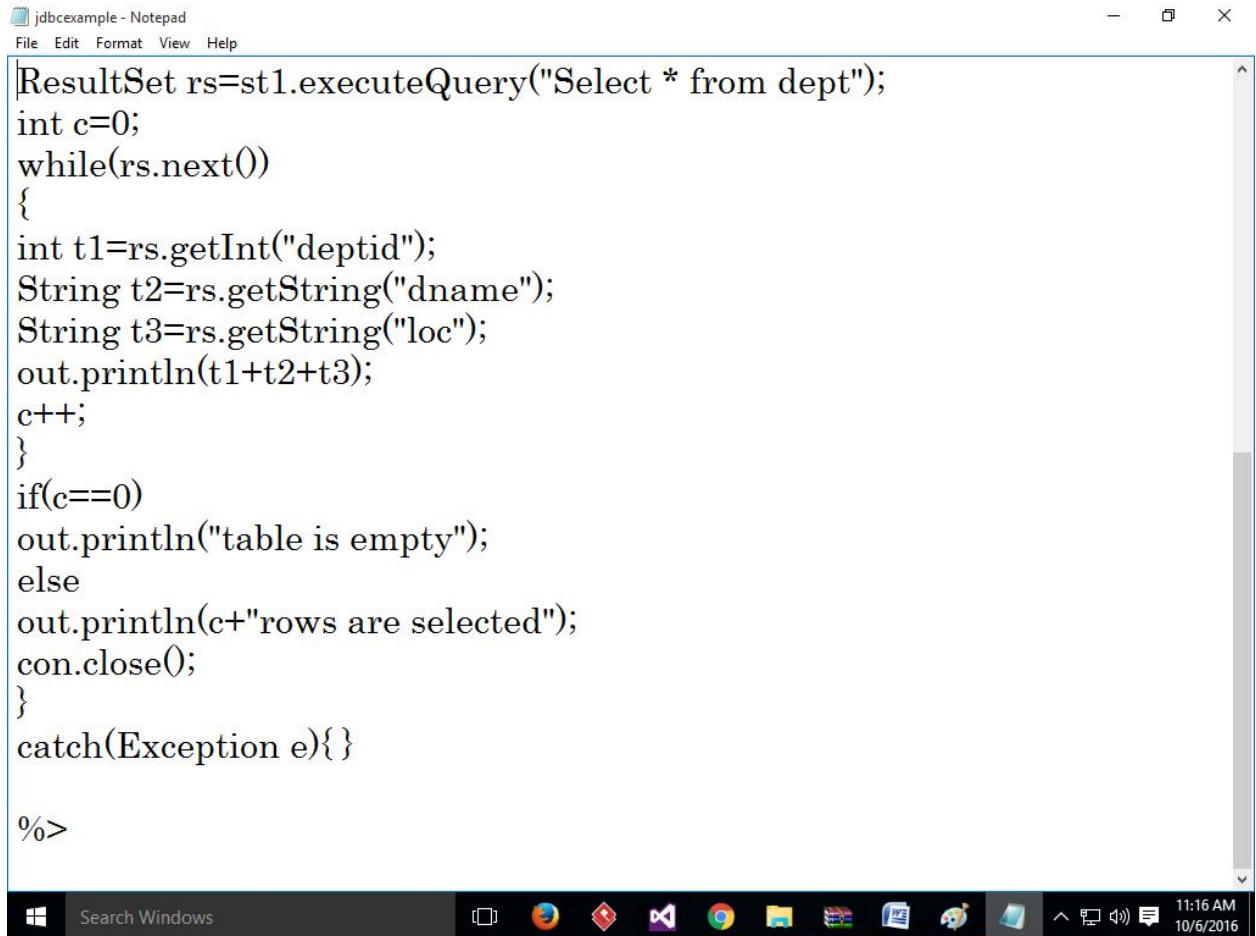


```
jdbcxample - Notepad
File Edit Format View Help

<%@page import="java.sql.*" %>

<%

try
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con=DriverManager.getConnection
("Jdbc:Odbc:ora","system","system");
Statement st=con.createStatement();
Statement st1=con.createStatement();
String str="insert into dept values(1,'sales','kakinada)";
int n=st.executeUpdate(str);
if(n>0)
out.println("Inserted");
else
out.println("error");
ResultSet rs=st1.executeQuery("Select * from dept");
int c=0;
```



```
ResultSet rs=st1.executeQuery("Select * from dept");
int c=0;
while(rs.next())
{
int t1=rs.getInt("deptid");
String t2=rs.getString("dname");
String t3=rs.getString("loc");
out.println(t1+t2+t3);
c++;
}
if(c==0)
out.println("table is empty");
else
out.println(c+"rows are selected");
con.close();
}
catch(Exception e){ }
```

%>

WRITE A JSP TO STORE USER_ID AND PASSWORD and Display
User.html

```
client - Notepad
File Edit Format View Help
<center> WRITE A JSP TO STORE USER_ID AND
PASSWORD</center>

<body text = ref bgcolor=cyan>
<center>
<h1>
<h1>Registration</h1>
<form method = post action = "http://localhost:4040/newuser.jsp">
userid<input type = text name="tid">
pwd<input type = password name="tpwd">
confirm password<input type = password name="tcpwd">
<input type = submit value = "Register">
</form>
</h1>
</center>
</body>
```

Newuser.jsp

```
newuser - Notepad
File Edit Format View Help
<% @ page import = "java.sql.*"%>
<body>
<%
try
{
class.forName
("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con=DriverManager.getConnection
("jdbc:odbc:indira","system","oracle");
Statement st=con.createStatement();
String id,pwd,cp;
id=request.getParameter("tid");
pwd=request.getParameter("tpwd");
cp=request.getparameter("tcpwd");
if(pwd.equals(cp))
ResultSet rs=st.executeQuery("select * from webusers where
userid="+id);
if(rs.next())
{
```

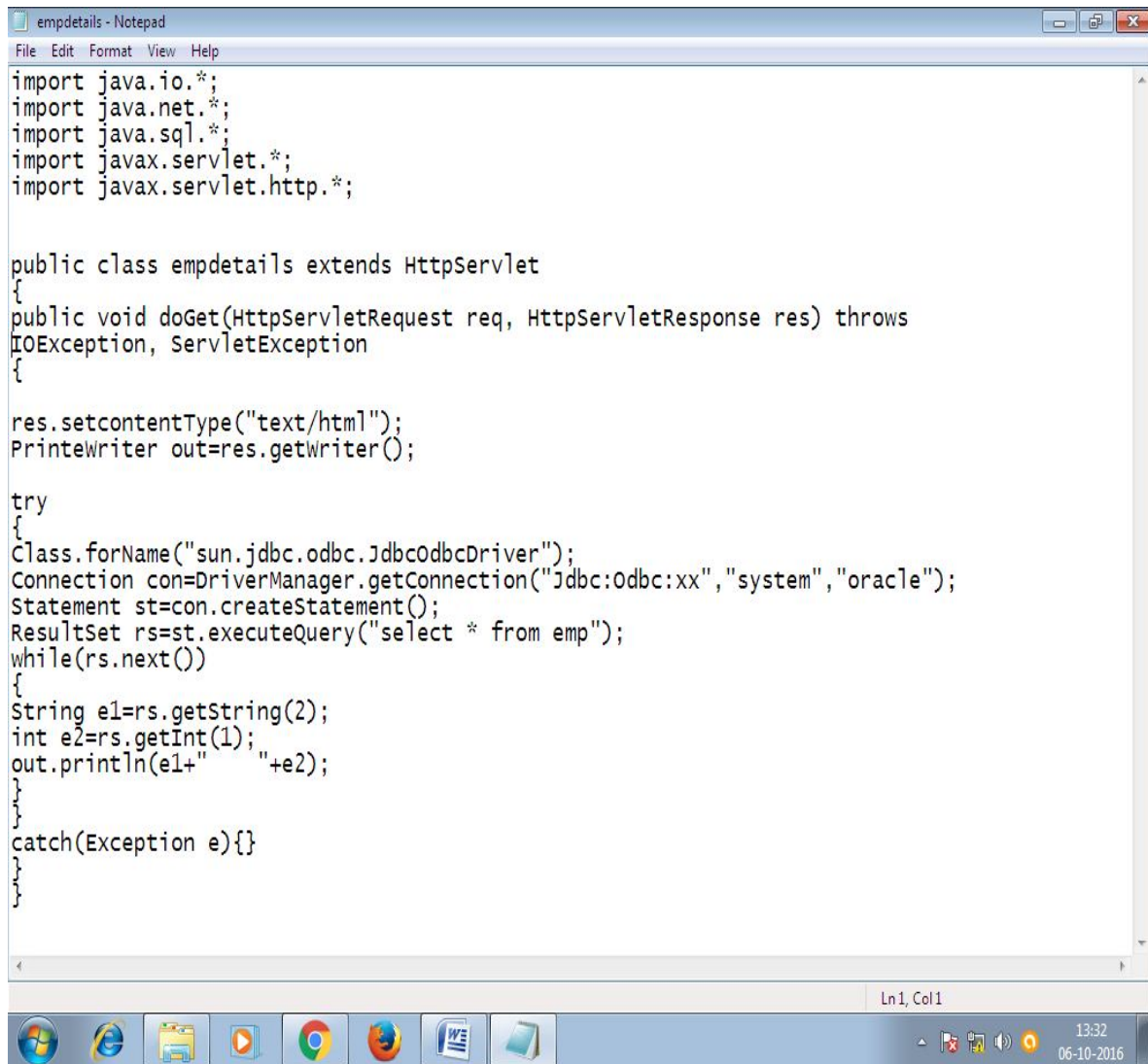
```
newuser - Notepad
File Edit Format View Help
{
int n=executeUpdate("insert into webusers values('"+id+"'"+pwd
+"""+"""+cpp""");
if(n>0)
System.out.println("Inserted
Successfully");
else
System.out.println("Id already
exists");
}
else
System.out.println("Check
password");
con.close();
}
catch(Exception e)
{
c.printStackTrace();
}
}

Search Windows 11:39 AM
10/6/2016
```

```
newuser - Notepad
File Edit Format View Help
+"""+"""+cpp""");
if(n>0)
System.out.println("Inserted
Successfully");
else
System.out.println("Id already
exists");
}
else
System.out.println("Check
password");
con.close();
}
catch(Exception e)
{
c.printStackTrace();
}
}
%>
</body>

Search Windows 11:39 AM
10/6/2016
```

- Servlet Program to print Emp details



```
empdetails - Notepad
File Edit Format View Help
import java.io.*;
import java.net.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

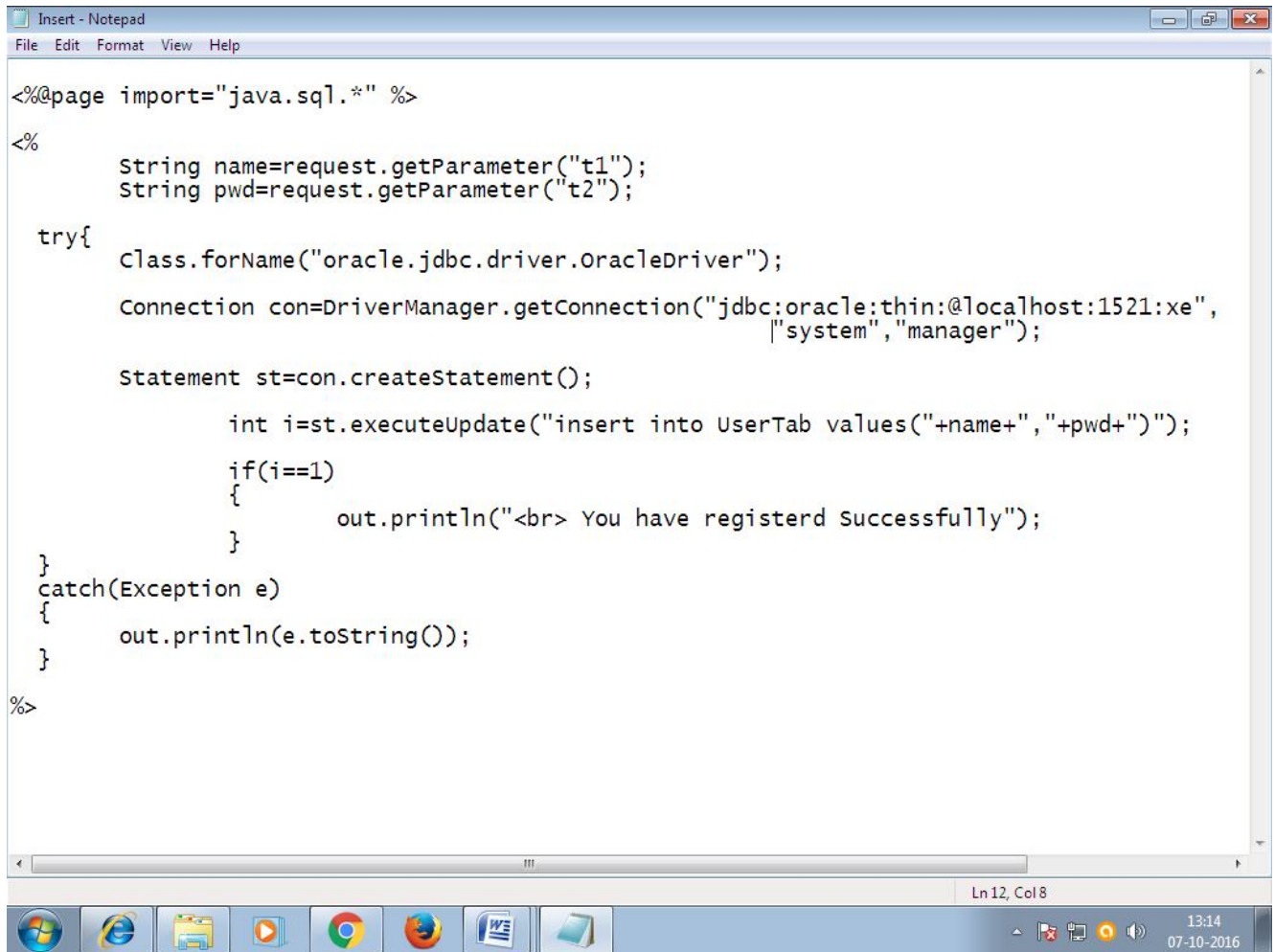
public class empdetails extends HttpServlet
{
public void doGet(HttpServletRequest req, HttpServletResponse res) throws
IOException, ServletException
{
res.setContentType("text/html");
PrintWriter out=res.getWriter();

try
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con=DriverManager.getConnection("Jdbc:Odbc:xx","system","oracle");
Statement st=con.createStatement();
ResultSet rs=st.executeQuery("select * from emp");
while(rs.next())
{
String e1=rs.getString(2);
int e2=rs.getInt(1);
out.println(e1+" "+e2);
}
}
catch(Exception e){}
}
```

Ln1, Col1

13:32
06-10-2016

Accessing Database using Type 4 Driver



```
<%@page import="java.sql.*" %>
<%
    String name=request.getParameter("t1");
    String pwd=request.getParameter("t2");

    try{
        Class.forName("oracle.jdbc.driver.OracleDriver");
        Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe",
            "system","manager");

        Statement st=con.createStatement();

        int i=st.executeUpdate("insert into UserTab values("+name+","+pwd+")");
        if(i==1)
        {
            out.println("<br> You have registered successfully");
        }
    }
    catch(Exception e)
    {
        out.println(e.toString());
    }
%>
```

5. Introduction to struts.

The Struts Framework is a standard for developing well-architected Web applications. It has the following features:

- Open source
- Based on the Model-View-Controller (MVC) design paradigm, distinctly separating all three levels:
 - **Model:** application state
 - **View:** presentation of data (JSP, HTML)

- **Controller:** routing of the application flow
- Implements the JSP Model 2 Architecture
- Stores application routing information and request mapping in a single core file, struts-config.xml

Struts Components

The Controller

This receives all incoming requests. Its primary function is the mapping of a request URI to an action class selecting the proper application module. It's provided by the framework.

The struts-config.xml File

This file contains all of the routing and configuration information for the Struts application. This XML file needs to be in the WEB-INF directory of the application.

Action Classes

It's the developer's responsibility to create these classes. They act as bridges between user-invoked URIs and business services. Actions process a request and return an ActionForward object that identifies the next component to invoke. They're part of the Controller layer, not the Model layer.

View Resources

View resources consist of Java Server Pages, HTML pages, JavaScript and Stylesheet files, Resource bundles, JavaBeans, and Struts JSP tags.

ActionForms

These greatly simplify user form validation by capturing user data from the HTTP request. They act as a "firewall" between forms (Web pages) and the application (actions). These components allow the validation of user input before proceeding to an Action. If the input is invalid, a page with an error can be displayed.

Model Components

The Struts Framework has no built-in support for the Model layer. Struts supports any model components:

- JavaBeans
- EJB
- CORBA
- JDO

- any other

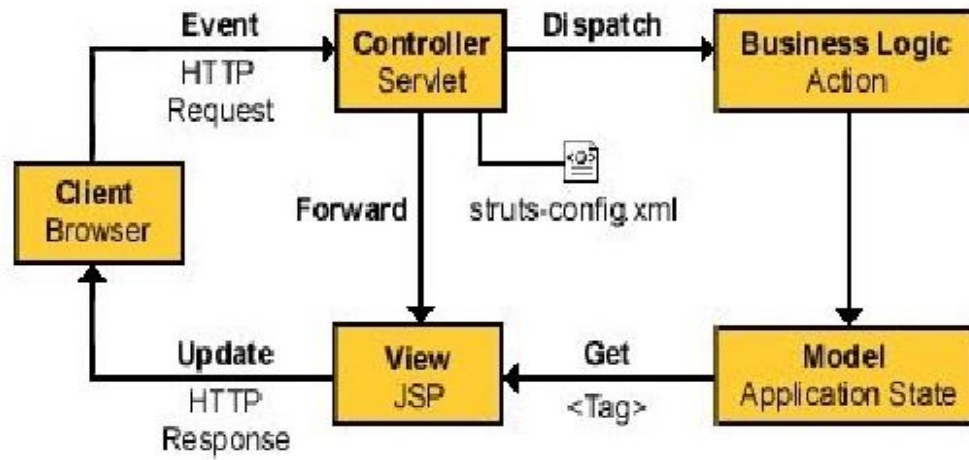


Fig: Struts framework Architecture

UNIT-VI
Assignment-Cum-Tutorial Questions
SECTION-A

Objective Questions

1. _____ is a set of classes and interfaces that allows a Java application to handle SQL statements to a Database. []
A) JNDI B) JDBC API C) ODBC D) ODC
2. How many JDBC driver types does Sun define? []
A) one B) two C) three D) four
3. Which type driver translates JDBC to ODBC and relies on ODBC to communicate with the database . []
A) Type 1 B) Type 2 C) Type 3 D) Type 4
4. Which type driver is a pure java library that translates JDBC requests directly to database specific protocol. []
A) Type 1 B) Type 2 C) Type 3 D) Type 4
5. Which class manages a List of database drivers. []
A) Driver B) DriverManager C) Statement D) Connection
6. Statement is used to execute _____ SQL queries. []
A) Static B) Dynamic C) User level D) Run time
7. Pre Compiled SQL Statements can be written by which class. []
A) Statement B) CallableStatement C) PreparedStatement D) ResultSet
8. Which method you can use to execute SQL SELECT Statement and to return a ResultSet. []
A) execute() B) executeQuery() C) executeUpdate() D) executeBatch()
9. Which of the following describes the correct sequence of the steps involved in making a connection with a database. []
 1. Loading the driver.
 2. Process the results.
 3. Making the connection with the database.
 4. Executing the SQL statements.


```
PreparedStatement stmt=con.prepareStatement("select * from emp");
ResultSet rs=stmt.executeQuery();
while(rs.next()){
    System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

- A) deletes the record B) retrieve the record
C) updates the record D) inserts the record

- 18.The method sets the query parameters of the PreparedStatement Object. []
A) putString() B) insertString() C) setString() D) setToString()

SECTION-B

SUBJECTIVE QUESTIONS

1. Define JDBC. Explain the JDBC architecture.
2. Discuss different JDBC drivers with its architectures.
3. Explain the following:
 - A) DriverManager class
 - B) Connection Interface
 - C) Statement Interface
 - D) executeQuery() method
 - E) ResultSet Interface
4. Explain with an example the steps to connect and access a database from a JSP page.
5. List and Explain various classes and interfaces in javax.sql package
6. Define Struts framework. Explain in detail the architecture of Struts framework.
7. Develop a JSP program to update the salaryRs.6000/- for an employee name "Ramu" using prepared statement.
8. Write a JSP program that lists all books and the authors for those books from technical library database.
9. Implement a JSP to display the results of a student by accepting studentid in a text box. Results will be maintained in a database table.

10. Design a JSP program that outputs details of all the books in the technical-library database with author names starting with the letter 'A' .
11. Write a JSP program to insert six subjects marks of a Student into database using PreparedStatement.
12. Implement a JSP program to insert the details of the users who register with the Online book store website by using registration form.
13. Write a JSP which Authenticates the *user* when he submits the login form using the user name and password from the database.