

WEB PROGRAMMING

III Year B. Tech. DS I- Sem

**L T P C
2 0 0 2**

Course Objectives: The student should be able to:

- Understand the technologies used in Web Programming.
- Know the importance of object-oriented aspects of Scripting.
- Understand creating database connectivity using JDBC.
- Learn the concepts of web-based application using sockets.

Course Outcomes: Upon Completion of the course, the students will be able to

- Design web pages.
- Use technologies of Web Programming.
- Apply object-oriented aspects to Scripting.
- Create databases with connectivity using JDBC.
- Build web-based application using sockets.

UNIT – I

SCRIPTING.

Web page Designing using HTML, Scripting basics- Client side and server side scripting. Java Script Object, names, literals, operators and expressions- statements and features- events - windows - documents – frames - data types - built-in functions- Browser object model - Verifying forms.-HTML5- CSS3- HTML 5 canvas - Web site creation using tools.

UNIT – II

JAVA

Introduction to object-oriented programming-Features of Java – Data types, variables and arrays – Operators – Control statements – Classes and Methods – Inheritance. Packages and Interfaces – Exception Handling – Multithreaded Programming – Input/Output – Files – Utility Classes – String Handling.

UNIT – III

JDBC

JDBC Overview – JDBC implementation – Connection class – Statements - Catching Database Results, handling database Queries. Networking– InetAddress class – URL class- TCP sockets – UDP sockets, Java Beans –RMI.

UNIT – IV

APPLETS

Java applets- Life cycle of an applet – Adding images to an applet – Adding sound to an applet. Passing parameters to an applet. Event Handling. Introducing AWT: Working with Windows Graphics and Text. Using AWT Controls, Layout Managers and Menus. Servlet – life cycle of a servlet.

The Servlet API, Handling HTTP Request and Response, using Cookies, Session Tracking Introduction to JSP.

UNIT – V

XML AND WEB SERVICES

Xml – Introduction-Form Navigation-XML Documents- XSL – XSLT- Web services-UDDI-WSDL-Java web services – Web resources.

TEXT BOOKS:

1. Harvey Deitel, Abbey Deitel, Internet and World Wide Web: How To Program 5th Edition.
2. Herbert Schildt, Java - The Complete Reference, 7th Edition. Tata McGraw- Hill Edition.
3. Michael Morrison XML Unleashed Tech media SAMS.

REFERENCE BOOKS:

1. John Pollock, Javascript - A Beginners Guide, 3rd Edition -- Tata McGraw-Hill Edition.
2. Keyur Shah, Gateway to Java Programmer Sun Certification, Tata McGraw Hill, 2002

UNIT - I

SCRIPTING

Web page Designing using HTML, Scripting basics- Client side and server side scripting. Java Script Object, names, literals, operators and expressions- statements and features- events - windows - documents - frames - data types - built-in functions- Browser object model - Verifying forms.-HTML5- CSS3- HTML 5 canvas - Web site creation using tools.

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

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

History of Internet

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

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

World Wide Web

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

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

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



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

Browsers:

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

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

1991	World Wide Web (Nexus)
1992	Viola WWW, Erwise, MidasWWW, MacWWW (Samba)
1993	Mosaic,Cello,[2] Lynx 2.0, Arena, AMosaic 1.0
1994	IBM WebExplorer, Netscape Navigator, SlipKnot 1.0, MacWeb, IBrowse, Agora (Argo), Minuet
1995	Internet Explorer 1, Internet Explorer 2, Netscape Navigator 2.0, OmniWeb, UdiWWW, Grail
1996	Arachne 1.0, Internet Explorer 3.0, Netscape Navigator 3.0,Opera 2.0, PowerBrowser 1.5,[4] Cyberdog,Amaya 0.9,[5] AWeb,Voyager
1997	Internet Explorer 4.0, Netscape Navigator 4.0, Netscape Communicator 4.0, Opera3.0,[6] Amaya 1.0[5]
1998	iCab, Mozilla
1999	Amaya 2.0,[5] Mozilla M3, Internet Explorer 5.0
2000	Konqueror,Netscape 6, Opera 4,[7] Opera 5,[8] K-Meleon 0.2, Amaya 3.0,[5] Amaya 4.0[5]
2001	Internet Explorer 6, Galeon 1.0, Opera 6,[9] Amaya 5.0[5]
2002	Netscape 7, Mozilla 1.0, Phoenix 0.1, Links 2.0, Amaya 6.0,[5] Amaya 7.0[5]
2003	Opera 7,[10] Apple Safari 1.0, Epiphany 1.0, Amaya 8.0[5]
2004	Firefox 1.0, Netscape Browser, OmniWeb 5.0
2005	Opera8,[11]Apple Safari2.0, Netscape Browser 8.0, Epiphany 1.8, Amaya 9.0,[5] AOL Explorer 1.0, Maxthon 1.0, Shiira 1.0
2006	Mozilla Firefox 2.0, Internet Explorer 7,Opera 9,[12], SeaMonkey 1.0, K-Meleon 1.0, Galeon 2.0, Camino 1.0, Avant11, iCab 3
2007	Apple Safari 3.0, Maxthon 2.0, Netscape Navigator9,NetSurf 1.0, Flock 1.0, Conkeror
2008	Google Chrome 1, Mozilla Firefox 3, Opera 9.5,[13], Apple Safari 3.1, Konqueror 4, Amaya 10.0,[5] Flock 2, Amaya 11.0[5]
2009	Google Chrome 2–3, Mozilla Firefox 3.5, Internet Explorer 8,Opera 10,[14], Apple Safari 4, SeaMonkey 2, Camino 2,surf, Pale Moon 3.0[15]
2010	Google Chrome 4–8, Mozilla Firefox 3.6, Opera 10.50,[16], Opera 11, Apple Safari 5, K-Meleon 1.5.4,
2011	Google Chrome 9–16, Mozilla Firefox 4-9, Internet Explorer 9,Opera 11.50, Apple Safari 5.1, Maxthon 3.0, SeaMonkey 2.1–2.6
2012	Google Chrome 17–23, Mozilla Firefox 10–17, Internet Explorer 10, Opera 12, Apple Safari 6, Maxthon 4.0, SeaMonkey 2.7–2.14
2013	Google Chrome 24–31,Mozilla Firefox 18–26,Internet Explorer 11, Opera 15–18, Apple Safari 7, SeaMonkey 2.15–2.23

2014	Google Chrome 32–39, Mozilla Firefox 27–34, Opera 19–26, Apple Safari 8
2015	Google Chrome 40–47, Microsoft Edge,Mozilla Firefox 35–43, Opera 27–34, Vivaldi
2016	Google Chrome 48–55,Mozilla Firefox 44–50,Microsoft Edge 14, Opera35–42, Apple Safari 10, SeaMonkey 2.24–2.30, Pale Moon 26.0.0[17], Pale Moon 27.0.0[18]
2017	Google Chrome56–60,Microsoft Edge 15,Mozilla Firefox 51–55.0.2, Opera43–45, Opera Neon

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

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

Hypertext protocol:

<http://www.aucegypt.edu> File Transfer

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

Protocol: telnet://pac.carl.org

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

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

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

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

MIME (Multi-Purpose Internet Mail Extensions):- MIME is an extension of the original Internet e-mail protocol that lets people use the protocol to exchange different kinds of data files on the Internet: audio, video, images, application programs, and other kinds, as well as the ASCII text handled in the original protocol, the Simple Mail Transport Protocol (SMTP). In 1991, Nathan Borenstein of Bellcore proposed to the IETF that SMTP be extended so that Internet (but

mainly Web) clients and servers could recognize and handle other kinds of data than ASCII text. As a result, new file types were added to "mail" as a supported Internet Protocol file type.

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

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

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

Hypertext Transport Protocol:

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

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

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

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

The Web Programmer's Toolbox:

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

- Filters
 - Convert documents in other formats to HTML
 - Advantages of both filters and plug-ins:
 - Existing documents produced with other tools can be converted to HTML documents
 - Use a tool you already know to produce HTML
 - Disadvantages of both filters and plug-ins:
 - HTML output of both is not perfect - must be finetuned
 - HTML may be non-standard
 - You have two versions of the document, which are difficult to synchronize
 - XML
 - A meta-markup language (a language for defining markup language)
 - Used to create a new markup language for a particular purpose or area
 - Because the tags are designed for a specific area, they can be meaningful
 - JavaScript
 - A client-side HTML-embedded scripting language
 - Provides a way to access elements of HTML documents and dynamically change them
 - Flash
 - A system for building and displaying text, graphics, sound, interactivity, and animation (movies)
 - Two parts:
 1. Authoring environment
 2. Player
- Supports both motion and shape animation
- PHP**
- A server-side scripting language
Great for form processing and database access through the Web
- Ajax**
- Asynchronous JavaScript + XML
 - No new technologies or languages

Much faster for Web applications that have extensive user/server interactions
Uses asynchronous requests to the server
Requests and receives small parts of documents, resulting in much faster responses

Java Web Software

Servlets – server-side Java classes
JavaServer Pages (JSP) – a Java-based approach to server-side scripting
JavaServer Faces – adds an event-driven interface model on JSP

ASP.NET

Does what JSP and JSF do, but in the .NET environment
Allows .NET languages to be used as server-side scripting language

Ruby

A pure object-oriented interpreted scripting language
Every data value is an object, and all operations are via method calls
Both classes and objects are dynamic

Rails

A development framework for Web-based applications
 Particularly useful for Web applications that access databases
 Written in Ruby and uses Ruby as its primary user language

HTML Common tags:-

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

- HTML stands for Hyper Text MarkupLanguage
- An HTML file is a text file containing small markuptags
- The markup tags tell the Web browser how to display thepage
- An HTML file must have an htm or html fileextension.

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

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

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

Headings:-

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

<h1>This is a heading</h1>

```
<h2>This is a heading</h2>
<h3>This is a heading</h3>
<h4>This is a heading</h4>
<h5>This is a heading</h5>
<h6>This is a heading</h6>
```

Paragraphs:-

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

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

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

Line Breaks:-

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

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

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

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

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

Sample html program

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



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

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

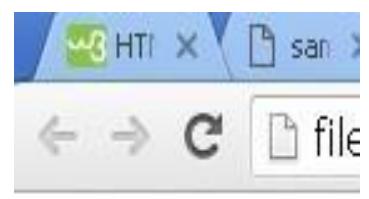
All lists must contain one or more list elements. Lists are of three types

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

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

Example

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



- Beetroot
- Ginger
- Potato
- Radish

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

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

Web T tag



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

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

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

```
<!DOCTYPE html>
<html>
  <head>
    <title>HTML Definition List</title>
  </head>
  <body>
    <dl>
      <dt><b>HTML</b></dt><dd>This stands for Hyper Text Markup Language</dd>
      <dt><b>HTTP</b></dt><dd>This stands for Hyper Text Transfer Protocol</dd>
    </dl>
  </body>
</html>
```

HTML tables:

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

Example:

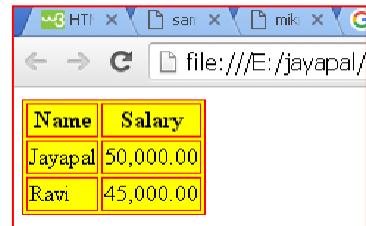
```
<!DOCTYPE html>
<html>
  <head>
    <title>HTML Tables</title>
  </head>
  <body>
    <table border="1">
      <tr>
        <td>Row 1, Column 1</td><td>Row 1, Column 2</td>
      </tr>
      <tr><td>Row 2, Column 1</td><td>Row 2, Column 2</td>
    </tr>
  </table>
</body>
```

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

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

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

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



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

Insert Image:

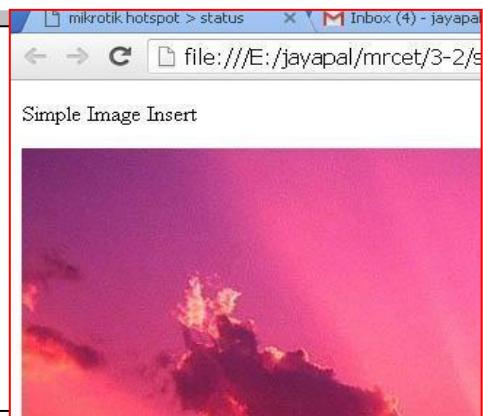
insert any image in the web page by using **** tag.

Attribute Values

Value	Description
left	Align the image to the left
right	Align the image to the right
middle	Align the image in the middle

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

 </body>
</html>
```



### HTML FORMS:

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

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

### Form Attributes

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

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

### HTML Form Controls

There are different types of form controls that you can use to collect data using HTML form:

- Text InputControls
- Checkboxes Controls
- Radio BoxControls
- Select BoxControls
- File Selectboxes
- Hidden Controls
- ClickableButtons
- Submit and ResetButton

#### **Text Input Controls:-**

There are three types of text input used on forms:

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

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

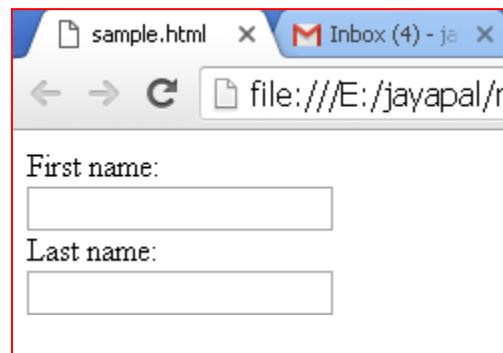
#### **Example:**

```
<form>
 Firstname:

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

 Lastname:

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



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

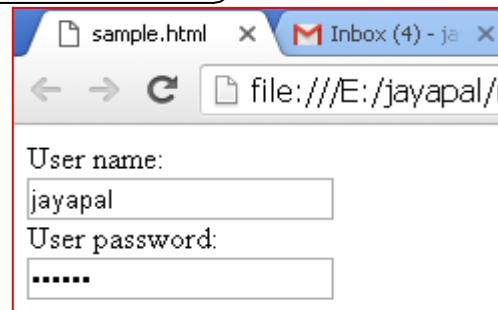
#### **Input Type Password**

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

```
<form>
 User name:

 <input type="text"
 name="username">
 User
 password:

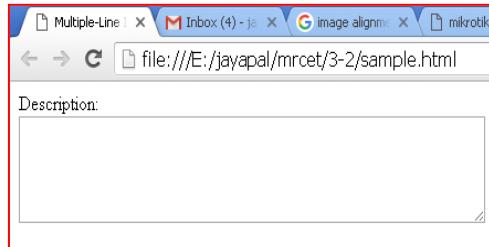
 <input type="password" name="psw">
</form>
```



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

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

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



### Checkboxes Controls:-

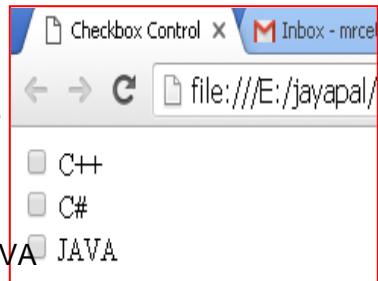
Checkboxes are used when more than one option is required to be selected. They are also created using **HTML <input>** tag but type attribute is set to checkbox.

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

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

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

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



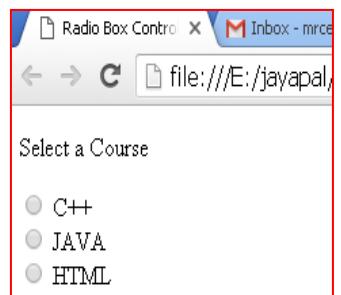
### Radio Button Control:-

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

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

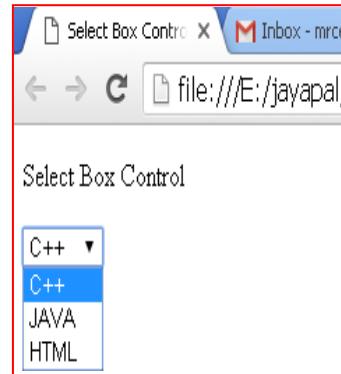
 <input type="radio" name="subject" value="JAVA"> JAVA

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



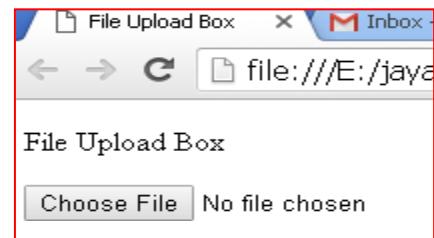
**Select Box Controls :-** A select box, also called drop down box which provides option to list down various options in the form of drop down list, from where a user can select one or more options.

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



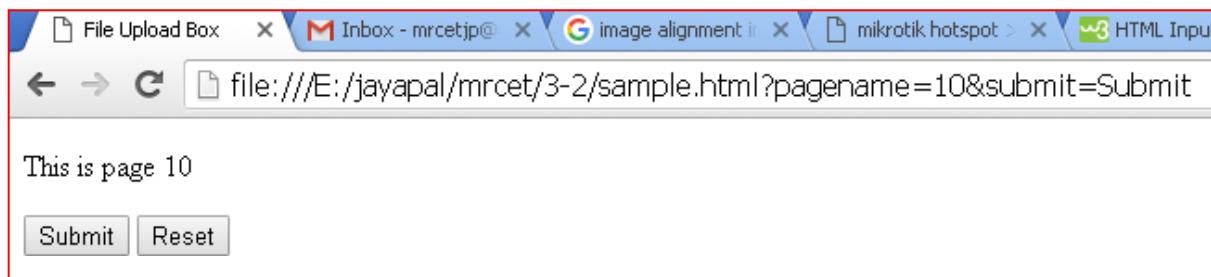
**File Select boxes:-** If you want to allow a user to upload a file to your web site, you will need to use a file upload box, also known as a file select box. This is also created using the <input> element but type attribute is set to **file**.

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



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

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

**Button Controls:-**

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

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

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



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

To use frames on a page we use `<frameset>` tag instead of `<body>` tag. The `<frameset>` tag defines, how to divide the window into frames. The **rows** attribute of `<frameset>` tag defines

horizontal frames and **cols** attribute defines vertical frames. Each frame is indicated by `<frame>` tag and it defines which HTML document shall open into the frame.

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

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

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



**CSS** stands for Cascading Style Sheets

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

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

CSS can be added to HTML elements in 3 ways:

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

### Inline CSS

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

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

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

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

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



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

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

#### External CSS:-

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

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

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

Here is how the "styles.css" looks:

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

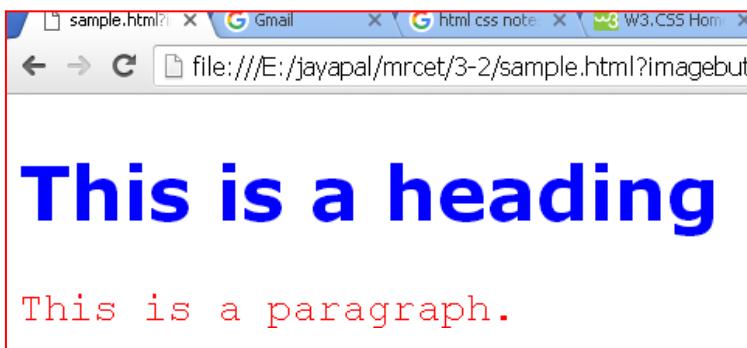


**CSS Fonts:** The CSS **color** property defines the text color to be used.

The CSS **font-family** property defines the font to be used.

The CSS **font-size** property defines the text size to be used.

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



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

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

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

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



## JavaScript:

### What is JavaScript?

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

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

### Difference between JavaScript and Java

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

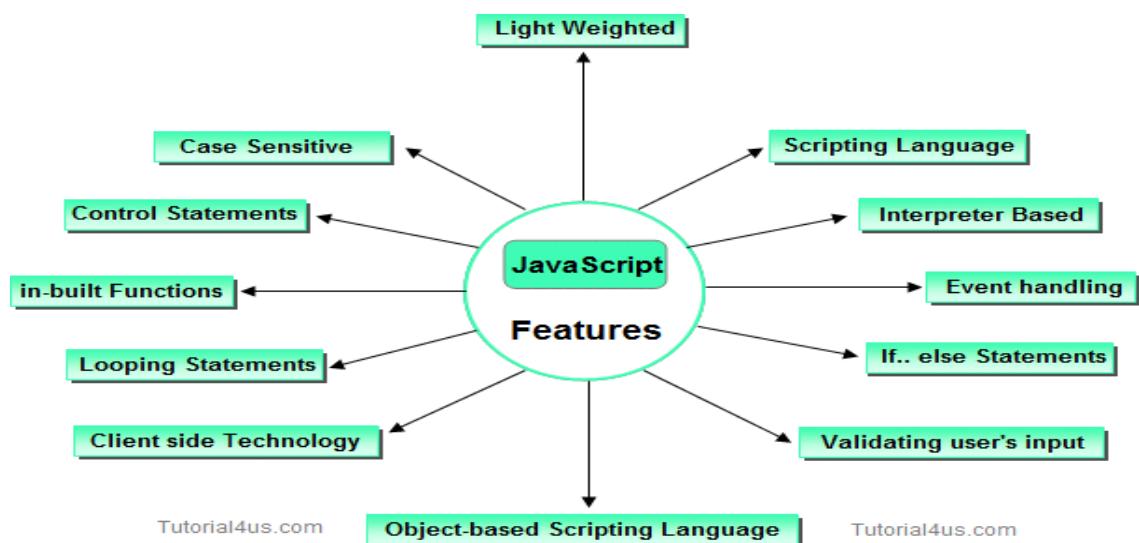
There are no relationship between Java & JavaScript. JavaScript is a scripting language that is always dependent on HTML language. It uses CSS commands. It is mainly used for creating DHTML pages & validating data. This is called client side validation.

### Why we Use JavaScript?

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

### Features of JavaScript

JavaScript is a client side technology, it is mainly used for client side validation, but it has many features which are given below;



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

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

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

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

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

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

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

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

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

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

→ JavaScript is **case sensitive**.

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

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

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

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

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

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

1) **Language attribute:-**

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

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

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

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

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

### Script in head section

```
<html>
 <head>
 <script type="text/JavaScript">
 iptcodehere
 </script>
 </head>
 <body>
 </body>
 </html>
```

### Script in body section

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

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

**Ex:** -

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

### Program: -

```
<html>
 <head>
 <script language="JavaScript">
 document.write("hai my name is Mamatha")
 </script>
 </head>
 <body text="red">
```

```
<marquee>
<script language="JavaScript">
document.write("hai my name is Srikanth")
</script></marquee>
</body>
</html>
```

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

**hai my name is Srikanth**

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

→ There are 2 ways of executing script code

- 1) directexecute
- 2) to execute script codedynamically

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

**Ex:** -

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

**O/P:** - to executescriptcode:

click me

**To execute scriptcode:**

touch me

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

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

**save:** - external.js

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

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

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

**O/P:-**

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

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

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

**Those the variable name „a“ is different from the variable named „A“.**

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

**→ ; is optional in general JavaScript.**

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

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

→ JavaScript ignore white space. In java script white space, tag space & empty lines are not preserved.

→ To display special symbols we use \.

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

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

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

```
vara; a=20;
varstr; str= "Sunil";
varc; c="a";
vard; d=30.7;
```

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

**c;** → not valid. In this solution var keyword must be declared.

→ During the script, we can change value of variable as well as type of value of variable.

**Ex:** -

```
a=20;
a=30.7;
```

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

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

```
{

}
```

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

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

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

**Ex:** -

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

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

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

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

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

**Ex:** -

```
<HTML>
<HEAD>
<TITLE> Function dynamically</TITLE>
<script language="JavaScript">
function add()
{
x=20
y=30
```

```
z=x+y
document.write("addition is :" +z);
}
</script>
</HEAD>
<BODY> to call function:
<input type="button" value="click here"
onclick="add()>"
```

```
</script>
</BODY>
</HTML>
```

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

**EVENTHANDLERS:** Events are not casesensitive.

**Java script events:** -

**Attribute**

	<b><u>The event occurs when...</u></b>
onclick	mouse click an object
ondblclick	mouse double clicks
onmouseover	a mouse cursor on touch here
onmousedown	a mouse button ispressed
onmousemove	the mouse is moved
onmouseout	the mouse is moved out anelement
onmouseup	a mouse button isreleased
onkeydown	a keyboard key ispressed
onkeypress	a keyboard key is pressed or held down
onkeyup	a keyboard key isreleased
onfocus	an elements getfocus
onblur	an element losesfocus
onchange	the content of a fieldchange
onselect	text isselected
onload	a page or an image is finishedloading
onunload	the user exist thepage
onerror	an error occurs when loading a document or animage
onabort	loading an image isinterrupted
onresize	a window or frame is resized
onreset	the reset button is pressed
onsubmit	the submit button isclicked

**Ex:** -

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

```


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

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

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

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

```

```


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

</BODY>
</HTML>
```

**O/P:** -

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

**Program:** -

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

**O/P:** -

Enter Studentname:

Enter Studentname:   
 sname you have entered is : true



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

**1) alert popup box :-**

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

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

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

**Ex: -**

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

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

**O/P: -**



Result is: 60

**2) confirm popupbox:-**

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

**Syntax:-      window.confirm(“message?”);**

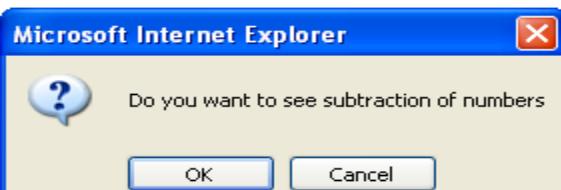
When the confirm box pop's up, user must click either ok or cancel buttons to proceed. If user clicks ok button it returns the boolean value true. If user clicks cancel button, it returns the boolean value false.

**Ex: -**

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

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

**O/P: -**



to see the o/p in pop up box:

result is :5

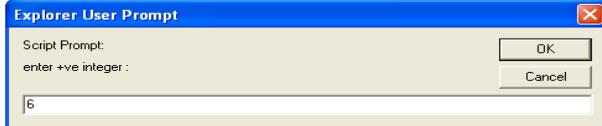
**3) Prompt popup box:-** It is useful to accept data from keyboard at runtime. Prompt box is created by prompt method of windowobject.

**window.prompt (“message”, “default text”);**

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

**Ex: -**

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



Script is turned off in the

browser and it can't be easily bypassed by malicious users. On the other hand, users will have to fill in the information without getting a response until they submit the form. This results in a slow response from the server.

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

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

### **CLIENT-SIDE VALIDATION**

Server-side validation is enough to have a successful and secure form validation. For better user experience, however, you might consider using client-side validation. This

**O/P: -**

### **FORM VALIDATION:**

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

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

### **SERVER-SIDE VALIDATION**

In the server-side validation, information is being sent to the server and validated using one of server-side languages. If the validation fails, the response is then sent back to the client, page that contains the web form is refreshed and a feedback is shown. This method is secure because it will work

type of validation is done on the client using script languages such as JavaScript. By using script languages user's input can be validated as they type. This means a more responsive, visually rich validation.

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

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

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

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

### Simple Example:

```
<html>
<head>
<title>FormValidation</title>
<script type="text/javascript">
<!--
// Form validation code will come here.
function validate()
{
var n = document.myForm.Name.value;
```

```
if(n == "" || (!isNaN(parseInt(n))) || n.length
< 3 || n.length >= 8)
{
alert("Please enter valid name and
minimum length 3 characters and maximum
length 8 characters !");
document.myForm.Name.focus();
return false;
}

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

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

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

var c = document.myForm.Country.value;
if(c == "-1")
{
alert("Please provide your country!");
```

```

return false;
}
return(true);
}

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

```

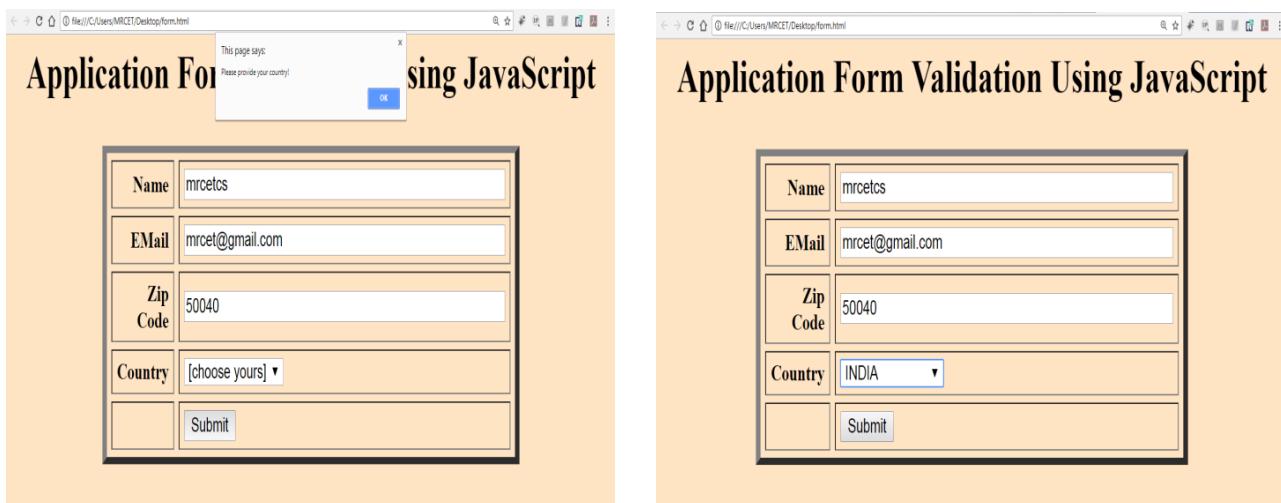
```

</td>
</tr>
<tr>
<td align="right"></td>
<td><input type="submit" value="Submit"
/></td>
</tr>
</table></form></body>
</html>

```

### **Output:**





JavaScript is an Object Oriented Programming (OOP) language. A programming language can be called object-oriented if it provides four basic capabilities to developers –

- **Encapsulation** – the capability to store related information, whether data or methods, together in an object.
- **Aggregation** – the capability to store one object inside another object.
- **Inheritance** – the capability of a class to rely upon another class (or number of classes) for some of its properties and methods.
- **Polymorphism** – the capability to write one function or method that works in a variety of different ways.

Objects are composed of attributes. If an attribute contains a function, it is considered to be a method of the object, otherwise the attribute is considered a property.

## Object Properties

Object properties can be any of the three primitive data types, or any of the abstract data types, such as another object. Object properties are usually variables that are used internally in the object's methods, but can also be globally visible variables that are used throughout the page.

The syntax for adding a property to an object is –

```
objectName.objectProperty = propertyName;
```

**For example** – The following code gets the document title using the "title" property of the **document** object.

```
var str = document.title;
```

## Object Methods

Methods are the functions that let the object do something or let something be done to it. There is a small difference between a function and a method – at a function is a standalone unit of statements and a method is attached to an object and can be referenced by the **this** keyword.

Methods are useful for everything from displaying the contents of the object to the screen to performing complex mathematical operations on a group of local properties and parameters.

**For example** – Following is a simple example to show how to use the **write()** method of document object to write any content on the document.

```
document.write("This is test");
```

Web Programming

Page

## User-Defined Objects

All user-defined objects and built-in objects are descendants of an object called **Object**.

### *The new Operator*

The **new** operator is used to create an instance of an object. To create an object, the **new** operator is followed by the constructor method.

In the following example, the constructor methods are **Object()**, **Array()**, and **Date()**. These constructors are built-in JavaScript functions.

```
var employee = new Object();
var books = new Array("C++", "Perl", "Java");
var day = new Date("August 15, 1947");
```

### *The Object() Constructor*

A constructor is a function that creates and initializes an object. JavaScript provides a special constructor function called **Object()** to build the object. The return value of the **Object()** constructor is assigned to a variable.

The variable contains a reference to the new object. The properties assigned to the object are not variables and are not defined with the **var** keyword.

### *Example 1*

Try the following example; it demonstrates how to create an Object.

[Live Demo](#)

```
<html>
 <head>
 <title>User-defined objects</title>
 <script type = "text/javascript">
 var book = new Object(); // Create the object
 book.subject = "Perl"; // Assign properties to the object
 book.author = "Mohtashim";
 </script>
 </head>

 <body>
 <script type = "text/javascript">
 document.write("Book name is : " + book.subject + "
");
 document.write("Book author is : " + book.author + "
");
 </script>
 </body>
</html>
```

### *Output*

Book name is : Perl  
Book author is : Mohtashim

### *Example 2*

This example demonstrates how to create an object with a User-Defined Function. Here **this** keyword is used to refer to the object that has been passed to a function.

[Live Demo](#)

```
<html>
 <head>
 <title>User-defined objects</title>
 <script type = "text/javascript">
 function book(title, author) {
 this.title = title;
 this.author = author;
 }
 </script>
 </head>
```

```

 }
 </script>
</head>

<body>
<script type = "text/javascript">
 var myBook = new book("Perl", "Mohtashim");
 document.write("Book title is : " + myBook.title + "
");
 document.write("Book author is : " + myBook.author + "
");
</script>
</body>
</html>

```

***Output***

Book title is : Perl  
 Book author is : Mohtashim

**Defining Methods for an Object**

The previous examples demonstrate how the constructor creates the object and assigns properties. But we need to complete the definition of an object by assigning methods to it.

***Example***

Try the following example; it shows how to add a function along with an object.

[Live Demo](#)

```

<html>

 <head>
 <title>User-defined objects</title>
 <script type = "text/javascript">
 // Define a function which will work as a method
 function addPrice(amount) {
 this.price = amount;
 }

 function book(title, author) {
 this.title = title;
 this.author = author;
 this.addPrice = addPrice; // Assign that method as property.
 }
 </script>
 </head>

 <body>
 <script type = "text/javascript">
 var myBook = new book("Perl", "Mohtashim");
 myBook.addPrice(100);

 document.write("Book title is : " + myBook.title + "
");
 document.write("Book author is : " + myBook.author + "
");
 document.write("Book price is : " + myBook.price + "
");
 </script>
 </body>
</html>

```

***Output***

Book title is : Perl  
 Book author is : Mohtashim  
 Book price is : 100

**The 'with' Keyword**

The '**with**' keyword is used as a kind of shorthand for referencing an object's properties or methods.

The object specified as an argument to **with** becomes the default object for the duration of the block that follows. The properties and methods for the object can be used without naming the object.

### Syntax

The syntax for with object is as follows -

```
with (object) {
 properties used without the object name and dot
}
```

### Example

Try the following example.

[Live Demo](#)

```
<html>
 <head>
 <title>User-defined objects</title>
 <script type = "text/javascript">
 // Define a function which will work as a method
 function addPrice(amount) {
 with(this) {
 price = amount;
 }
 }
 function book(title, author) {
 this.title = title;
 this.author = author;
 this.price = 0;
 this.addPrice = addPrice; // Assign that method as property.
 }
 </script>
 </head>

 <body>
 <script type = "text/javascript">
 var myBook = new book("Perl", "Mohtashim");
 myBook.addPrice(100);

 document.write("Book title is : " + myBook.title + "
");
 document.write("Book author is : " + myBook.author + "
");
 document.write("Book price is : " + myBook.price + "
");
 </script>
 </body>
</html>
```

### Output

Book title is : Perl  
 Book author is : Mohtashim  
 Book price is : 100

## JavaScript Native Objects

JavaScript has several built-in or native objects. These objects are accessible anywhere in your program and will work the same way in any browser running in any operating system.

Here is the list of all important JavaScript Native Objects -

- [JavaScript Number Object](#)
- [JavaScript Boolean Object](#)

- [JavaScript String Object](#)
- [JavaScript Array Object](#)
- [JavaScript Date Object](#)
- [JavaScript Math Object](#)
- [JavaScript RegExp Object](#)

## JavaScript - The Number Object

The **Number** object represents numerical date, either integers or floating-point numbers. In general, you do not need to worry about **Number** objects because the browser automatically converts number literals to instances of the number class.

### Syntax

The syntax for creating a **number** object is as follows –

```
var val = new Number(number);
```

In the place of number, if you provide any non-number argument, then the argument cannot be converted into a number, it returns **NaN** (Not-a-Number).

### Number Properties

Here is a list of each property and their description.

Sr.No.	Property & Description
1	<u>MAX_VALUE</u> The largest possible value a number in JavaScript can have 1.7976931348623157E+308
2	<u>MIN_VALUE</u> The smallest possible value a number in JavaScript can have 5E-324
3	<u>NaN</u> Equal to a value that is not a number.
4	<u>NEGATIVE_INFINITY</u> A value that is less than MIN_VALUE.
5	<u>POSITIVE_INFINITY</u> A value that is greater than MAX_VALUE

6	<u>prototype</u> A static property of the Number object. Use the prototype property to assign new properties and methods to the Number object in the current document
7	<u>constructor</u> Returns the function that created this object's instance. By default this is the Number object.

In the following sections, we will take a few examples to demonstrate the properties of Number.

## Number Methods

The Number object contains only the default methods that are a part of every object's definition.

Sr.No.	Method & Description
1	<u>toExponential()</u> Forces a number to display in exponential notation, even if the number is in the range in which JavaScript normally uses standard notation.
2	<u>toFixed()</u> Formats a number with a specific number of digits to the right of the decimal.
3	<u>toLocaleString()</u> Returns a string value version of the current number in a format that may vary according to a browser's local settings.
4	<u>toPrecision()</u> Defines how many total digits (including digits to the left and right of the decimal) to display of a number.
5	<u>toString()</u> Returns the string representation of the number's value.
6	<u>valueOf()</u> Returns the number's value.

# JavaScript - The Boolean Object

Advertisements

[Previous Page](#)

[Next Page](#)

The **Boolean** object represents two values, either "true" or "false". If *value* parameter is omitted or is 0, -0, null, false, **NaN**, undefined, or the empty string (""), the object has an initial value of false.

## Syntax

Use the following syntax to create a **boolean** object.

```
var val = new Boolean(value);
```

## Boolean Properties

Here is a list of the properties of Boolean object –

Sr.No.	Property & Description
1	<u>constructor</u> Returns a reference to the Boolean function that created the object.
2	<u>prototype</u> The prototype property allows you to add properties and methods to an object.

In the following sections, we will have a few examples to illustrate the properties of Boolean object.

## Boolean Methods

Here is a list of the methods of Boolean object and their description.

Sr.No.	Method & Description
1	<u>toSource()</u> Returns a string containing the source of the Boolean object; you can use this string to create an equivalent object.
2	<u>toString()</u>

	Returns a string of either "true" or "false" depending upon the value of the object.
3	<u>valueOf()</u> Returns the primitive value of the Boolean object

## JavaScript - The Strings Object

Advertisements

[Previous Page](#)

[Next Page](#)

The **String** object lets you work with a series of characters; it wraps Javascript's string primitive data type with a number of helper methods.

As JavaScript automatically converts between string primitives and String objects, you can call any of the helper methods of the String object on a string primitive.

### Syntax

Use the following syntax to create a String object –

```
var val = new String(string);
```

The **String** parameter is a series of characters that has been properly encoded.

### String Properties

Here is a list of the properties of String object and their description.

Sr.No.	Property & Description
1	<u>constructor</u> Returns a reference to the String function that created the object.
2	<u>length</u> Returns the length of the string.

3      [prototype](#)

The prototype property allows you to add properties and methods to an object.

In the following sections, we will have a few examples to demonstrate the usage of String properties.

## String Methods

Here is a list of the methods available in String object along with their description.

Sr.No.	Method & Description
1	<u><a href="#">charAt()</a></u> Returns the character at the specified index.
2	<u><a href="#">charCodeAt()</a></u> Returns a number indicating the Unicode value of the character at the given index.
3	<u><a href="#">concat()</a></u> Combines the text of two strings and returns a new string.
4	<u><a href="#">indexOf()</a></u> Returns the index within the calling String object of the first occurrence of the specified value, or -1 if not found.
5	<u><a href="#">lastIndexOf()</a></u> Returns the index within the calling String object of the last occurrence of the specified value, or -1 if not found.
6	<u><a href="#">localeCompare()</a></u> Returns a number indicating whether a reference string comes before or after or is the same as the given string in sort order.
7	<u><a href="#">match()</a></u> Used to match a regular expression against a string.
8	<u><a href="#">replace()</a></u> Used to find a match between a regular expression and a string, and to replace

	the matched substring with a new substring.
9	<b><u>search()</u></b> Executes the search for a match between a regular expression and a specified string.
10	<b><u>slice()</u></b> Extracts a section of a string and returns a new string.
11	<b><u>split()</u></b> Splits a String object into an array of strings by separating the string into substrings.
12	<b><u>substr()</u></b> Returns the characters in a string beginning at the specified location through the specified number of characters.
13	<b><u>substring()</u></b> Returns the characters in a string between two indexes into the string.
14	<b><u>toLocaleLowerCase()</u></b> The characters within a string are converted to lower case while respecting the current locale.
15	<b><u>toLocaleUpperCase()</u></b> The characters within a string are converted to upper case while respecting the current locale.
16	<b><u>toLowerCase()</u></b> Returns the calling string value converted to lower case.
17	<b><u>toString()</u></b> Returns a string representing the specified object.
18	<b><u>toUpperCase()</u></b> Returns the calling string value converted to uppercase.
19	<b><u>valueOf()</u></b>

	Returns the primitive value of the specified object.
--	------------------------------------------------------

## String HTML Wrappers

Here is a list of the methods that return a copy of the string wrapped inside an appropriate HTML tag.

Sr.No.	Method & Description
1	<u><a href="#">anchor()</a></u> Creates an HTML anchor that is used as a hypertext target.
2	<u><a href="#">big()</a></u> Creates a string to be displayed in a big font as if it were in a <b>&lt;big&gt;</b> tag.
3	<u><a href="#">blink()</a></u> Creates a string to blink as if it were in a <b>&lt;blink&gt;</b> tag.
4	<u><a href="#">bold()</a></u> Creates a string to be displayed as bold as if it were in a <b>&lt;b&gt;</b> tag.
5	<u><a href="#">fixed()</a></u> Causes a string to be displayed in fixed-pitch font as if it were in a <b>&lt;tt&gt;</b> tag
6	<u><a href="#">fontcolor()</a></u> Causes a string to be displayed in the specified color as if it were in a <b>&lt;font color="color"&gt;</b> tag.
7	<u><a href="#">fontsize()</a></u> Causes a string to be displayed in the specified font size as if it were in a <b>&lt;font size="size"&gt;</b> tag.
8	<u><a href="#">italics()</a></u> Causes a string to be italic, as if it were in an <b>&lt;i&gt;</b> tag.
9	<u><a href="#">link()</a></u> Creates an HTML hypertext link that requests another URL.

10	<u>small()</u> Causes a string to be displayed in a small font, as if it were in a <small> tag.
11	<u>strike()</u> Causes a string to be displayed as struck-out text, as if it were in a <strike> tag.
12	<u>sub()</u> Causes a string to be displayed as a subscript, as if it were in a <sub> tag
13	<u>sup()</u> Causes a string to be displayed as a superscript, as if it were in a <sup> tag

## JavaScript - The Arrays Object

Advertisements

[Previous Page](#)

[Next Page](#)

The **Array** object lets you store multiple values in a single variable. It stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

### Syntax

Use the following syntax to create an **Array** object –

```
var fruits = new Array("apple", "orange", "mango");
```

The **Array** parameter is a list of strings or integers. When you specify a single numeric parameter with the Array constructor, you specify the initial length of the array. The maximum length allowed for an array is 4,294,967,295.

You can create array by simply assigning values as follows –

```
var fruits = ["apple", "orange", "mango"];
```

You will use ordinal numbers to access and to set values inside an array as follows.

fruits[0] is the first element  
 fruits[1] is the second element  
 fruits[2] is the third element

### Array Properties

Here is a list of the properties of the Array object along with their description.

Sr.No.	Property & Description
1	<u>constructor</u> Returns a reference to the array function that created the object.
2	<b>index</b> The property represents the zero-based index of the match in the string
3	<b>input</b> This property is only present in arrays created by regular expression matches.
4	<u>length</u> Reflects the number of elements in an array.
5	<u>prototype</u> The prototype property allows you to add properties and methods to an object.

In the following sections, we will have a few examples to illustrate the usage of Array properties.

## Array Methods

Here is a list of the methods of the Array object along with their description.

Sr.No.	Method & Description
1	<u>concat()</u> Returns a new array comprised of this array joined with other array(s) and/or value(s).
2	<u>every()</u> Returns true if every element in this array satisfies the provided testing function.
3	<u>filter()</u> Creates a new array with all of the elements of this array for which the provided filtering function returns true.

4	<u>forEach()</u> Calls a function for each element in the array.
5	<u>indexOf()</u> Returns the first (least) index of an element within the array equal to the specified value, or -1 if none is found.
6	<u>join()</u> Joins all elements of an array into a string.
7	<u>lastIndexOf()</u> Returns the last (greatest) index of an element within the array equal to the specified value, or -1 if none is found.
8	<u>map()</u> Creates a new array with the results of calling a provided function on every element in this array.
9	<u>pop()</u> Removes the last element from an array and returns that element.
10	<u>push()</u> Adds one or more elements to the end of an array and returns the new length of the array.
11	<u>reduce()</u> Apply a function simultaneously against two values of the array (from left-to-right) as to reduce it to a single value.
12	<u>reduceRight()</u> Apply a function simultaneously against two values of the array (from right-to-left) as to reduce it to a single value.
13	<u>reverse()</u> Reverses the order of the elements of an array -- the first becomes the last, and the last becomes the first.
14	<u>shift()</u>

	Removes the first element from an array and returns that element.
15	<u><a href="#">slice()</a></u> Extracts a section of an array and returns a new array.
16	<u><a href="#">some()</a></u> Returns true if at least one element in this array satisfies the provided testing function.
17	<u><a href="#">toSource()</a></u> Represents the source code of an object
18	<u><a href="#">sort()</a></u> Sorts the elements of an array
19	<u><a href="#">splice()</a></u> Adds and/or removes elements from an array.
20	<u><a href="#">toString()</a></u> Returns a string representing the array and its elements.
21	<u><a href="#">unshift()</a></u> Adds one or more elements to the front of an array and returns the new length of the array.

## JavaScript - The Date Object

Advertisements

[Previous Page](#)

[Next Page](#)

The Date object is a datatype built into the JavaScript language. Date objects are created with the **new Date( )** as shown below.

Once a Date object is created, a number of methods allow you to operate on it. Most methods simply allow you to get and set the year, month, day, hour, minute, second, and millisecond fields of the object, using either local time or UTC (universal, or GMT) time.

The ECMAScript standard requires the Date object to be able to represent any date and time, to millisecond precision, within 100 million days before or after 1/1/1970. This is a range of plus or minus 273,785 years, so JavaScript can represent date and time till the year 275755.

## Syntax

You can use any of the following syntaxes to create a Date object using Date() constructor.

```
new Date()
new Date(milliseconds)
new Date(datestring)
new Date(year,month,date[,hour,minute,second,millisecond])
```

**Note** – Parameters in the brackets are always optional.

Here is a description of the parameters –

- **No Argument** – With no arguments, the Date() constructor creates a Date object set to the current date and time.
- **milliseconds** – When one numeric argument is passed, it is taken as the internal numeric representation of the date in milliseconds, as returned by the getTime() method. For example, passing the argument 5000 creates a date that represents five seconds past midnight on 1/1/70.
- **datestring** – When one string argument is passed, it is a string representation of a date, in the format accepted by the **Date.parse()** method.
- **7 arguments** – To use the last form of the constructor shown above. Here is a description of each argument –
  - **year** – Integer value representing the year. For compatibility (in order to avoid the Y2K problem), you should always specify the year in full; use 1998, rather than 98.
  - **month** – Integer value representing the month, beginning with 0 for January to 11 for December.
  - **date** – Integer value representing the day of the month.
  - **hour** – Integer value representing the hour of the day (24-hour scale).
  - **minute** – Integer value representing the minute segment of a time reading.
  - **second** – Integer value representing the second segment of a time reading.
  - **millisecond** – Integer value representing the millisecond segment of a time reading.

## Date Properties

Here is a list of the properties of the Date object along with their description.

Sr.No.	Property & Description
1	<u>constructor</u>

	Specifies the function that creates an object's prototype.
2	<u>prototype</u> The prototype property allows you to add properties and methods to an object

In the following sections, we will have a few examples to demonstrate the usage of different Date properties.

## Date Methods

Here is a list of the methods used with **Date** and their description.

Sr.No.	Method & Description
1	<u>Date()</u> Returns today's date and time
2	<u>getDate()</u> Returns the day of the month for the specified date according to local time.
3	<u>getDay()</u> Returns the day of the week for the specified date according to local time.
4	<u>getFullYear()</u> Returns the year of the specified date according to local time.
5	<u>getHours()</u> Returns the hour in the specified date according to local time.
6	<u>getMilliseconds()</u> Returns the milliseconds in the specified date according to local time.
7	<u>getMinutes()</u> Returns the minutes in the specified date according to local time.
8	<u>getMonth()</u> Returns the month in the specified date according to local time.

9	<u>getSeconds()</u> Returns the seconds in the specified date according to local time.
10	<u>getTime()</u> Returns the numeric value of the specified date as the number of milliseconds since January 1, 1970, 00:00:00 UTC.
11	<u>getTimezoneOffset()</u> Returns the time-zone offset in minutes for the current locale.
12	<u>getUTCDate()</u> Returns the day (date) of the month in the specified date according to universal time.
13	<u>getUTCDay()</u> Returns the day of the week in the specified date according to universal time.
14	<u>getUTCFullYear()</u> Returns the year in the specified date according to universal time.
15	<u>getUTCHours()</u> Returns the hours in the specified date according to universal time.
16	<u>getUTCMilliseconds()</u> Returns the milliseconds in the specified date according to universal time.
17	<u>getUTCMinutes()</u> Returns the minutes in the specified date according to universal time.
18	<u>getUTCMonth()</u> Returns the month in the specified date according to universal time.
19	<u>getUTCSeconds()</u> Returns the seconds in the specified date according to universal time.
20	<u>getYear()</u> <b>Deprecated</b> - Returns the year in the specified date according to local time. Use

	getFullYear instead.
21	<u> setDate()</u> Sets the day of the month for a specified date according to local time.
22	<u> setFullYear()</u> Sets the full year for a specified date according to local time.
23	<u> setHours()</u> Sets the hours for a specified date according to local time.
24	<u> setMilliseconds()</u> Sets the milliseconds for a specified date according to local time.
25	<u> setMinutes()</u> Sets the minutes for a specified date according to local time.
26	<u> setMonth()</u> Sets the month for a specified date according to local time.
27	<u> setSeconds()</u> Sets the seconds for a specified date according to local time.
28	<u> setTime()</u> Sets the Date object to the time represented by a number of milliseconds since January 1, 1970, 00:00:00 UTC.
29	<u> setUTCDate()</u> Sets the day of the month for a specified date according to universal time.
30	<u> setUTCFullYear()</u> Sets the full year for a specified date according to universal time.
31	<u> setUTCHours()</u> Sets the hour for a specified date according to universal time.
32	<u> setUTCMilliseconds()</u>

	Sets the milliseconds for a specified date according to universal time.
33	<u><a href="#">setUTCMilliseconds()</a></u> Sets the minutes for a specified date according to universal time.
34	<u><a href="#">setUTCMonth()</a></u> Sets the month for a specified date according to universal time.
35	<u><a href="#">setUTCSeconds()</a></u> Sets the seconds for a specified date according to universal time.
36	<u><a href="#">setYear()</a></u> <b>Deprecated</b> - Sets the year for a specified date according to local time. Use setFullYear instead.
37	<u><a href="#">toDateString()</a></u> Returns the "date" portion of the Date as a human-readable string.
38	<u><a href="#">toGMTString()</a></u> <b>Deprecated</b> - Converts a date to a string, using the Internet GMT conventions. Use toUTCString instead.
39	<u><a href="#">toLocaleDateString()</a></u> Returns the "date" portion of the Date as a string, using the current locale's conventions.
40	<u><a href="#">toLocaleFormat()</a></u> Converts a date to a string, using a format string.
41	<u><a href="#">toLocaleString()</a></u> Converts a date to a string, using the current locale's conventions.
42	<u><a href="#">toLocaleTimeString()</a></u> Returns the "time" portion of the Date as a string, using the current locale's conventions.
43	<u><a href="#">toSource()</a></u>

	Returns a string representing the source for an equivalent Date object; you can use this value to create a new object.
44	<u><a href="#">toString()</a></u> Returns a string representing the specified Date object.
45	<u><a href="#">toTimeString()</a></u> Returns the "time" portion of the Date as a human-readable string.
46	<u><a href="#">toUTCString()</a></u> Converts a date to a string, using the universal time convention.
47	<u><a href="#">valueOf()</a></u> Returns the primitive value of a Date object.

Converts a date to a string, using the universal time convention.

## Date Static Methods

In addition to the many instance methods listed previously, the Date object also defines two static methods. These methods are invoked through the Date() constructor itself.

Sr.No.	Method & Description
1	<u><a href="#">Date.parse()</a></u> Parses a string representation of a date and time and returns the internal millisecond representation of that date.
2	<u><a href="#">Date.UTC()</a></u> Returns the millisecond representation of the specified UTC date and time.

## JavaScript - The Math Object

Advertisements

[Previous Page](#)[Next Page](#)

The **math** object provides you properties and methods for mathematical constants and functions. Unlike other global objects, **Math** is not a constructor. All the properties and methods of **Math** are static and can be called by using Math as an object without creating it.

Thus, you refer to the constant **pi** as **Math.PI** and you call the *sine* function as **Math.sin(x)**, where x is the method's argument.

## Syntax

The syntax to call the properties and methods of Math are as follows

```
var pi_val = Math.PI;
var sine_val = Math.sin(30);
```

## Math Properties

Here is a list of all the properties of Math and their description.

Sr.No.	Property & Description
1	<u>E</u> Euler's constant and the base of natural logarithms, approximately 2.718.
2	<u>LN2</u> Natural logarithm of 2, approximately 0.693.
3	<u>LN10</u> Natural logarithm of 10, approximately 2.302.
4	<u>LOG2E</u> Base 2 logarithm of E, approximately 1.442.
5	<u>LOG10E</u> Base 10 logarithm of E, approximately 0.434.
6	<u>PI</u> Ratio of the circumference of a circle to its diameter, approximately 3.14159.
7	<u>SQRT1_2</u> Square root of 1/2; equivalently, 1 over the square root of 2, approximately 0.707.

8	<u>SQRT2</u>
	Square root of 2, approximately 1.414.

In the following sections, we will have a few examples to demonstrate the usage of Math properties.

## Math Methods

Here is a list of the methods associated with Math object and their description

Sr.No.	<b>Method &amp; Description</b>
1	<u>abs()</u> Returns the absolute value of a number.
2	<u>acos()</u> Returns the arccosine (in radians) of a number.
3	<u>asin()</u> Returns the arcsine (in radians) of a number.
4	<u>atan()</u> Returns the arctangent (in radians) of a number.
5	<u>atan2()</u> Returns the arctangent of the quotient of its arguments.
6	<u>ceil()</u> Returns the smallest integer greater than or equal to a number.
7	<u>cos()</u> Returns the cosine of a number.
8	<u>exp()</u> Returns $E^N$ , where N is the argument, and E is Euler's constant, the base of the natural logarithm.
9	<u>floor()</u>

	Returns the largest integer less than or equal to a number.
10	<u><a href="#">log()</a></u> Returns the natural logarithm (base E) of a number.
11	<u><a href="#">max()</a></u> Returns the largest of zero or more numbers.
12	<u><a href="#">min()</a></u> Returns the smallest of zero or more numbers.
13	<u><a href="#">pow()</a></u> Returns base to the exponent power, that is, base exponent.
14	<u><a href="#">random()</a></u> Returns a pseudo-random number between 0 and 1.
15	<u><a href="#">round()</a></u> Returns the value of a number rounded to the nearest integer.
16	<u><a href="#">sin()</a></u> Returns the sine of a number.
17	<u><a href="#">sqrt()</a></u> Returns the square root of a number.
18	<u><a href="#">tan()</a></u> Returns the tangent of a number.
19	<u><a href="#">toSource()</a></u> Returns the string "Math".

**NAMES**

`name`' is **the property of the window object of the browser**. It is a built-in property in JavaScript.

# JavaScript Literals

JavaScript Literals are constant values that can be assigned to the variables that are called literals or constants.

JavaScript Literals are syntactic representations for different types of data like numeric, string, Boolean, array, etc data. Literals in JavaScript provide a means of representing particular or some specific values in our program.

Consider an example, var name = “john”, a string variable named name is declared and assigned a string value “john”. The literal “john” represents, the value john for the variable name. There are different types of literals that are supported by JavaScript.

## ***What are the Types of JavaScript Literals?***

Javascript literals hold different types of values. Examples of JavaScript Literals are given below:

### **1. Integer Literals**

Integer literals are numbers, must have minimum one digit (0-9). No blank or comma is allowed within an integer.

It can store positive numbers or negative numbers. In integers, literals in JavaScript can be supported in three different bases. The base 10 that is Decimal (Decimal numbers contain digits (0,9) ) examples for Decimal numbers are 234, -56, 10060. Second is base 8 that is Octal (Octal numbers contains digits (0,7) and leading 0 indicates the number is octal), 0X 073, -089, 02003. Third is base 16 that is Hexadecimal numbers (Hexadecimal numbers contains (0,9) digits and (A,F) or (a, f) letters and leading 0x or 0X indicates the number is hexadecimal), examples for hexadecimal numbers are 0X8b, – 0X89, 0X2003.

Let us understand with example code.

### **Example:**

```
<!DOCTYPE html>

<html>

<head>

<meta charset= "utf-8" >

<title> This is an example for numeric literals </title>

</head>

<body>

<h1>JavaScript Numbers </h1>

<p> Number can be written of any base.</p>

Decimal number : <b id="no1">

Octal number : <b id="no2">

Hexadecimal number : <b id="no3">

<script>

document.getElementById("no1").innerHTML = 100.25;

</script>

<script>

document.getElementById("no2").innerHTML = 073;

</script>

<script>

document.getElementById("no3").innerHTML = 0X8b;
```

```
</script>
```

```
</body>
```

```
</html>
```

## Output:

### 2. Floating Number Literals

Floating numbers are decimal numbers or fraction numbers or even can have an exponent part as well. Examples for hexadecimal numbers are 78.90, -234.90, 78.6e4 etc.

Let us understand with example code.

## Example:

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<meta charset= "utf-8" >
```

```
<title> This is an example for float literals </title>
```

```
</head>
```

```
<body>
```

```
<h1>JavaScript Float </h1>
```

```
<p> Float Examples are : </p>
```

```
1. <b id="no1">

```

```
2. <b id="no2">

```

```
3. <b id="no3">

```

```
<script>
```

```
document.getElementById("no1").innerHTML = 100.25;
```

```
</script>
```

```
<script>
```

```
document.getElementById("no2").innerHTML = -78.34;
```

```
</script>
```

```
<script>
```

```
document.getElementById("no3").innerHTML = 56e4;
```

```
</script>
```

```
</body>
```

```
</html>
```

### 3. String Literals

A string literals are a sequence of zero or more characters. A string literals are either enclosed in the single quotation or double quotation as ( ‘ ) and ( “ ) respectively and to concatenate two or more string we can use + operator. Examples for string are “hello”, “hello world”, “123”, “hello” + “world” etc.

List of special characters that can be used in JavaScript string are.

- **\b:** Backspace.
- **\n:** New Line
- **\t:** Tab
- **\f:** Form Feed
- **\r:** Carriage Return
- **\\: Backslash Character (\)**
- **\': Single Quote**
- **\": Double Quote**

Let us understand with an example code –

#### Example:

```
<!DOCTYPE html>

<html>

<head>

<meta charset= "utf-8" >
```

```
<title> This is an example for float literals </title>
```

```
</head>
```

```
<body>
```

```
<h1>JavaScript String </h1>
```

```
<p> String Examples are : </p>
```

```
1. <b id="no1">

```

```
2. <b id="no2">

```

```
3. <b id="no3">

```

```
4. <b id="no4">

```

```
<script>
```

```
var str = "This is first string";
```

```
document.getElementById("no1").innerHTML = str;
```

```
</script>
```

```
<script>
```

```
var strobj = new String("This is string store as object");
```

```
document.getElementById("no2").innerHTML = strobj;
```

```
</script>
```

```
<script>
```

```
var str = "This is first string";
```

```
document.getElementById("no3").innerHTML = str.length;
```

```
</script>
```

```
<script>
```

```
var str = "This is first string";
```

```
document.getElementById("no4").innerHTML = str + " This is second string";
```

```
</script>
```

```
</body>
```

```
</html>
```

## Output:

### 4. Array Literals

Array literals are a list of expressions or other constant values, each of which expression known as an array element. An array literal contains a list of elements within square brackets ‘ [ ] ’ . If no value is passed when it creates an empty array with zero length. If elements are passed then its length is set to the number of elements passed. Examples for string are var color = [ ], var fruits = [“Apple”, “Orange”, “Mango”, “Banana”] (an array of four elements).

Let us understand with example code.

## Example:

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<meta charset= "utf-8" >
```

```
<title> This is an example for float literals </title>
```

```
</head>
```

```
<body>
```

```
<h1>JavaScript Array </h1>
```

```
<p> Array Examples are : </p>
```

```
1. <b id="no1">

```

```
2. <b id="no2">

```

```
3. <b id="no3">

```

```
4. <b id="no4">

```

```
<script>
```

```
var fruits = ["Apple", "Orange", "Mango", "Banana"];
```

```
document.getElementById("no1").innerHTML = fruits;
```

```
</script>
```

```
<script>
```

```
document.getElementById("no2").innerHTML = fruits[0];
```

```
</script>
```

```
<script>
```

```
document.getElementById("no3").innerHTML = fruits[fruits.length - 1];
```

```
</script>
```

```
<script>

document.getElementById("no4").innerHTML = fruits.length;

</script>

</body>

</html>
```

**Output:**

**5. Boolean Literals**

Boolean literals in JavaScript have only two literal values that are true and false.

Let us understand with an example code.

**Example:**

```
<!DOCTYPE html>

<html>

<head>

<meta charset= "utf-8" >
```

```
<title> This is an example for Boolean literals </title>
```

```
</head>
```

```
<body>
```

```
<h1>JavaScript Boolean </h1>
```

```
<p> Boolean Examples are : </p>
```

```
<script>
```

```
document.write('Boolean(12) is ' + Boolean(12));
```

```
document.write('
');
```

```
document.write('Boolean("Hello") is ' + Boolean("Hello"));
```

```
document.write('
');
```

```
document.write('Boolean(2 > 3) is ' + Boolean(2 > 3));
```

```
document.write('
');
```

```
document.write('Boolean(3 > 2) is ' + Boolean(3 > 2));
```

```
document.write('
');
```

```
document.write('Boolean(-12) is ' + Boolean(-12));
```

```
document.write('
');
```

```
document.write("Boolean('true') is " + Boolean('true'));
```

```
document.write('
');
```

```
document.write("Boolean('false') is " + Boolean('false'));
```

```
document.write('
');
```

```
document.write('Boolean(0) is ' + Boolean(0));
</script>
</body>
</html>
```

**Output:**

**6. Object Literals**

Object literals are collection zero or more key-value pairs of a comma-separated list, which are enclosed by a pair of curly braces ‘ { } ‘. Examples for object literal with declaration are var userObject = { }, var student = { f-name : “John”, l-name : “D”, “rno” : 23, “marks” : 60}

Let understand with an example code –

**Example:**

```
<!DOCTYPE html>
<html>
```

```
<head>
```

```
<meta charset= "utf-8" >
```

```
<title> This is an example for Object literals </title>
```

```
</head>
```

```
<body>
```

```
<h1>JavaScript Object </h1>
```

```
<p> Object Examples are :</p>
```

```
<p id= "no1"></p>
```

```
<script>
```

```
// Create an object:
```

```
var student = {firstName:"John", lastName:"D", "rno" : 23, "marks" : 60 };
```

```
// Displaying some data from the object:
```

```
document.getElementById("no1").innerHTML =
```

```
student.firstName + " got " + student.marks + " marks.";
```

```
</script>
```

```
</body>
```

```
</html>
```

## Expressions and operators

JavaScript has the following types of operators. This section describes the operators and contains information about operator precedence.

- Assignment operators
- Comparison operators
- Arithmetic operators
- Bitwise operators
- Logical operators
- String operators
- Conditional (ternary) operator
- Comma operator
- Unary operators
- Relational operators

JavaScript has both *binary* and *unary* operators, and one special ternary operator, the conditional operator. A binary operator requires two operands, one before the operator and one after the operator:

operand1 operator operand2

Copy to Clipboard

For example,  $3+4$  or  $x*y$ .

A unary operator requires a single operand, either before or after the operator:

operator operand

Copy to Clipboard

or

operand operator

Copy to Clipboard

For example, `x++` or `++x`.

### Assignment operators

An assignment operator assigns a value to its left operand based on the value of its right operand. The simple assignment operator is equal (`=`), which assigns the value of its right operand to its left operand. That is, `x = f()` is an assignment expression that assigns the value of `f()` to `x`.

There are also compound assignment operators that are shorthand for the operations listed in the following table:

Name	Shorthand operator	Meaning
<u>Assignment</u>	<code>x = f()</code>	<code>x =</code>
<u>Addition assignment</u>	<code>x += f()</code>	<code>x =</code>
<u>Subtraction assignment</u>	<code>x -= f()</code>	<code>x =</code>
<u>Multiplication assignment</u>	<code>x *= f()</code>	<code>x =</code>
<u>Division assignment</u>	<code>x /= f()</code>	<code>x =</code>
<u>Remainder assignment</u>	<code>x %= f()</code>	<code>x =</code>

Name	Shorthand operator	Meaning
<u>Exponentiation assignment</u>	<code>x **= f()</code>	<code>x =</code>
<u>Left shift assignment</u>	<code>x &lt;&lt;= f()</code>	<code>x =</code>
<u>Right shift assignment</u>	<code>x &gt;&gt;= f()</code>	<code>x =</code>
<u>Unsigned right shift assignment</u>	<code>x &gt;&gt;&gt;= f()</code>	<code>x =</code>
<u>Bitwise AND assignment</u>	<code>x &amp;= f()</code>	<code>x =</code>
<u>Bitwise XOR assignment</u>	<code>x ^= f()</code>	<code>x =</code>
<u>Bitwise OR assignment</u>	<code>x  = f()</code>	<code>x =</code>
<u>Logical AND assignment</u>	<code>x &amp;&amp;= f()</code>	<code>x &amp;</code>
<u>Logical OR assignment</u>	<code>x   = f()</code>	<code>x   </code>
<u>Logical nullish assignment</u>	<code>x ??= f()</code>	<code>x ??</code>

Assigning to properties

If a variable refers to an object, then the left-hand side of an assignment expression may make assignments to properties of that variable. For example:

```
let obj = {};
```

```
obj.x = 3;
```

```
console.log(obj.x); // Prints 3.
```

```
console.log(obj); // Prints { x: 3 }.
```

```
const key = "y";
```

```
obj[key] = 5;
```

```
console.log(obj[key]); // Prints 5.
```

```
console.log(obj); // Prints { x: 3, y: 5 }.
```

Copy to Clipboard

For more information about objects, read [Working with Objects](#).

Destructuring

For more complex assignments, the [destructuring assignment](#) syntax is a JavaScript expression that makes it possible to extract data from arrays or objects using a syntax that mirrors the construction of array and object literals.

```
var foo = ['one', 'two', 'three'];
```

// without destructuring

```
var one = foo[0];
```

```
var two = foo[1];
```

```
var three = foo[2];
```

// with destructuring

```
var [one, two, three] = foo;
```

Copy to Clipboard

Evaluation and nesting

In general, assignments are used within a variable declaration (i.e., with const, let, or var) or as standalone statements.

// Declares a variable x and initializes it to the result of f().

// The result of the x = f() assignment expression is discarded.

```
let x = f();
```

// Declares a variable x and initializes it to the result of f().

// The result of the x = g() assignment expression is discarded.

x = g(); Reassigns the variable x to the result of g().

Copy to Clipboard

However, like other expressions, assignment expressions like x = f() evaluate into a result value. Although this result value is usually not used, it can then be used by another expression.

Chaining assignments or nesting assignments in other expressions can result in surprising behavior. For this reason, some JavaScript style guides discourage chaining or nesting assignments). Nevertheless, assignment chaining and nesting may occur sometimes, so it is important to be able to understand how they work.

By chaining or nesting an assignment expression, its result can itself be assigned to another variable. It can be logged, it can be put inside an array literal or function call, and so on.

let x;

const y = (x = f()); // Or equivalently: const y = x = f();

console.log(y); // Logs the return value of the assignment x = f().

console.log(x = f()); // Logs the return value directly.

// An assignment expression can be nested in any place

```
// where expressions are generally allowed,

// such as as array literals' elements or as function calls' arguments.

console.log([0, x = f(), 0]);

console.log(f(0, x = f(), 0));
```

Copy to Clipboard

The evaluation result matches the expression to the right of the = sign in the “Meaning” column of the table above.

That means that x = f() evaluates into whatever f()'s result is, x += f() evaluates into the resulting sum x + f(), x \*\*= f() evaluates into the resulting power x \*\* y, and so on.

In the case of logical assignments, x &&= f(), x ||= f(), and x ??= f(), the return value is that of the logical operation without the assignment, so x && f(), x || f(), and x ?? f(), respectively.

When chaining these expressions without parentheses or other grouping operators like array literals, the assignment expressions are **grouped right to left** (they are right-associative), but they are **evaluated left to right**.

Note that, for all assignment operators other than = itself, the resulting values are always based on the operands’ values *before* the operation.

For example, assume that the following functions f and g and the variables x and y have been declared:

```
function f () {
```

```
 console.log('F!');
```

```
 return 2;
```

}

```
function g () {
```

```
 console.log('G!');
```

```
 return 3;
```

```
}
```

```
let x, y;
```

[Copy to Clipboard](#)

Consider these three examples:

```
y = x = g()
```

```
y = [f(), x = g()]
```

```
x[f()] = g()
```

[Copy to Clipboard](#)

Evaluation example 1

$y = x = f()$  is equivalent to  $y = (x = f())$ , because the assignment operator  $=$  is right-associative. However, it evaluates from left to right:

1. The assignment expression  $y = x = f()$  starts to evaluate.

1. The  $y$  on this assignment's left-hand side evaluates into a reference to the variable named  $y$ .

W    ~    ~

~

2. The assignment expression `x = f()` starts to evaluate.
    1. The `x` on this assignment's left-hand side evaluates into a reference to the variable named `y`.
    2. The function call `f()` prints "F!" to the console and then evaluates to the number 2.
    3. That 2 result from `f()` is assigned to `x`.
  3. The assignment expression `x = f()` has now finished evaluating; its result is the new value of `x`, which is 2.
  4. That 2 result in turn is also assigned to `y`.
2. The assignment expression `y = x = f()` has now finished evaluating; its result is the new value of `y` – which happens to be 2. `x` and `y` are assigned to 2, and the console has printed "F!".

## Evaluation example 2

`y = [ f(), x = g() ]` also evaluates from left to right:

1. The assignment expression `y = [ f(), x = g() ]` starts to evaluate.
  1. The `y` on this assignment's left-hand evaluates into a reference to the variable named `y`.
  2. The inner array literal `[ f(), x = g() ]` starts to evaluate.
  3. The function call `f()` prints "F!" to the console and then evaluates to the number 2.
  4. The assignment expression `x = g()` starts to evaluate.
    1. The `x` on this assignment's left-hand side evaluates into a reference to the variable named `x`.
    2. The function call `g()` prints "3!" to the console and then evaluates to the number 3.
    3. That 3 result from `f()` is assigned to `x`.
5. The assignment expression `x = g()` has now finished evaluating; its result is the new value of `x`, which is 3. That 3 result becomes the next element in the inner array literal (after the 2 from the `f()`).

6. The inner array literal [ f(), x = g() ] has now finished evaluating; its result is an array with two values: [ 2, 3 ].
  7. That [ 2, 3 ] array is now assigned to y.
2. The assignment expression y = [ f(), x = g() ] has now finished evaluating; its result is the new value of y – which happens to be [ 2, 3 ]. x is now assigned to 3, y is now assigned to [ 2, 3 ], and the console has printed "F!" then "G!".

## Evaluation example 2

x[f()] = g() also evaluates from left to right. (This example assumes that x is already assigned to some object. For more information about objects, read [Working with Objects](#).)

1. The assignment expression x[f()] = g() starts to evaluate.
  1. The x[f()] property access on this assignment's left-hand starts to evaluate.
    1. The x in this property access evaluates into a reference to the variable named x.
    2. Then the function call f() prints "F!" to the console and then evaluates to the number 2.
  2. The x[f()] property access on this assignment has now finished evaluating; its result is a variable property reference: x[2].
  3. Then the function call g() prints "G!" to the console and then evaluates to the number 3.
  4. That 3 is now assigned to x[2]. (This step will succeed only if x is assigned to an object.)
2. The assignment expression x[f()] = g() has now finished evaluating; its result is the new value of x[2] – which happens to be 3. x[2] is now assigned to 3, and the console has printed "F!" then "G!".

## Avoid assignment chains

Chaining assignments or nesting assignments in other expressions can result in surprising behavior. For this reason, chaining assignments in the same statement is discouraged).

In particular, putting a variable chain in a const, let, or var statement often does *not* work. Only the outermost/leftmost variable would get declared; any other variables within the assignment chain are *not* declared by the const/let/var statement. For example:

```
let z = y = x = f();
```

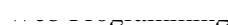
Copy to Clipboard

This statement seemingly declares the variables x, y, and z. However, it only actually declares the variable z. y and x are either invalid references to nonexistent variables (in strict mode) or, worse, would implicitly create global variables for x and y in sloppy mode.

### Comparison operators

A comparison operator compares its operands and returns a logical value based on whether the comparison is true. The operands can be numerical, string, logical, or object values. Strings are compared based on standard lexicographical ordering, using Unicode values. In most cases, if the two operands are not of the same type, JavaScript attempts to convert them to an appropriate type for the comparison. This behavior generally results in comparing the operands numerically. The sole exceptions to type conversion within comparisons involve the === and !== operators, which perform strict equality and inequality comparisons. These operators do not attempt to convert the operands to compatible types before checking equality. The following table describes the comparison operators in terms of this sample code:

```
var var1 = 3;
```



```
var var2 = 4;
```

Copy to Clipboard

## Comparison operators

Operator	Description
<u>Equal</u> (==)	Returns true if the operands are equal.
<u>Not equal</u> (!=)	Returns true if the operands are not equal.
<u>Strict equal</u> (===)	Returns true if the operands are equal and of the same type. See also <u>Object.is</u> and <u>sameness in JS</u> .
<u>Strict not equal</u> (!==)	Returns true if the operands are of the same type but not equal, or are of different type.
<u>Greater than</u> (>)	Returns true if the left operand is greater than the right operand.

## Comparison operators

Operator	Description
<u>Greater than or equal</u> ( $\geq$ )	Returns true if the left operand is greater than or equal to the right operand.
<u>Less than</u> ( $<$ )	Returns true if the left operand is less than the right operand.
<u>Less than or equal</u> ( $\leq$ )	Returns true if the left operand is less than or equal to the right operand.

**Note:**  $\Rightarrow$  is not a comparison operator but rather is the notation for Arrow functions.

## Arithmetic operators

An arithmetic operator takes numerical values (either literals or variables) as their operands and returns a single numerical value. The standard arithmetic operators are addition (+), subtraction (-), multiplication (\*), and division (/). These operators work as they do in most other programming languages when used with floating point numbers (in particular, note that division by zero produces Infinity). For example:

```
1 / 2; // 0.5
```

```
1 / 2 == 1.0 / 2.0; // this is true
```

[Copy to Clipboard](#)

In addition to the standard arithmetic operations (+, -, \*, /), JavaScript provides the arithmetic operators listed in the following table:

### **Arithmetic operators**

<b>Operator</b>	<b>Description</b>	<b>Example</b>
<u>Remainder (%)</u>	Binary operator. Returns the integer remainder of dividing the two operands.	12 % 5 returns 2
<u>Increment (++)</u>	Unary operator. Adds one to its operand. If used as a prefix operator (++x), returns the value of its operand after adding one; if used as a postfix operator (x++), returns the value of its operand before adding one.	If x is 3, then ++x returns 4, whereas x++ returns 3.
<u>Decrement (--)</u>	Unary operator. Subtracts one from its operand. The return value is analogous to that for the increment operator.	If x is 3, then --x returns 2, whereas x-- returns 3.
<u>Unary negation (-)</u>	Unary operator. Returns the negation of its operand.	If x is 3, then -x returns -3.
<u>Unary plus (+)</u>	Unary operator. Attempts to convert the operand to a number, if it is not already.	+ "3" returns 3. + true returns 1.
<u>Exponentiation operator (**)</u>	Calculates the base to the exponent power, that is, base <sup>exponent</sup>	2 ** 3 returns 8. 10 ** -1 returns 0.1.
<u>Bitwise operators</u>		

A bitwise operator treats their operands as a set of 32 bits (zeros and ones), rather than as decimal, hexadecimal, or octal numbers. For example, the decimal number nine has a binary representation of 1001. Bitwise operators perform their operations on such binary representations, but they return standard JavaScript numerical values.

The following table summarizes JavaScript's bitwise operators.

<b>Operator</b>	<b>Usage</b>	<b>Description</b>
<u>Bitwise AND</u>	<code>a &amp; b</code>	Returns a one in each bit position for which the corresponding bits of both operands are ones.
<u>Bitwise OR</u>	<code>a   b</code>	Returns a zero in each bit position for which the corresponding bits of both operands are zeros.
<u>Bitwise XOR</u>	<code>a ^ b</code>	Returns a zero in each bit position for which the corresponding bits are the same. [Returns a one in each bit position for which the corresponding bits are different.]
<u>Bitwise NOT</u>	<code>~ a</code>	Inverts the bits of its operand.
<u>Left shift</u>	<code>a &lt;&lt; b</code>	Shifts a in binary representation b bits to the left, shifting in zeros from the right.
<u>Sign-propagating right shift</u>	<code>a &gt;&gt; b</code>	Shifts a in binary representation b bits to the right, discarding bits shifted off.
<u>Zero-fill right shift</u>	<code>a &gt;&gt;&gt; b</code>	Shifts a in binary representation b bits to the right, discarding bits shifted off, and shifting in zeros from the left.

Bitwise logical operators

Conceptually, the bitwise logical operators work as follows:

- The operands are converted to thirty-two-bit integers and expressed by a series of bits (zeros and ones). Numbers with more than 32 bits get their most significant bits discarded. For example, the following integer with more than 32 bits will be converted to a 32 bit integer:
- Before: 1110 0110 1111 1010 0000 0000 0000 0110 0000 0000 0001
- After: 1010 0000 0000 0000 0110 0000 0000 0001
- Each bit in the first operand is paired with the corresponding bit in the second operand: first bit to first bit, second bit to second bit, and so on.
- The operator is applied to each pair of bits, and the result is constructed bitwise.

For example, the binary representation of nine is 1001, and the binary representation of fifteen is 1111. So, when the bitwise operators are applied to these values, the results are as follows:

<b>Expression</b>	<b>Result</b>	<b>Binary Description</b>
15 & 9	9	1111 & 1001 = 1001
15   9	15	1111   1001 = 1111
15 ^ 9	6	1111 ^ 1001 = 0110
$\sim 15$	-16	$\sim 0000 0000 \dots 0000 1111 = 1111 1111 \dots 1111 0000$
$\sim 9$	-10	$\sim 0000 0000 \dots 0000 1001 = 1111 1111 \dots 1111 0110$

Note that all 32 bits are inverted using the Bitwise NOT operator, and that values with the most significant (left-most) bit set to 1 represent negative numbers (two's-complement representation).  $\sim x$  evaluates to the same value that  $-x - 1$  evaluates to.

### Bitwise shift operators

The bitwise shift operators take two operands: the first is a quantity to be shifted, and the second specifies the number of bit positions by which the first operand is to be shifted. The direction of the shift operation is controlled by the operator used.

Shift operators convert their operands to thirty-two-bit integers and return a result of either type Number or BigInt: specifically, if the type of the left operand is BigInt, they return BigInt; otherwise, they return Number.

The shift operators are listed in the following table.

### Bitwise shift operators

Operator	Description	Example
<u>Left shift</u> <code>(&lt;&lt;)</code>	This operator shifts the first operand the specified number of bits to the left. Excess bits shifted off to the left are discarded. Zero bits are shifted in from the right.	9<<2 yields 36, because 1001 shifted left becomes 100100, which is 36.
<u>Sign-propagating right shift</u> ( <code>&gt;&gt;</code> )	This operator shifts the first operand the specified number of bits to the right. Excess bits shifted off to the right are discarded. Copies of the leftmost bit are shifted in from the	9>>2 yields 2, because 1001 shifted right becomes 10, which is 2. Likewise, -9>>2 yields -2, because -1001 shifted right becomes -10, which is -2. The sign is preserved.

## Bitwise shift operators

Operator	Description	Example
	left.	
<u>Zero-fill right shift (&gt;&gt;)</u>	This operator shifts the first operand the specified number of bits to the right. Excess bits shifted off to the right are discarded. Zero bits are shifted in from the left.	19>>>2 yields 4, because 10011 shifts becomes 100, which is 4. For non-negative right shift and sign-propagating right result.

## Logical operators

Logical operators are typically used with Boolean (logical) values; when they are, they return a Boolean value. However, the `&&` and `||` operators actually return the value of one of the specified operands, so if these operators are used with non-Boolean values, they may return a non-Boolean value. The logical operators are described in the following table.

## Logical operators

Operator	Usage	Description
<u>Logical AND (&amp;&amp;)</u>	<code>expr1 &amp;&amp; expr2</code>	Returns <code>expr1</code> if it can be converted to false; otherwise, returns <code>expr2</code> . Thus, when used with Boolean values, <code>&amp;&amp;</code> returns true if both operands are true; otherwise, returns false.
<u>Logical OR (  )</u>	<code>expr1    expr2</code>	Returns <code>expr1</code> if it can be converted to true; otherwise, returns <code>expr2</code> . Thus, when used with Boolean values, <code>  </code> returns true if either operand is true; otherwise, returns false.

## Logical operators

Operator	Usage	Description
	expr2	values,    returns true if either operand is true; if both are false, returns false.
<u>Logical NOT (!)</u>	!expr	Returns false if its single operand that can be converted to true; otherwise, returns true.

Examples of expressions that can be converted to false are those that evaluate to null, 0, NaN, the empty string (""), or undefined.

The following code shows examples of the && (logical AND) operator.

```
var a1 = true && true; // t && t returns true

var a2 = true && false; // t && f returns false

var a3 = false && true; // f && t returns false

var a4 = false && (3 == 4); // f && f returns false

var a5 = 'Cat' && 'Dog'; // t && t returns Dog

var a6 = false && 'Cat'; // f && t returns false

var a7 = 'Cat' && false; // t && f returns false
```

Copy to Clipboard

The following code shows examples of the || (logical OR) operator.

```
var o1 = true || true; // t || t returns true
```

```
var o2 = false || true; // f || t returns true
```

```
var o3 = true || false; // t || f returns true
```

```
var o4 = false || (3 == 4); // f || f returns false
```

```
var o5 = 'Cat' || 'Dog'; // t || t returns Cat
```

```
var o6 = false || 'Cat'; // f || t returns Cat
```

```
var o7 = 'Cat' || false; // t || f returns Cat
```

Copy to Clipboard

The following code shows examples of the ! (logical NOT) operator.

```
var n1 = !true; // !t returns false
```

```
var n2 = !false; // !f returns true
```

```
var n3 = !'Cat'; // !t returns false
```

Copy to Clipboard

Short-circuit evaluation

As logical expressions are evaluated left to right, they are tested for possible "short-circuit" evaluation using the following rules:

- false `&&` anything is short-circuit evaluated to false.
- true `||` anything is short-circuit evaluated to true.

The rules of logic guarantee that these evaluations are always correct. Note that the *anything* part of the above expressions is not evaluated, so any side effects of doing so do not take effect.

Note that for the second case, in modern code you can use the new Nullish coalescing operator (`??`) that works like `||`, but it only returns the second expression, when the first one is "nullish", i.e. `null` or `undefined`. It is thus the better alternative to provide defaults, when values like " or 0 are valid values for the first expression, too.

## String operators

In addition to the comparison operators, which can be used on string values, the concatenation operator (+) concatenates two string values together, returning another string that is the union of the two operand strings.

For example,

```
console.log('my ' + 'string'); // console logs the string "my string".
```

[Copy to Clipboard](#)

The shorthand assignment operator `+=` can also be used to concatenate strings.

For example,

```
var mystring = 'alpha';
```

~ ~

```
mystring += 'bet'; // evaluates to "alphabet" and assigns this value to mystring.
```

Copy to Clipboard

### Conditional (ternary) operator

The conditional operator is the only JavaScript operator that takes three operands. The operator can have one of two values based on a condition. The syntax is:

```
condition ? val1 : val2
```

Copy to Clipboard

If condition is true, the operator has the value of val1. Otherwise it has the value of val2. You can use the conditional operator anywhere you would use a standard operator.

For example,

```
var status = (age >= 18) ? 'adult' : 'minor';
```

Copy to Clipboard

This statement assigns the value "adult" to the variable status if age is eighteen or more. Otherwise, it assigns the value "minor" to status.

### Comma operator

The comma operator (,) evaluates both of its operands and returns the value of the last operand. This operator is primarily used inside a for loop, to allow multiple variables to be updated each time through the loop. It is regarded

bad style to use it elsewhere, when it is not necessary. Often two separate statements can and should be used instead.

For example, if a is a 2-dimensional array with 10 elements on a side, the following code uses the comma operator to update two variables at once. The code prints the values of the diagonal elements in the array:

```
var x = [0,1,2,3,4,5,6,7,8,9]
```

```
var a = [x, x, x, x, x];
```

```
for (var i = 0, j = 9; i <= j; i++, j--)
```

```
// ^
```

```
console.log('a[' + i + '][' + j + ']' + a[i][j]);
```

[Copy to Clipboard](#)

## Unary operators

A unary operation is an operation with only one operand.

`delete`

The delete operator deletes an object's property. The syntax is:

```
delete object.property;
```

```
delete object[propertyKey];
```

```
delete objectName[index];
```

Copy to Clipboard

where object is the name of an object, property is an existing property, and propertyKey is a string or symbol referring to an existing property.

If the delete operator succeeds, it removes the property from the object. Trying to access it afterwards will yield undefined. The delete operator returns true if the operation is possible; it returns false if the operation is not possible.

```
delete Math.PI; // returns false (cannot delete non-configurable properties)
```

```
const myObj = {h: 4};
```

```
delete myobj.h; // returns true (can delete user-defined properties)
```

Copy to Clipboard

Deleting array elements

Since arrays are just objects, it's technically possible to delete elements from them. This is however regarded as a bad practice, try to avoid it. When you delete an array property, the array length is not affected and other elements are not re-indexed. To achieve that behavior, it is much better to just overwrite the element with the value undefined. To actually manipulate the array, use the various array methods such as splice.

typeof



The typeof operator is used in either of the following ways:

typeof operand

typeof (operand)

Copy to Clipboard

The typeof operator returns a string indicating the type of the unevaluated operand. operand is the string, variable, keyword, or object for which the type is to be returned. The parentheses are optional.

Suppose you define the following variables:

```
var myFun = new Function('5 + 2');
```

```
var shape = 'round';
```

```
var size = 1;
```

```
var foo = ['Apple', 'Mango', 'Orange'];
```

```
var today = new Date();
```

Copy to Clipboard

The typeof operator returns the following results for these variables:

```
typeof myFun; // returns "function"
```

```
typeof shape; // returns "string"
```

```
typeof size; // returns "number"
```

```
typeof foo; // returns "object"
```

```
typeof today; // returns "object"
```

```
typeof doesntExist; // returns "undefined"
```

Copy to Clipboard

For the keywords true and null, the typeof operator returns the following results:

```
typeof true; // returns "boolean"
```

```
typeof null; // returns "object"
```

Copy to Clipboard

For a number or string, the typeof operator returns the following results:

```
typeof 62; // returns "number"
```

```
typeof 'Hello world'; // returns "string"
```

Copy to Clipboard

For property values, the typeof operator returns the type of value the property contains:

```
typeof document.lastModified; // returns "string"
```

```
typeof window.length; // returns "number"
```

~ ~

```
typeof Math.LN2; // returns "number"
```

Copy to Clipboard

For methods and functions, the typeof operator returns results as follows:

```
typeof blur; // returns "function"
```

```
typeof eval; // returns "function"
```

```
typeof parseInt; // returns "function"
```

```
typeof shape.split; // returns "function"
```

Copy to Clipboard

For predefined objects, the typeof operator returns results as follows:

```
typeof Date; // returns "function"
```

```
typeof Function; // returns "function"
```

```
typeof Math; // returns "object"
```

```
typeof Option; // returns "function"
```

```
typeof String; // returns "function"
```

Copy to Clipboard

void

~ ~

The void operator is used in either of the following ways:

void (expression)

void expression

Copy to Clipboard

The void operator specifies an expression to be evaluated without returning a value. expression is a JavaScript expression to evaluate. The parentheses surrounding the expression are optional, but it is good style to use them.

### Relational operators

A relational operator compares its operands and returns a Boolean value based on whether the comparison is true.

in

The in operator returns true if the specified property is in the specified object. The syntax is:

propNameOrNumber in objectName

Copy to Clipboard

where propNameOrNumber is a string, numeric, or symbol expression representing a property name or array index, and objectName is the name of an object.

The following examples show some uses of the in operator.

// Arrays

```
var trees = ['redwood', 'bay', 'cedar', 'oak', 'maple'];
```

```
0 in trees; // returns true

3 in trees; // returns true

6 in trees; // returns false

'bay' in trees; // returns false (you must specify the index number,
 // not the value at that index)

'length' in trees; // returns true (length is an Array property)

// built-in objects

'PI' in Math; // returns true

var myString = new String('coral');

'length' in myString; // returns true

// Custom objects

var mycar = { make: 'Honda', model: 'Accord', year: 1998 };

'make' in mycar; // returns true

'model' in mycar; // returns true
```

- - -

Copy to Clipboard

instanceof

The instanceof operator returns true if the specified object is of the specified object type. The syntax is:

objectName instanceof objectType

Copy to Clipboard

where objectName is the name of the object to compare to objectType, and objectType is an object type, such as Date or Array.

Use instanceof when you need to confirm the type of an object at runtime. For example, when catching exceptions, you can branch to different exception-handling code depending on the type of exception thrown.

For example, the following code uses instanceof to determine whether theDay is a Date object. Because theDay is a Date object, the statements in the if statement execute.

```
var theDay = new Date(1995, 12, 17);
```

```
if (theDay instanceof Date) {
```

```
 // statements to execute
```

```
}
```

Copy to Clipboard

Operator precedence



The *precedence* of operators determines the order they are applied when evaluating an expression. You can override operator precedence by using parentheses.

The following table describes the precedence of operators, from highest to lowest.

<b>Operator type</b>	<b>Individual operators</b>
member	. []
call / create instance	() new
negation/increment	! ~ - + ++ -- typeof void delete
multiply/divide	* / %
addition/subtraction	+ -
bitwise shift	<< >> >>>
relational	< <= > >= in instanceof
equality	== != === !==
bitwise-and	&
bitwise-xor	^
bitwise-or	

<b>Operator type</b>	<b>Individual operators</b>
logical-and	<code>&amp;&amp;</code>
logical-or	<code>  </code>
conditional	<code>?:</code>
assignment	<code>= += -= *= /= %= &lt;=&gt;= &gt;&gt;= &amp;= ^=  = &amp;&amp;=   = ??=</code>
comma	<code>,</code>

A more detailed version of this table, complete with links to additional details about each operator, may be found in [JavaScript Reference](#).

## **Expressions**

An *expression* is any valid unit of code that resolves to a value.

Every syntactically valid expression resolves to some value but conceptually, there are two types of expressions: with side effects (for example: those that assign value to a variable) and those that in some sense evaluate and therefore resolve to a value.

The expression `x = 7` is an example of the first type. This expression uses the `= operator` to assign the value seven to the variable `x`. The expression itself evaluates to seven.

The code `3 + 4` is an example of the second expression type. This expression uses the `+` operator to add three and four together without assigning the result, seven, to a variable.

JavaScript has the following expression categories:

- Arithmetic: evaluates to a number, for example 3.14159. (Generally uses arithmetic operators.)
- String: evaluates to a character string, for example, "Fred" or "234". (Generally uses string operators.)
- Logical: evaluates to true or false. (Often involves logical operators.)
- Primary expressions: Basic keywords and general expressions in JavaScript.
- Left-hand-side expressions: Left values are the destination of an assignment.

## Primary expressions

Basic keywords and general expressions in JavaScript.

this

Use the this keyword to refer to the current object. In general, this refers to the calling object in a method.

Use this either with the dot or the bracket notation:

this['propertyName']

this.propertyName

Copy to Clipboard

Suppose a function called validate validates an object's value property, given the object and the high and low values:

```
function validate(obj, lowval, hival) {
```

```
 if ((obj.value < lowval) || (obj.value > hival))
```

~ ~ ~

~

```
 console.log('Invalid Value!');
```

```
}
```

[Copy to Clipboard](#)

You could call validate in each form element's onChange event handler, using this to pass it to the form element, as in the following example:

```
<p>Enter a number between 18 and 99:</p>
```

```
<input type="text" name="age" size=3 onChange="validate(this, 18, 99);">
```

[Copy to Clipboard](#)

[Grouping operator](#)

The grouping operator ( ) controls the precedence of evaluation in expressions. For example, you can override multiplication and division first, then addition and subtraction to evaluate addition first.

```
var a = 1;
```

```
var b = 2;
```

```
var c = 3;
```

```
// default precedence
```

```
a + b * c // 7
```

// evaluated by default like this

a + (b \* c) // 7

// now overriding precedence

// addition before multiplication

(a + b) \* c // 9

// which is equivalent to

a \* c + b \* c // 9

Copy to Clipboard

### Left-hand-side expressions

Left values are the destination of an assignment.

new

You can use the new operator to create an instance of a user-defined object type or of one of the built-in object types. Use new as follows:

var objectName = new objectType([param1, param2, ..., paramN]);

[Copy to Clipboard](#)

super

The super keyword is used to call functions on an object's parent. It is useful with classes to call the parent constructor, for example.

super([arguments]); // calls the parent constructor.

super.functionOnParent([arguments]);

## Statements and declarations

JavaScript applications consist of statements with an appropriate syntax. A single statement may span multiple lines. Multiple statements may occur on a single line if each statement is separated by a semicolon. This isn't a keyword, but a group of keywords.

### Statements and declarations by category

For an alphabetical listing see the sidebar on the left.

#### Control flow

##### Block

A block statement is used to group zero or more statements. The block is delimited by a pair of curly brackets.

##### break

Terminates the current loop, switch, or label statement and transfers program control to the statement following the terminated statement.

##### continue

Terminates execution of the statements in the current iteration of the current or labeled loop, and continues execution of the loop with the next iteration.

##### Empty

An empty statement is used to provide no statement, although the JavaScript syntax would expect one.

##### if...else

Executes a statement if a specified condition is true. If the condition is false, another statement can be executed.

## **switch**

Evaluates an expression, matching the expression's value to a case clause, and executes statements associated with that case.

## **throw**

Throws a user-defined exception.

## **try...catch**

Marks a block of statements to try, and specifies a response, should an exception be thrown.

### *Declarations*

#### **var**

Declares a variable, optionally initializing it to a value.

#### **let**

Declares a block scope local variable, optionally initializing it to a value.

#### **const**

Declares a read-only named constant.

### *Functions and classes*

#### **function**

Declares a function with the specified parameters.

#### **function\***

Generator Functions enable writing iterators more easily.

#### **async function**

Declares an async function with the specified parameters.

#### **return**

Specifies the value to be returned by a function.

#### **class**

Declares a class.

### *Iterations*

#### **do...while**

Creates a loop that executes a specified statement until the test condition evaluates to false. The condition is evaluated after executing the statement, resulting in the specified statement executing at least once.

**for**

Creates a loop that consists of three optional expressions, enclosed in parentheses and separated by semicolons, followed by a statement executed in the loop.

**for...in**

Iterates over the enumerable properties of an object, in arbitrary order. For each distinct property, statements can be executed.

**for...of**

Iterates over iterable objects (including arrays, array-like objects, iterators and generators), invoking a custom iteration hook with statements to be executed for the value of each distinct property.

**for await...of**

Iterates over async iterable objects, array-like objects, iterators and generators, invoking a custom iteration hook with statements to be executed for the value of each distinct property.

**while**

Creates a loop that executes a specified statement as long as the test condition evaluates to true. The condition is evaluated before executing the statement.

**Window Object**

The window object represents an open window in a browser.

If a document contain frames (<iframe> tags), the browser creates one window object for the HTML document, and one additional window object for each frame.

**Note:** There is no public standard that applies to the Window object, but all major browsers support it.

**Window Object Properties**

Property	Description
<u>closed</u>	Returns a Boolean value indicating whether a window has been closed or not
<u>console</u>	Returns a reference to the Console object, which provides methods for logging information to the browser's console ( <u>See Console object</u> )

defaultStatus Sets or returns the default text in the statusbar of a window

document Returns the Document object for the window ([See Document object](#))

frameElement Returns the <iframe> element in which the current window is inserted

frames Returns all <iframe> elements in the current window

history Returns the History object for the window ([See History object](#))

innerHeight Returns the height of the window's content area (viewport) including scrollbars

innerWidth Returns the width of a window's content area (viewport) including scrollbars

length Returns the number of <iframe> elements in the current window

localStorage Allows to save key/value pairs in a web browser. Stores the data with no expiration date

location Returns the Location object for the window ([See Location object](#))

name Sets or returns the name of a window

<u>navigator</u>	Returns the Navigator object for the window ( <a href="#">See Navigator object</a> )
<u>opener</u>	Returns a reference to the window that created the window
<u>outerHeight</u>	Returns the height of the browser window, including toolbars/scrollbars
<u>outerWidth</u>	Returns the width of the browser window, including toolbars/scrollbars
<u>pageXOffset</u>	Returns the pixels the current document has been scrolled (horizontally) from the upper left corner of the window
<u>pageYOffset</u>	Returns the pixels the current document has been scrolled (vertically) from the upper left corner of the window
<u>parent</u>	Returns the parent window of the current window
<u>screen</u>	Returns the Screen object for the window ( <a href="#">See Screen object</a> )
<u>screenLeft</u>	Returns the horizontal coordinate of the window relative to the screen
<u>screenTop</u>	Returns the vertical coordinate of the window relative to the screen
<u>screenX</u>	Returns the horizontal coordinate of the window relative to the screen

<u>screenY</u>	Returns the vertical coordinate of the window relative to the screen
<u>sessionStorage</u>	Allows to save key/value pairs in a web browser. Stores the data for one session
<u>scrollX</u>	An alias of <u>pageXOffset</u>
<u>scrollY</u>	An alias of <u>pageYOffset</u>
<u>self</u>	Returns the current window
<u>status</u>	Sets or returns the text in the statusbar of a window
<u>top</u>	Returns the topmost browser window

### Window Object Methods

Method	Description
<u>alert()</u>	Displays an alert box with a message and an OK button
<u>atob()</u>	Decodes a base-64 encoded string

blur() Removes focus from the current window

btoa() Encodes a string in base-64

clearInterval() Clears a timer set with setInterval()

clearTimeout() Clears a timer set with setTimeout()

close() Closes the current window

confirm() Displays a dialog box with a message and an OK and a Cancel button

focus() Sets focus to the current window

getComputedStyle() Gets the current computed CSS styles applied to an element

getSelection() Returns a Selection object representing the range of text selected by the user

matchMedia() Returns a MediaQueryList object representing the specified CSS media query string

<u>moveBy()</u>	Moves a window relative to its current position
<u>moveTo()</u>	Moves a window to the specified position
<u>open()</u>	Opens a new browser window
<u>print()</u>	Prints the content of the current window
<u>prompt()</u>	Displays a dialog box that prompts the visitor for input
<u>requestAnimationFrame()</u>	Requests the browser to call a function to update an animation before the next repaint
<u>resizeBy()</u>	Resizes the window by the specified pixels
<u>resizeTo()</u>	Resizes the window to the specified width and height
<u>scroll()</u>	Deprecated. This method has been replaced by the <u>scrollTo()</u> method.

<u>scrollBy()</u>	Scrolls the document by the specified number of pixels
<u>scrollTo()</u>	Scrolls the document to the specified coordinates
<u>setInterval()</u>	Calls a function or evaluates an expression at specified intervals (in milliseconds)
<u>setTimeout()</u>	Calls a function or evaluates an expression after a specified number of milliseconds
<u>stop()</u>	Stops the window from loading

The Window object is used to open a window in a browser to display the Web page.

The following figure shows the Window object in the hierarchy of Browsers objects.

The window object has the following three features in JavaScript:

- collection
- properties
- methods

### **JavaScript Window Object Collection**

The window object collection is a set of all the window objects available in an HTML document.

### **JavaScript Window Object Properties**

Web Programming

Page

All data and information about any browser is attached to the window object as properties and the frames property in the window object returns all the frames in the current window. The table given below describes properties of the window object in JavaScript.

<b>Property</b>	<b>Description</b>
closed	returns a boolean value that specifies whether a window has been closed or not
defaultStatus	specifies the default message that has to be appeared in the statusbar of a window
document	specifies a document object in the window
frames	specifies an array of all the frames in the current window
history	specifies a history object for the window
innerHeight	specifies the inner height of a window's content area
innerWidth	specifies the inner width of a window's content area
length	specifies the number of frames contained in a window
location	specifies a location object for the window
name	specifies the name of a window
outerHeight	specifies the height of the outside boundary of a window in pixels
outerWidth	specifies the width of the outside boundary of a window in pixels
parent	returns the parent frame or window of the current window
screenLeft	specifies the x coordinate of the window relative to a user's monitor screen
screenTop	specifies the y coordinate of the window relative to a user's monitor screen
screenX	specifies the x coordinate of the window relative to a user's monitor screen
screenY	specifies the y coordinate of the window relative to a user's monitor screen
self	returns the reference of the current active frame or window
status	specifies the message that is displayed in the status bar of a window, when an activity is performed on the window
top	specifies the reference of the topmost browser window

## **JavaScript Window Object Properties Example**

Here is an example demonstrates window object properties in JavaScript:

```
<!DOCTYPE HTML>
<html>
<head>
 <title>JavaScript Window Object Properties</title>
</head>
<body>

<h3>JavaScript Window Object Properties Example</h3>
<iframe src="http://codescracker.com/java"></iframe>
<iframe src="http://codescracker.com/c"></iframe>
<iframe src="http://codescracker.com/cpp"></iframe>
<script type="text/javascript">
 for(var i=0; i<frames.length; i++)
 {
 frames[i].location = http://codescracker.com"
 }
</script>

</body>
</html>
```

[Here is the sample output produced by the above JavaScript Window Object Properties example code:](#)

## **JavaScript Window Object Methods**

[The methods of the window object enable you to perform various tasks such as open a url in a new window or to close a window. The following table describes the methods of the Window object in JavaScript.](#)

Method	Description
alert()	displays an alert box with a message and an OK button
blur()	removes the focus from the current window
clearInterval()	clears the timer, which is set by using the setInterval() method
clearTimeout()	clears the timer, which is set by using the setTimeout() method

close()	closes the current window
confirm()	displays a dialog box with a message and two buttons, OK and Cancel
createPopup()	creates a pop-up window
focus()	sets focus on the current window
moveBy()	moves a window relative to its current position
moveTo()	moves a window to an specified position
open()	opens a new browser window
print()	sends a print command to print the content of the current window
prompt()	prompts for input
resizeBy()	resizes a window with the specified pixels
resizeTo()	resizes a window with the specified width and height
scrollBy()	scrolls the content of a window by the specified number of pixels
scrollTo()	scrolls the content of a window up to the specified coordinates
setInterval()	evaluates an expression at specified time intervals in milliseconds
setTimeout()	evaluates an expression after a specified number of milliseconds

### **JavaScript Window Object Methods Example**

Here is an example uses some window object methods in JavaScript:

```
<!DOCTYPE HTML>
<html>
<head>
<title>JavaScript Window Object Methods</title>
<script type="text/javascript">
var mywin;
function openMidWin(url)
{
 var wid = 500;
 var hei = 200;
 var winFeat = "width = " + wid + ", height = " + hei + ", status, resizable";
 myWin = window.open(url, "subWind", winFeat);
}
function disp_alert()
{
 alert("Hello, I am a subwindow!");
}
```

```

 {
 alert("Hi, This is an alert box.");
 }
 function resize_win()
 {
 window.resizeBy(-100, -100)
 }
 function close_win()
 {
 if(window.confirm("Do you really want to close the browser ?"))
 window.close();
 }
</script>
</head>
<body>
```

```

<h3>JavaScript Window Object Methods Example</h3>
<input type="button" value="Open New Window" onclick="openMidWin('WindowObjectMethod.htm')"/>
<input type="button" value="Alert" onclick="disp_alert()"/>
<input type="button" value="Resize Window" onclick="resize_win()"/>
<input type="button" value="Close Window" onclick="close_win()"/>

</body>
</html>
```

Here are some sample output produced by the above Window Object Methods in JavaScript example code. This is initial output:

This is the output, after clicking on Open New Window button:

This is the output, after clicking on Alert button:

This is the output produced after clicking on Resize Window button:

This is the output produced after clicking on **Close Window** button:

**UNIT – II****TOPICS:****Introduction to XML**

- Basic XMLdocument
- PresentingXML
- Document TypeDefinition(DTD)
- XMLSchemas
- Document ObjectModel(DOM)
- Introduction to XHTML
- Using XML Processors: DOM andSAX

**Introduction to PHP**

- DeclaringVariables
- DataTypes
- Operators
- ControlStructures
- Functions
- Reading data from WEB form controls like text boxes, radio buttons, listsetc..
- Handling FileUploads
- Handling Sessions andCookies

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

**The Difference between XML and HTML**

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

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

- XML documents must have a rootelement
- XML elements must have a closing tag(start tag must have matching endtag).
- XML tags are casesensitive
- XML elements must be properly nestedEx:<one><two>Hello</two></one>
- XML attribute values must be quoted

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

## What is Markup?

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

### Example for XML Document

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

- Xml document begins with XML declaration statement: <? xml version="1.0" encoding="ISO-8859-1"?>.
- The next line describes the **root element** of the document:<b><note></note></b>.
- This element is "the parent" of all other elements.
- The next 4 lines describe **4child elements** of the root: to, from, heading, and body. And finally the last line defines the end of the root element :<b></note></b>.
- The XML declaration has no closing tag i.e.<?xml>
- The **default standalone value** is set to **no**. Setting it to **yes** tells the processor there are no external declarations (DTD) required for parsing the document. The file name extension used for xml program is.xml.

### Valid XML document

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

### XML DTD

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

### Why use a DTD?

XML provides an application independent way of sharing data. With a DTD, independent groups of people can agree to use a common DTD for interchanging data. Your application can use a standard DTD to verify that data that you receive from the outside world is valid. You can also use a DTD to verify your own data.

### DTD - XML building blocks

Various building blocks of XML are-

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

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

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

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

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

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

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

The keyword ANY declares an element with any content.

**Example:**

```
<!ELEMENT note (#PCDATA)>
```

### Elements with children (sequences)

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

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

```
<!ELEMENT element-name(child-element-name,child-element-name,.)>
```

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

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

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

## 2. Tags

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

**Examples:**

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

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

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

## 4. Entities

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

**The following entities are predefined in XML:**

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

**5. CDATA:** It stands for character data. CDATA is text that will NOT be parsed by a parser. Tags inside the text will NOT be treated as markup and entities will not be expanded.

**6. PCDATA:** It stands for Parsed Character Data(i.e., text). Any parsed character data should not contain the markup characters. The markup characters are < or > or &. If we want to use these characters then make use of &lt;, &gt; or &amp;. Think of character data as the text found between the start tag and the end tag of an XML element. PCDATA is text that will be parsed by a parser. Tags inside the text will be treated as markup and entities will be expanded.

```
<!DOCTYPE note
```

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

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

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

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

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

### **Types of DTD:**

1. InternalDTD
2. ExternalDTD

### **1. Internal DTD**

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

### **Syntax:**

The syntax of internal DTD is as shown:

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

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

### **Example:**

Following is a simple example of internal DTD:

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

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

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

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

```
<!DOCTYPE address [
```

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

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

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

Several elements are declared here that make up the vocabulary of the <name> document.

<!ELEMENT name (#PCDATA)> defines the element name to be of type "#PCDATA". Here #PCDATA means parse-able text data. End Declaration - Finally, the declaration section of the DTD is closed using a closing bracket and a closing angle bracket (]>). This effectively ends the definition, and thereafter, the XML document followsimmediately.

### Rules

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

### External DTD

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

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

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

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

**Example** The following example shows external DTDusage:

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

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

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

### Types

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

## **SYSTEM IDENTIFIERS**

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

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

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

## **PUBLIC IDENTIFIERS**

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

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

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

## **XML Schemas**

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

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

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

### **Example : contact.xsd**

The following example shows how to use schema:

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

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

### **XML Document: myschema.xml**

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

```
<contact xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="contact.xsd">
<name>KALPANA</name>
<company>04024056789</company>
<phone>9876543210</phone>
</contact>
```

### **Limitations of DTD:**

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

### **Strengths of Schema:**

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

## **SCHEMA STRUCTURE**

### **The Schema Element**

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

### **Element definitions**

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

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

### **Data types:**

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

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

### **EX: Sample.xsd**

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

```
</xs:schema>
```

### **Sample.xml:**

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

### **Definition Types**

You can define XML schema elements in following ways:

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

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

### **Default and Fixed Values for Simple Elements**

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

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

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

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

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

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

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

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

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

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

```

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

```

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

### **Attributes**

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

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

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

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

### **Default and Fixed Values for Attributes**

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

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

### **Optional and Required Attributes**

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

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

### **Restrictions on Content**

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

### **Restrictions on Values:**

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

```

<xs:element name="age">
 <xs:simpleType>
 <xs:restriction base="xs:integer">
 <xs:minInclusive value="0"/>
 <xs:maxInclusive value="120"/>
 </xs:restriction>
 </xs:simpleType></xs:element>

```

## Restrictions on a Set of Values

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

```
<xs:element name="car">
 <xs:simpleType>
 <xs:restriction base="xs:string">
 <xs:enumeration value="Audi"/>
 <xs:enumeration value="Golf"/>
 <xs:enumeration value="BMW"/>
 </xs:restriction>
 </xs:simpleType>
</xs:element>
```

## Restrictions on Length

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

```
<xs:element name="password">
 <xs:simpleType>
 <xs:restriction base="xs:string">
 <xs:lengthvalue="8"/> [<xs:minLengthvalue="5"/> <xs:maxLengthvalue="8"/>]
 </xs:restriction></xs:simpleType></xs:element>
```

## XSD Indicators

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

**Indicators:** There are seven indicators

**Order indicators:**

- All
- Choice
- Sequence

**Occurrence indicators:**

- maxOccurs
- minOccurs

**Group indicators:**

- Groupname
- attributeGroupname

## →Order Indicators

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

### All Indicator

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

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

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

### Choice Indicator

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

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

### Sequence Indicator

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

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

### →Occurrence Indicators

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

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

### maxOccurs Indicator

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

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

### minOccurs Indicator

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

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

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

**maxOccurs="unbounded"** statement:

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

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

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

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

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

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

### **Element Groups**

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

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



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

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

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

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

### **Attribute Groups**

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

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

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

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

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

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

### **Example Program: "shiporder.xml"**

```
<?xml version="1.0" encoding="UTF-8"?>
<shiporder orderid="889923"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-
 instance"
 xsi:noNamespaceSchemaLocation="shiporder.xs
 d">
 <orderperson>John Smith</orderperson>
 <shipto>
 <name>Ola Nordmann</name>
 <address>Langgt 23</address>
```

<city>4000 Stavanger</city>

---

---

```

<country>Norway</country>
</shipto>
<item>
 <title>Empire Burlesque</title>
 <note>Special Edition</note>
 <quantity>1</quantity>
 <price>10.90</price>
</item>
<item>
 <title>Hide your heart</title> <quantity>1</quantity>
 <price>9.90</price></item>
</shiporder>
```

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

```

<?xml version="1.0" encoding="UTF-8" ?>
<xsschema xmlns:xss="http://www.w3.org/2001/XMLSchema">
<xselement name="shiporder">
<xsccomplexType>
<xsssequence>
<xselement name="orderperson" type="xs:string"/>
<xselement name="shipto">
<xsccomplexType>
<xsssequence>
<xselement name="name" type="xs:string"/>
<xselement name="address" type="xs:string"/>
<xselement name="city" type="xs:string"/>
<xselement name="country" type="xs:string"/>
</xsssequence>
</xsccomplexType>
</xselement>
<xselement name="item" maxOccurs="unbounded">
<xsccomplexType>
<xsssequence>
<xselement name="title" type="xs:string"/>
<xselement name="note" type="xs:string" minOccurs="0"/>
<xselement name="quantity" type="xs:positiveInteger"/>
<xselement name="price" type="xs:decimal"/>
</xsssequence>
</xsccomplexType>
</xselement>
</xsssequence>
<xseattribute name="orderid" type="xs:string" use="required"/>
</xsccomplexType>
</xselement>
</xsschema>
```

#### **XML DTD vs XML Schema**

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

## Schema vs. DTD

- XML Schemas are extensible to future additions
- XML Schemas are richer and more powerful than DTDs
- XML Schemas are written in XML
- XML Schemas support datatypes
- XML Schemas support namespaces

## XML Parsers

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

### Two types of XML parsers:

#### ➤ ValidatingParser

- It requires document type declaration
- It generates error if document doesnot
  - Conform with DTDand
  - Meet XML validityconstraints

#### ➤ Non-validating Parser

- It checks well-formedness for xmldocument
- It can ignore externalDTD

## What is XML Parser?

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

### Types of parsers:

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

## DOM-Document Object Model

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

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

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

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

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

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

- The entire document is a documentnode
- Every XML element is an elementnode
- The text in the XML elements are textnodes
- Every attribute is an attributenode
- Comments are comment nodes

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

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

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

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

### **DOM Levels**

- Level 1 Core: W3C Recommendation, October1998
- ✓ It has feature for primitive navigation and manipulation of XMLtrees
- ✓ other Level 1 features are: All HTMLfeatures
- Level 2 Core: W3C Recommendation, November2000
- ✓ It adds Namespace support and minor newfeatures
- ✓ other Level 2 features are: Events, Views, Style, Traversal andRange
- Level 3 Core: W3C Working Draft, April2002
- ✓ It supports: Schemas, XPath, XSL, XSLT

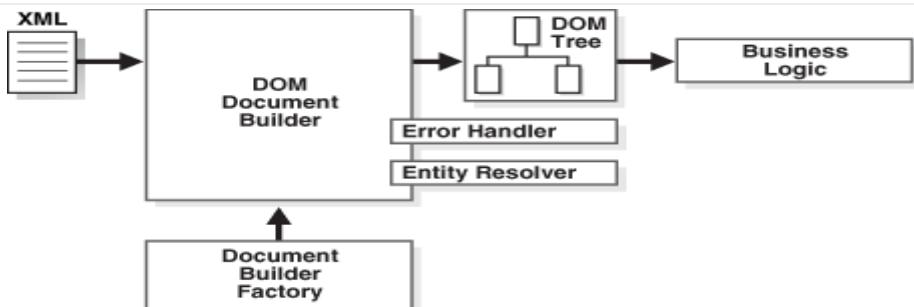
We can access and parse the XML document in two ways:

- **Parsing using DOM (treebased)**
- **Parsing using SAX (Eventbased)**

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

## Steps to Using DOM Parser

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



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

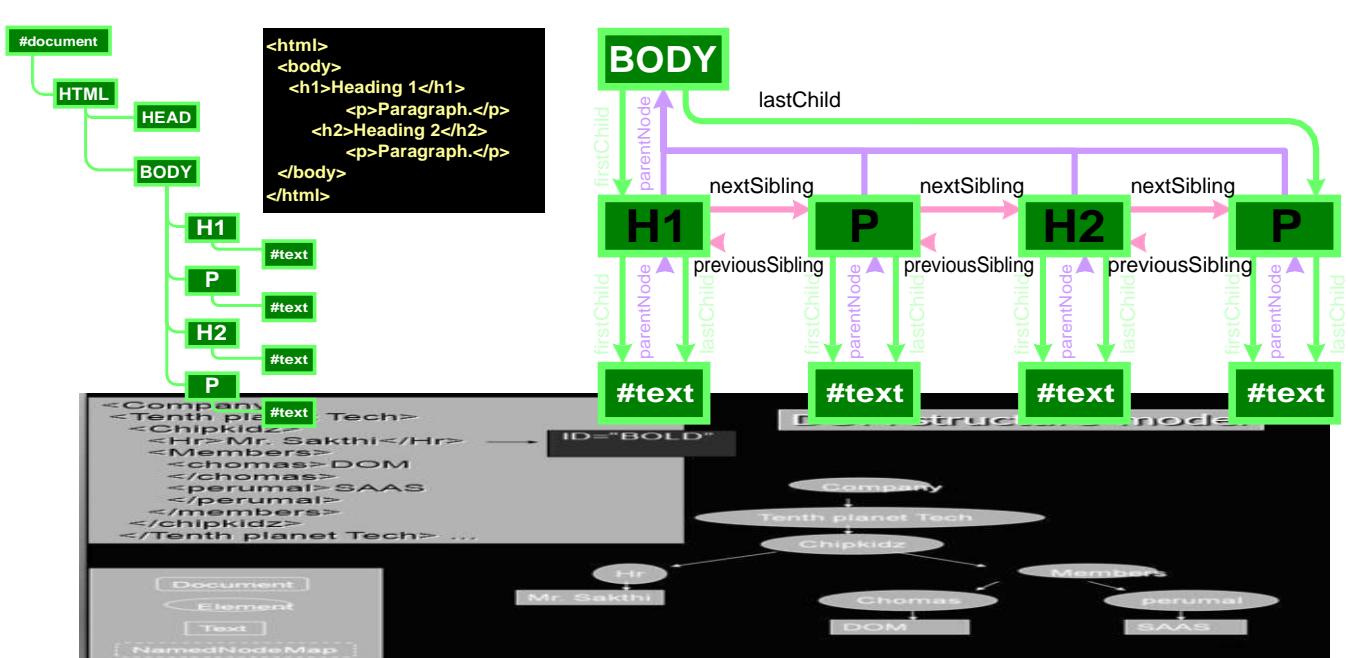
JAXP is a tool, stands for Java Api for Xml Processing, used for accessing and manipulating xml document in a tree based manner.

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

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

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

Property	Meaning
nodeName	Finding the name of the node
nodeValue	Obtaining value of the node
parentNode	To get parent node
childNodes	Obtain child nodes
Attributes	For getting the attributes values
Method	Meaning
getElementsByName(name)	To access the element by specifying its name
appendChild(node)	To insert a child node
removeChild(node)	To remove existing child node



***DOM Document Object***

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

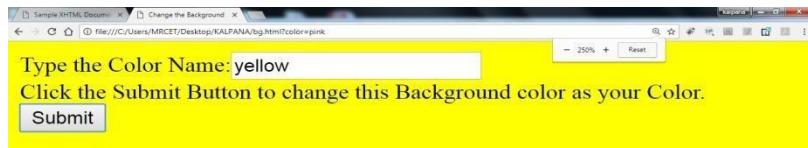
1. Document node	7. EntityReferencenode
2. Elementnode	8. Entitynode
3. Textnode	9. Commentnode
4. Attributemode	10. DocumentTypenode
5. Processing instructionnode	11. DocumentFragmentnode
6. CDATA Sectionnode	12. Notationnode

## Examples for Document method

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

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

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



## DOM NODE Methods

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

## DOM Advantages & Disadvantages

### ADVANTAGES

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

### Disadvantages

- Stores the entire document in memory, It is memoryinefficient
- AsDomwaswrittenforanylanguage,methodnamingconventionsdon'tfollowstandard java programmingconventions

## **DOM or SAX**

### **DOM**

- Suitable for smalldocuments
- Easily modifydocument
- Memory intensive;Load the complete XMLdocument

### **SAX**

- Suitable for large documents; saves significant amounts ofmemory
- Only traverse document once, start toend
- Eventdriven
- Limited standardfunctions.
- 

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

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

### **faculty.xml:**

```
<?xml version="1.0"?>
< faculty >
 <eno>30</eno>
<personal_inf>
 <name>Kalpana</name>
 <address>Hyd</address>
```

```

<phone>9959967192</phone>
</personal_inf>
<dept>CSE</dept>
<col>MRCE</col>
<group>MRGI</group>
</faculty>
```

**OUTPUT:** XML document student is loaded

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

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

```

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

**OUTPUT:** XML document student is loaded

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

```
<html><head>
```

```

<script type="text/javascript" src="my_function_file.js"></script>
</head><body>
<script type="text/javascript">
xmlDocument=My_function("faculty.xml");
document.write(-XMLdocumentfacultyisloadedandcontentofthisfileis:);
document.write(-
);
document.write(-ENO:+
xmlDocument.getElementsByTagName(-eno)[0].childNodes[0].nodeValue);
document.write(-
);
document.write(-Name:+
xmlDocument.getElementsByTagName(-name)[0].childNodes[0].nodeValue);
document.write(-
);
document.write(-ADDRESS:+
xmlDocument.getElementsByTagName(-address)[0].childNodes[0].nodeValue);
document.write(-
);
document.write(-PHONE:+
xmlDocument.getElementsByTagName(-phone)[0].childNodes[0].nodeValue);
document.write(-
);
document.write(-DEPARTMENT:+
xmlDocument.getElementsByTagName(-dept)[0].childNodes[0].nodeValue);
document.write(-
);
document.write(-COLLEGE:+
xmlDocument.getElementsByTagName(-coll)[0].childNodes[0].nodeValue);
document.write(-
);
document.write(-GROUP:+
xmlDocument.getElementsByTagName(-group)[0].childNodes[0].nodeValue);
</script>
</body>
</html>

```

**OUTPUT:**

XML document faculty is loaded and content of this file is

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

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

<html><head>

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

**OUTPUT:** Kalpana

### **XHTML: eXtensible Hypertext Markup Language**

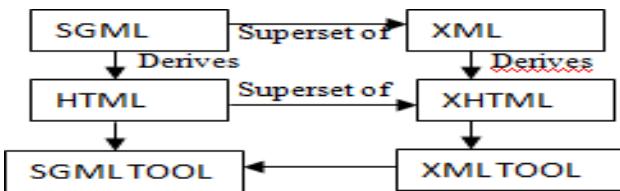
**Hypertext** is simply a piece of text that works as a link. **Markup language** is a language of writing layout information within documents. The XHTML recommended by **W3C**. Basically an XHTML document is a plain text file and it is very much similar to HTML. It contains rich text, means text with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

#### **HTML Vs XHTML**

1. The HTML tags are case insensitive. EX:<BoDy> ----- </body>	1. The XHTML tags are case sensitive. EX:<body>----- </body>
2. We can omit the closing tags sometimes.	2. For every tag there must be a closing tag. EX: <h1>-----</h1>or<h1 -----/>
3. The attribute values not always necessary to quote.	3. The attribute values are must be quoted.
4. In HTML there are some implicit attribute values.	4. In XHTML the attribute values must be specified explicitly.
5. In HTML even if we do not follow the nesting rules strictly it does not cause much difference.	5. In XHTML the nesting rules must be strictly followed. These nesting rules are- <ul style="list-style-type: none"> <li>- A form element cannot contain another form element.</li> <li>-an anchor element does not contain another form element</li> <li>-List element cannot be nested in the list element</li> <li>-If there are two nested elements then the inner element must be enclosed first before closing the outer element</li> <li>-Text element cannot be directly nested in form element</li> </ul>

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





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

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

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

**EX:**

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

<html xmlns="http://www.w3c.org/1999/xhtml">
<head>
<title>Sample XHTML Document</title>
</head>
<body bgcolor="#FF0000">
<basefont face="arial" size="1">
<h1>MR CET</h1>
<h2>MR CET</h2>
<h3>MR CET</h3>
<h4> KALPANA</h4>
<h5> KALPANA</h5>
<h6>KALPANA</h6>
<p><center> XHTML syntax rules are specified by the file xhtml11.dtd file. </center></p>
<div align="right">XHTML standards for eXtensible Hypertext Markup Language

XHTML syntax rules are specified by the file xhtml11.dtd file.</div>
<pre>XHTML standards for <i>eXtensible Hypertext Markup Language</i>

XHTML syntax rules are specified by the file xhtml11.dtd file.</pre>
</basefont>
</body>
</html>
```

## DOM in JAVA

### DOM interfaces

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

- **Node** - The base datatype of the DOM.
- **Element** - The vast majority of the objects you'll deal with are Elements.

- **Attr** Represents an attribute of an element.
-

- **Text** The actual content of an Element or Attr.
- **Document** Represents the entire XML document. A Document object is often referred to as a DOMtree.

### Common DOM methods

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

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

### Steps to Using DOM

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

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

### DOM

```
import java.io.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.*;
public class parsing_DOMDemo
{
 public static void main(String[] args)
 {
 try
 {
 System.out.println("Enter the name of XML document");
 BufferedReader input=new BufferedReader(new InputStreamReader(System.in));
 String file_name=input.readLine();
 File fp=new File(file_name);
 if(fp.exists())
 {
 try
 {
 DocumentBuilderFactory factory_obj= DocumentBuilderFactory.newInstance();
```

```

DocumentBuilder builder=Factory_obj.newDocumentBuilder();
InputSource ip_src=new InputSource(file_name);
Document doc=builder.parse(ip_src);
System.out.println(~file_name+is well-formed.);
}
catch (Exception e)
{
System.out.println(file_name+is not well-formed.);
System.exit(1);
} }
else
{
System.out.println(~file not found:|+file_name);
} }
catch(IOException ex)
{
ex.printStackTrace();
}
} }

```

### **SAX:**

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

- Reads an XML document from top to bottom, recognizing the tokens that make up a well-formed XMLdocument
- Tokens are processed in the same order that they appear in thedocument
- Reports the application program the nature of tokens that the parser has encountered as theyoccur
- The application program provides an "event" handler that must be registered with the parser
- As the tokens are identified, callback methods in the handler are invoked with the relevantinformation

### **When to use?**

You should use a SAX parser when:

- You can process the XML document in a linear fashion from the topdown
- The document is not deeplynested
- You are processing a very large XML document whose DOM tree would consume too much memory.Typical DOM implementations use ten bytes of memory to represent one byte ofXML
- The problem to be solved involves only part of the XMLdocument

- Data is available as soon as it is seen by the parser, so SAX works well for an XML document that arrives over a stream

## Disadvantages of SAX

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

## The kinds of events are:

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

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

## SAX packages

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

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

## SAX classes

- **SAXParser** Defines the API that wraps an XMLReader implementation class
- **SAXParserFactory** Defines a factory API that enables applications to configure and obtain a SAX based parser to parse XML documents
- **ContentHandler** Receive notification of the logical content of a document.
- **DTDHandler** Receive notification of basic DTD-related events.
- **EntityResolver** Basic interface for resolving entities.
- **ErrorHandler** Basic interface for SAX error handlers.
- **DefaultHandler** Default base class for SAX event handlers.

## SAX parser methods

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

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

**Characters()** – method called with the text contents in between the start and end tags of

an XML document element.

### **ContentHandler Interface**

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

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

### **Attributes Interface**

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

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

### **SAX simple API for XML**

```
import java.io.*;
import org.xml.sax;
import org.xml.sax.helpers;
public class parsing_SAXDemo
{
 public static void main(String[] args) throws IOException
 {
 try{
 System.out.println("Enter the name of XML document");
 BufferedReader input=new BufferedReader(new InputStreamReader(System.in));
 String file_name=input.readLine();
 File fp=new File(file_name);
 if(fp.exists())
 {
 try
 {
 XMLReader reader=XMLReaderFactory.createXMLReader();
 }
 }
 }
 }
}
```

```

reader.parse(file_name);
System.out.println(-file_name+is well-formed.());
}
catch (Exception e)
{
System.out.println(file_name+is not well-formed.());
System.exit(1);
}
}
else
{
System.out.println(-file not found:+file_name);
}
}
catch(IOException ex){ex.printStackTrace();}

```

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

} }

### Differences between DOM and SAX

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

## UNIT III

### Web Servers and Servlets

#### JDBC

JDBC Overview – JDBC implementation – Connection class – Statements - Caching Database Results, handling database Queries. Networking– InetAddress class – URL class- TCP sockets UDP sockets, Java Beans –RMI.

#### What is JDBC Driver?

JDBC drivers implement the defined interfaces in the JDBC API, for interacting with your database server.

For example, using JDBC drivers enable you to open database connections and to interact with it by sending SQL or database commands then receiving results with Java.

The `java.sql` package that ships with JDK, contains various classes with their behaviours defined and their actual implementations are done in third-party drivers. Third party vendors implements the `java.sql.Driver` interface in their database driver.

#### JDBC Drivers Types

JDBC driver implementations vary because of the wide variety of operating systems and hardware platforms in which Java operates. Sun has divided the implementation types into four categories, Types 1, 2, 3, and 4, which is explained below –

#### Type 1: JDBC-ODBC Bridge Driver

In a Type 1 driver, a JDBC bridge is used to access ODBC drivers installed on each client machine. Using ODBC, requires configuring on your system a Data Source Name (DSN) that represents the target database.

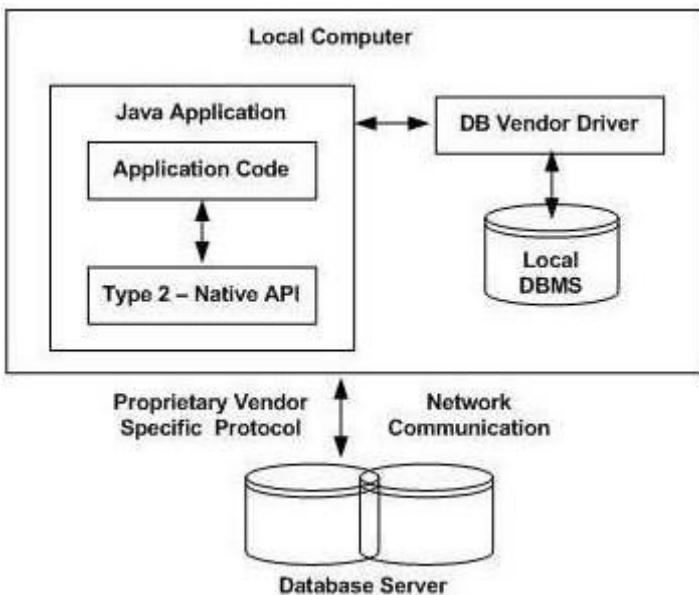
When Java first came out, this was a useful driver because most databases only supported ODBC access but now this type of driver is recommended only for experimental use or when no other alternative is available.

The JDBC-ODBC Bridge that comes with JDK 1.2 is a good example of this kind of driver.

### Type 2: JDBC-Native API

In a Type 2 driver, JDBC API calls are converted into native C/C++ API calls, which are unique to the database. These drivers are typically provided by the database vendors and used in the same manner as the JDBC-ODBC Bridge. The vendor-specific driver must be installed on each client machine.

If we change the Database, we have to change the native API, as it is specific to a database and they are mostly obsolete now, but you may realize some speed increase with a Type 2 driver, because it eliminates ODBC's overhead.

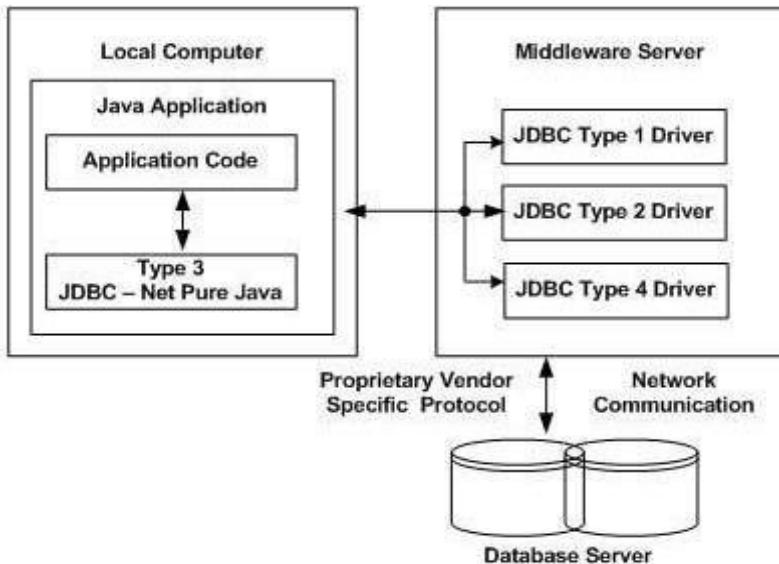


The Oracle Call Interface (OCI) driver is an example of a Type 2 driver.

### Type 3: JDBC-Net pure Java

In a Type 3 driver, a three-tier approach is used to access databases. The JDBC clients use standard network sockets to communicate with a middleware application server. The socket information is then translated by the middleware application server into the call format required by the DBMS, and forwarded to the database server.

This kind of driver is extremely flexible, since it requires no code installed on the client and a single driver can actually provide access to multiple databases.



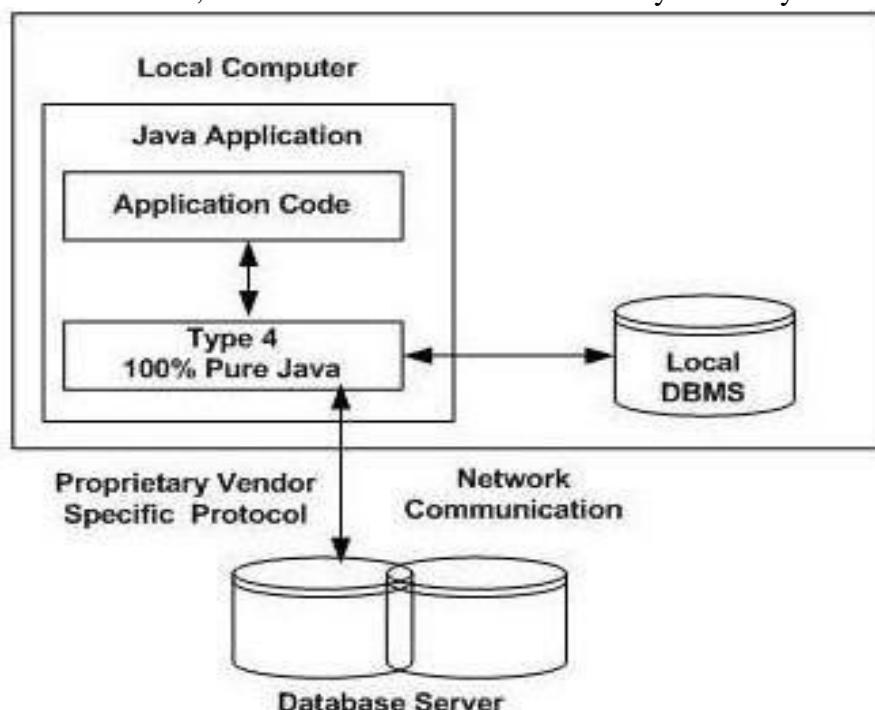
You can think of the application server as a JDBC "proxy," meaning that it makes calls for the client application. As a result, you need some knowledge of the application server's configuration in order to effectively use this driver type.

Your application server might use a Type 1, 2, or 4 driver to communicate with the database, understanding the nuances will prove helpful.

#### Type 4: 100% Pure Java

In a Type 4 driver, a pure Java-based driver communicates directly with the vendor's database through socket connection. This is the highest performance driver available for the database and is usually provided by the vendor itself.

This kind of driver is extremely flexible, you don't need to install special software on the client or server. Further, these drivers can be downloaded dynamically.



MySQL's Connector/J driver is a Type 4 driver. Because of the proprietary nature of their network protocols, database vendors usually supply type 4 drivers.

### Which Driver should be Used?

If you are accessing one type of database, such as Oracle, Sybase, or IBM, the preferred driver type is 4.

If your Java application is accessing multiple types of databases at the same time, type 3 is the preferred driver.

Type 2 drivers are useful in situations, where a type 3 or type 4 driver is not available yet for your database.

The type 1 driver is not considered a deployment-level driver, and is typically used for development and testing purposes only.

### JDBC( Java Database Connectivity):

The first thing you need to do is check that you are set up properly. This involves the following steps:

#### 1. Install Java and JDBC on your machine.

To install both the Java tm platform and the JDBC API, simply follow the instructions for downloading the latest release of the JDK tm (Java Development Kit tm ). When you download the JDK, you will get JDBC as well.

#### 2. Install a driver on your machine.

Your driver should include instructions for installing it. For JDBC drivers written for specific DBMSs, installation consists of just copying the driver onto your machine; there is no special configuration needed.

The JDBC-ODBC Bridge driver is not quite as easy to set up. If you download JDK, you will automatically get the JDBC-ODBC Bridge driver, which does not itself require any special configuration. ODBC, however, does. If you do not already have ODBC on your machine, you will need to see your ODBC driver vendor for information on installation and configuration.

#### 3. Install your DBMS if needed.

If you do not already have a DBMS installed, you will need to follow the vendor's instructions for installation. Most users will have a DBMS installed and will be working with an established database.

### Configuring Database:

Configuring a database is not at all difficult, but it requires special permissions and is normally done by a database administrator.

First, open the control panel. You might find "Administrative tools" select it, again you may find shortcut for "Data Sources (ODBC)". When you open the -Data Source (ODBC) 32bit ODBC icon, you'll see a "ODBC Data Source Administrator" dialog window with a number of tabs, including -User DSN, -System DSN, -File DSN, etc., in which -DSN means -Data Source Name. Select -System DSN, and add a new entry there, Select appropriate driver for the data source or directory where database lives. You can name the entry anything you want, assume here we are giving our data source name as "MySource".

### JDBC Database Access

JDBC was designed to keep simple things simple. This means that the JDBC API makes everyday database tasks, like simple SELECT statements, very easy.

**Import a package java.sql.\* :** This package provides you set of all classes that enables a network interface between the front end and back end database.

- DriverManager will create a Connection object.

- java.sql.Connection interface represents a connection with a specific database. Methods of connection is close(), createStatement(), prepareStatement(), commit(), close() and prepareCall()

- Statement interface used to interact with database via the execution of SQL statements. Methods of this interface are executeQuery(), executeUpdate(), execute() and getResultSet().

- A ResultSet is returned when you execute an SQL statement. It maintains a pointer to a row within the tabular results. Methods of this interface are next(), getBoolean(), getByte(), getDouble(), getString() close() and getInt().

### **Establishing a Connection**

The first thing you need to do is establish a connection with the DBMS you want to use. This involves two

steps: (1) loading the driver and (2) making the connection.

**Loading Drivers:** Loading the driver or drivers you want to use is very simple and involves just one line of code. If, for example, you want to use the JDBC-ODBC Bridge driver, the following code will load it

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

Your driver documentation will give you the class name to use. For instance, if the class name is jdbc.DriverXYZ , you would load the driver with the following line of code:

```
Class.forName("jdbc.DriverXYZ");
```

**Making the Connection:** The second step in establishing a connection is to have the appropriate driver connect to the DBMS. The following line of code illustrates the general idea:

```
Connection con = DriverManager.getConnection(url,"myLogin", "myPassword");
```

If you are using the JDBC-ODBC Bridge driver, the JDBC URL will start with jdbc:odbc: . The rest of the URL is generally your data source name or database system. So, if you are using ODBC to access an ODBC data source called "MySource, " for example, your JDBC URL could be jdbc:odbc:MySource . In place of " myLogin " you put the name you use to log in to the DBMS; in place of " myPassword " you put your password for the DBMS. So if you log in to your DBMS with a login name of " scott " and a password of "tiger" just these two lines of code will establish a connection:

```
String url = "jdbc:odbc:MySource";
Connection con = DriverManager.getConnection(url, "scott", "tiger");
```

The connection returned by the method DriverManager.getConnection is an open connection you can use to create JDBC statements that pass your SQL statements to the DBMS. In the

previous example, con is an open connection, and we will use it in the dorth coming examples.

### **Creating JDBC Statements**

A Statement object is what sends your SQL statement to the DBMS. You simply create a Statement object and then execute it, supplying the appropriate execute method with the SQL statement you want to send. For a SELECT statement, the method to use is executeQuery . For statements that create or modify tables, the method to use is executeUpdate .

It takes an instance of an active connection to create a Statement object. In the following example, we use our Connection object con to create the Statement object stmt :

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

At this point stmt exists, but it does not have an SQL statement to pass on to the DBMS. We need to supply that to the method we use to execute stmt.

For example, in the following code fragment, we supply executeUpdate with the SQL statement from the example above:

```
stmt.executeUpdate("CREATE TABLE STUDENT " +
"(S_NAME VARCHAR(32), S_ID INTEGER, COURSE VARCHAR2(10), YEAR
VARCHAR2(3));");
```

Since the SQL statement will not quite fit on one line on the page, we have split it into two strings concatenated by a plus sign (+) so that it will compile. ExecutingStatements.

Statements that create a table, alter a table, or drop a table are all examples of DDL statements and are executed with the method executeUpdate. The method executeUpdate is also used to execute SQL statements that update a table. In practice, executeUpdate is used far more often to update tables than it is to create them because a table is created once but may be updated manytimes.

The method used most often for executing SQL statements is executeQuery . This method is used to execute SELECT statements, which comprise the vast majority of SQL statements.

### **Entering Data into a Table**

We have shown how to create the table STUDENT by specifying the names of the columns and the data types to be stored in those columns, but this only sets up the structure of the table. The table does not yet contain any data. We will enter our data into the table one row at a time, supplying the information to be stored in each column of that row. Note that the values to be inserted into the columns are listed in the same order that the columns were declared when the table was created, which is the default order.

The following code inserts one row of data,

```
Statement stmt = con.createStatement();
stmt.executeUpdate("INSERT INTO STUDENT VALUES ('xStudent', 501,
B.Tech,'IV')");
```

Note that we use single quotation marks around the student name because it is nested within double quotation marks. For most DBMSs, the general rule is to alternate double quotation marks and single quotation marks to indicate nesting.

The code that follows inserts a second row into the table STUDENT . Note that we can just reuse the Statement object stmt rather than having to create a new one for each execution.

```
stmt.executeUpdate("INSERT INTO STUDENT " + "VALUES ('yStudent', 502,
_B.Tech','III')");
```

### **Getting Data from a Table**

Now that the table STUDENT has values in it, we can write a SELECT statement to access those values. The star (\*) in the following SQL statement indicates that all columns should be selected. Since there is no WHERE clause to narrow down the rows from which to select, the following SQL statement selects the whole table:

```
SQL> SELECT * FROM STUDENT;
```

### **Retrieving Values from Result Sets**

We now show how you send the above SELECT statements from a program written in the Java programming language and how you get the results we showed.

JDBC returns results in a ResultSet object, so we need to declare an instance of the class ResultSet to hold our results. The following code demonstrates declaring the ResultSet object rs and assigning the results of our earlier query to it:

```
ResultSet rs = stmt.executeQuery("SELECT S_NAME, YEAR FROM STUDENT");
```

The following code accesses the values stored in the current row of rs. Each time the method next is invoked, the next row becomes the current row, and the loop continues until there are no more rows in rs .

```
String query = "SELECT COF_NAME, PRICE FROM STUDENT";
ResultSet rs = stmt.executeQuery(query);
while (rs.next())
{
 String s = rs.getString("S_NAME");
 Integer i = rs.getInt("S_ID");
 String c = rs.getString("COURSE");
 String y = rs.getString("YEAR");
 System.out.println(i + " " + s + " " + c + " " + y);
}
```

### **Updating Tables**

Suppose that after a period of time we want update the YEAR column in the table STUDENT. The SQL statement to update one row might look like this:

```
String updateString = "UPDATE STUDENT " +
"SET YEAR = IV WHERE S-NAME LIKE 'yStudent'";
```

Using the Statement object stmt , this JDBC code executes the SQL statement contained in updateString :

```
stmt.executeUpdate(updateString);
```

### **Using try and catch Blocks:**

Something else all the sample applications include is try and catch blocks. These are the Java programming language's mechanism for handling exceptions. Java requires that when a method throws an exception, there be some mechanism to handle it. Generally a catch block will catch the exception and specify what happens (which you may choose to be nothing). In the sample code, we use two try blocks and two catch blocks. The first try block contains the method `Class.forName`, from the `java.lang` package. This method throws a `ClassNotFoundException`, so the catch block immediately following it deals with that exception. The second try block contains JDBC methods, which all throw `SQLExceptions`, so one catch block at the end of the application can handle all of the rest of the exceptions that might be thrown because they will all be `SQLException` objects.

### **Retrieving Exceptions**

JDBC lets you see the warnings and exceptions generated by your DBMS and by the Java compiler. To see exceptions, you can have a catch block print them out. For example, the following two catch blocks from the sample code print out a message explaining the exception:

```
Try
{
 // Code that could generate an exception goes here.
 // If an exception is generated, the catch block below
 // will print out information about it.
} catch(SQLException ex)
{
 System.err.println("SQLException: " + ex.getMessage());
}
```

## **JDBC - Database Connections**

After you've installed the appropriate driver, it is time to establish a database connection using JDBC.

The programming involved to establish a JDBC connection is fairly simple. Here are these simple four steps –

- **Import JDBC Packages** – Add **import** statements to your Java program to import required classes in your Java code.
- **Register JDBC Driver** – This step causes the JVM to load the desired driver implementation into memory so it can fulfill your JDBC requests.
- **Database URL Formulation** – This is to create a properly formatted address that points to the database to which you wish to connect.

- **Create Connection Object** – Finally, code a call to the *DriverManager* object's *getConnection( )* method to establish actual database connection.

### Import JDBC Packages

The **Import** statements tell the Java compiler where to find the classes you reference in your code and are placed at the very beginning of your source code.

To use the standard JDBC package, which allows you to select, insert, update, and delete data in SQL tables, add the following *imports* to your source code –

```
import java.sql.* ; // for standard JDBC programs
import java.math.* ; // for BigDecimal and BigInteger support
```

### Register JDBC Driver

You must register the driver in your program before you use it. Registering the driver is the process by which the Oracle driver's class file is loaded into the memory, so it can be utilized as an implementation of the JDBC interfaces.

You need to do this registration only once in your program. You can register a driver in one of two ways.

#### Approach I - Class.forName()

The most common approach to register a driver is to use Java's **Class.forName()** method, to dynamically load the driver's class file into memory, which automatically registers it. This method is preferable because it allows you to make the driver registration configurable and portable.

The following example uses *Class.forName( )* to register the Oracle driver –

```
try {
 Class.forName("oracle.jdbc.driver.OracleDriver");
}
catch(ClassNotFoundException ex) {
 System.out.println("Error: unable to load driver class!");
 System.exit(1);
}
```

You can use **getInstance()** method to work around noncompliant JVMs, but then you'll have to code for two extra Exceptions as follows –

```
try {
```

```

Class.forName("oracle.jdbc.driver.OracleDriver").newInstance();
}

catch(ClassNotFoundException ex) {
 System.out.println("Error: unable to load driver class!");
 System.exit(1);
}

catch(IllegalAccessException ex) {
 System.out.println("Error: access problem while loading!");
 System.exit(2);
}

catch(InstantiationException ex) {
 System.out.println("Error: unable to instantiate driver!");
 System.exit(3);
}

```

#### Approach II - DriverManager.registerDriver()

The second approach you can use to register a driver, is to use the static **DriverManager.registerDriver()** method.

You should use the *registerDriver()* method if you are using a non-JDK compliant JVM, such as the one provided by Microsoft.

The following example uses *registerDriver()* to register the Oracle driver –

```

try {
 Driver myDriver = new oracle.jdbc.driver.OracleDriver();
 DriverManager.registerDriver(myDriver);
}

catch(ClassNotFoundException ex) {
 System.out.println("Error: unable to load driver class!");
 System.exit(1);
}

```

#### Database URL Formulation

After you've loaded the driver, you can establish a connection using the **DriverManager.getConnection()** method. For easy reference, let me list the three overloaded *DriverManager.getConnection()* methods –

- *getConnection(String url)*
- *getConnection(String url, Properties prop)*

- `getConnection(String url, String user, String password)`

Here each form requires a database **URL**. A database URL is an address that points to your database.

Formulating a database URL is where most of the problems associated with establishing a connection occurs.

Following table lists down the popular JDBC driver names and database URL.

<b>RDBMS</b>	<b>JDBC driver name</b>	<b>URL format</b>
MySQL	<code>com.mysql.jdbc.Driver</code>	<b>jdbc:mysql://hostname/</b> databaseName
ORACLE	<code>oracle.jdbc.driver.OracleDriver</code>	<b>jdbc:oracle:thin:@hostname:port</b> Number:databaseName
DB2	<code>COM.ibm.db2.jdbc.net.DB2Driver</code>	<b>jdbc:db2:hostname:port</b> Number/databaseName
Sybase	<code>com.sybase.jdbc.SybDriver</code>	<b>jdbc:sybase:Tds:hostname: port</b> Number/databaseName

All the highlighted part in URL format is static and you need to change only the remaining part as per your database setup.

### Create Connection Object

We have listed down three forms of **DriverManager.getConnection()** method to create a connection object.

#### Using a Database URL with a username and password

The most commonly used form of `getConnection()` requires you to pass a database URL, a *username*, and a *password* –

Assuming you are using Oracle's **thin** driver, you'll specify a host:port:databaseName value for the database portion of the URL.

If you have a host at TCP/IP address 192.0.0.1 with a host name of amrood, and your Oracle listener is configured to listen on port 1521, and your database name is EMP, then complete database URL would be –  
`jdbc:oracle:thin:@amrood:1521:EMP`

Now you have to call `getConnection()` method with appropriate username and password to get a **Connection** object as follows –

```
String URL = "jdbc:oracle:thin:@amrood:1521:EMP";
```

```
String USER = "username";
String PASS = "password"
Connection conn = DriverManager.getConnection(URL, USER, PASS);
```

#### Using Only a Database URL

A second form of the DriverManager.getConnection( ) method requires only a database URL –

```
DriverManager.getConnection(String url);
```

However, in this case, the database URL includes the username and password and has the following general form –

jdbc:oracle:driver:username/password@database

So, the above connection can be created as follows –

```
String URL = "jdbc:oracle:thin:username/password@amrood:1521:EMP";
Connection conn = DriverManager.getConnection(URL);
```

#### Using a Database URL and a Properties Object

A third form of the DriverManager.getConnection( ) method requires a database URL and a Properties object –

```
DriverManager.getConnection(String url, Properties info);
```

A Properties object holds a set of keyword-value pairs. It is used to pass driver properties to the driver during a call to the getConnection() method.

To make the same connection made by the previous examples, use the following code –

```
import java.util.*;
String URL = "jdbc:oracle:thin:@amrood:1521:EMP";
Properties info = new Properties();
info.put("user", "username");
info.put("password", "password");
```

```
Connection conn = DriverManager.getConnection(URL, info);
```

#### Closing JDBC Connections

At the end of your JDBC program, it is required explicitly to close all the connections to the database to end each database session. However, if you forget, Java's garbage collector will close the connection when it cleans up stale objects.

Relying on the garbage collection, especially in database programming, is a very poor programming practice. You should make a habit of always closing the connection with the `close()` method associated with connection object.

To ensure that a connection is closed, you could provide a 'finally' block in your code. A *finally* block always executes, regardless of an exception occurs or not.

To close the above opened connection, you should call `close()` method as follows –

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

## **Statements:**

There are three types of statements in JDBC namely, Statement, Prepared Statement, Callable statement.

### **Statement**

The Statement interface represents the static SQL statement. It helps you to create a general purpose SQL statements using Java.

#### **Creating a statement**

You can create an object of this interface using the `createStatement()` method of the **Connection** interface.

Create a statement by invoking the `createStatement()` method as shown below.

```
Statement stmt = null;
```

```
try {
```

```
stmt = conn.createStatement();
```

```
...
```

```
}
```

```
catch (SQLException e) {
```

```
...
```

```
}
```

```
finally {
```

```

...
}

```

## Executing the Statement object

Once you have created the statement object you can execute it using one of the execute methods namely, execute(), executeUpdate() and, executeQuery().

- **execute():** This method is used to execute SQL DDL statements, it returns a boolean value specifying whether the ResultSet object can be retrieved.
- **executeUpdate():** This method is used to execute statements such as insert, update, delete. It returns an integer value representing the number of rows affected.
- **executeQuery():** This method is used to execute statements that returns tabular data (example SELECT statement). It returns an object of the class ResultSet.

## Prepared Statement

The **PreparedStatement** interface extends the Statement interface. It represents a precompiled SQL statement which can be executed multiple times. This accepts parameterized SQL queries and you can pass 0 or more parameters to this query.

Initially, this statement uses place holders “?” instead of parameters, later on, you can pass arguments to these dynamically using the **setXXX()** methods of the **PreparedStatement** interface.

## Creating a PreparedStatement

You can create an object of the **PreparedStatement** (interface) using the **prepareStatement()** method of the Connection interface. This method accepts a query (parameterized) and returns a PreparedStatement object.

When you invoke this method the Connection object sends the given query to the database to compile and save it. If the query got compiled successfully then only it returns the object.

To compile a query, the database doesn't require any values so, you can use (zero or more) **placeholders** (Question marks “?”) in the place of values in the query.

For example, if you have a table named **Employee** in the database created using the following query:

```

CREATE TABLE Employee(Name VARCHAR(255), Salary INT NOT NULL, Location
VARCHAR(255));

```

Then, you can use a **PreparedStatement** to insert values into it as shown below.

```

//Creating a Prepared Statement
Web Programming

```

```
String query="INSERT INTO Employee(Name, Salary, Location)VALUES(?, ?, ?);
```

```
Statement pstmt = con.prepareStatement(query);
```

## **Setting values to the place holders**

The **PreparedStatement** interface provides several setter methods such as `setInt()`, `setFloat()`, `setArray()`, `setDate()`, `setDouble()` etc.. to set values to the place holders of the prepared statement.

These methods accepts two arguments one is an integer value representing the placement index of the place holder and the other is an int or, String or, float etc... representing the value you need to insert at that particular position.

Once you have created a prepared statement object (with place holders) you can set values to the place holders of the prepared statement using the setter methods as shown below:

```
pstmt.setString(1, "Amit");
```

```
pstmt.setInt(2, 3000);
```

```
pstmt.setString(3, "Hyderabad");
```

## **Executing the Prepared Statement**

Once you have created the **PreparedStatement** object you can execute it using one of the **execute()** methods of the **PreparedStatement** interface namely, `execute()`, `executeUpdate()` and, `executeQuery()`.

- **execute():** This method executes normal static SQL statements in the current prepared statement object and returns a boolean value.
- **executeQuery():** This method executes the current prepared statement and returns a **ResultSet** object.
- **executeUpdate():** This method executes SQL DML statements such as insert update or delete in the current Prepared statement. It returns an integer value representing the number of rows affected.

## **CallableStatement**

The **CallableStatement** interface provides methods to execute stored procedures. Since the JDBC API provides a stored procedure SQL escape syntax, you can call stored procedures of all RDBMS in a single standard way.

## **Creating a CallableStatement**

You can create an object of the **CallableStatement** (interface) using the **prepareCall()** method of the **Connection** interface.

This method accepts a string variable representing a query to call the stored procedure and returns a **CallableStatement** object.

A CallableStatement can have input parameters or, output parameters or, both. To pass input parameters to the procedure call you can use place holder and set values to these using the setter methods (`setInt()`, `setString()`, `setFloat()`) provided by the CallableStatement interface.

Suppose, you have a procedure name `myProcedure` in the database you can prepare a callable statement as:

```
//Preparing a CallableStatement
```

```
CallableStatement cstmt = con.prepareCall("{call myProcedure(?, ?, ?)}");
```

### **Setting values to the input parameters**

You can set values to the input parameters of the procedure call using the setter methods.

These accept two arguments one is an integer value representing the placement index of the input parameter and the other is an int or, String or, float etc... representing the value you need to pass an input parameter to the procedure.

**Note:** Instead of index you can also pass the name of the parameter in String format.

```
cstmt.setString(1, "Raghav");
cstmt.setInt(2, 3000);
cstmt.setString(3, "Hyderabad");
```

### **Executing the Callable Statement**

Once you have created the CallableStatement object you can execute it using one of the `execute()` method.

```
cstmt.execute();
```

### **Catching Database Results**

The SQL statements that read data from a database query, return the data in a result set. The `SELECT` statement is the standard way to select rows from a database and view them in a result set.

The `java.sql.ResultSet` interface represents the result set of a database query.

A ResultSet object maintains a cursor that points to the current row in the result set. The term "result set" refers to the row and column data contained in a ResultSet object.

The methods of the ResultSet interface can be broken down into three categories –

- **Navigational methods** – Used to move the cursor around.
- **Get methods** – Used to view the data in the columns of the current row being pointed by the cursor.

- **Update methods** – Used to update the data in the columns of the current row. The updates can then be updated in the underlying database as well.

The cursor is movable based on the properties of the ResultSet. These properties are designated when the corresponding Statement that generates the ResultSet is created.

JDBC provides the following connection methods to create statements with desired ResultSet –

- **createStatement(int RSType, int RSConcurrency);**
- **prepareStatement(String SQL, int RSType, int RSConcurrency);**
- **prepareCall(String sql, int RSType, int RSConcurrency);**

The first argument indicates the type of a ResultSet object and the second argument is one of two ResultSet constants for specifying whether a result set is read-only or updatable.

#### Type of ResultSet

The possible RSType are given below. If you do not specify any ResultSet type, you will automatically get one that is TYPE\_FORWARD\_ONLY.

Type	Description
ResultSet.TYPE_FORWARD_ONLY	The cursor can only move forward in the result set.
ResultSet.TYPE_SCROLL_INSENSITIVE	The cursor can scroll forward and backward, and the result set is not sensitive to changes made by others to the database that occur after the result set was created.
ResultSet.TYPE_SCROLL_SENSITIVE	The cursor can scroll forward and backward, and the result set is sensitive to changes made by others to the database that occur after the result set was created.

#### Concurrency of ResultSet

The possible RSConcurrency are given below. If you do not specify any Concurrency type, you will automatically get one that is CONCUR\_READ\_ONLY.

Concurrency	Description
ResultSet.CONCUR_READ_ONLY	Creates a read-only result set. This is the default
ResultSet.CONCUR_UPDATABLE	Creates an updateable result set.

All our examples written so far can be written as follows, which initializes a Statement object to create a forward-only, read only ResultSet object –

```
try {
 Statement stmt = conn.createStatement(ResultSet.TYPE_FORWARD_ONLY,
 ResultSet.CONCUR_READ_ONLY);
}
catch(Exception ex) {

}
finally {
```

....  
}

### Navigating a Result Set

There are several methods in the ResultSet interface that involve moving the cursor, including –

S.N.	<b>Methods &amp; Description</b>
1	<b>public void beforeFirst() throws SQLException</b> Moves the cursor just before the first row.
2	<b>public void afterLast() throws SQLException</b> Moves the cursor just after the last row.
3	<b>public boolean first() throws SQLException</b> Moves the cursor to the first row.
4	<b>public void last() throws SQLException</b> Moves the cursor to the last row.
5	<b>public boolean absolute(int row) throws SQLException</b> Moves the cursor to the specified row.
6	<b>public boolean relative(int row) throws SQLException</b> Moves the cursor the given number of rows forward or backward, from where it is currently pointing.
7	<b>public boolean previous() throws SQLException</b> Moves the cursor to the previous row. This method returns false if the previous row is off the result set.
8	<b>public boolean next() throws SQLException</b> Moves the cursor to the next row. This method returns false if there are no more rows in the result set.
9	<b>public int getRow() throws SQLException</b> Returns the row number that the cursor is pointing to.
10	<b>public void moveToInsertRow() throws SQLException</b> Moves the cursor to a special row in the result set that can be used to insert a new row into the database. The current cursor location is remembered.
11	<b>public void moveToCurrentRow() throws SQLException</b>

	Moves the cursor back to the current row if the cursor is currently at the insert row; otherwise, this method does nothing
--	----------------------------------------------------------------------------------------------------------------------------

For a better understanding, let us study [Navigate - Example Code](#).

#### Viewing a Result Set

The ResultSet interface contains dozens of methods for getting the data of the current row.

There is a get method for each of the possible data types, and each get method has two versions –

- One that takes in a column name.
- One that takes in a column index.

For example, if the column you are interested in viewing contains an int, you need to use one of the getInt() methods of ResultSet –

S.N.	Methods & Description
1	<b>public int getInt(String columnName) throws SQLException</b> Returns the int in the current row in the column named columnName.
2	<b>public int getInt(int columnIndex) throws SQLException</b> Returns the int in the current row in the specified column index. The column index starts at 1, meaning the first column of a row is 1, the second column of a row is 2, and so on.

Similarly, there are get methods in the ResultSet interface for each of the eight Java primitive types, as well as common types such as java.lang.String, java.lang.Object, and java.net.URL.

There are also methods for getting SQL data types java.sql.Date, java.sql.Time, java.sql.Timestamp, java.sql.Clob, and java.sql.Blob. Check the documentation for more information about using these SQL data types.

For a better understanding, let us study [Viewing - Example Code](#).

#### Updating a Result Set

The ResultSet interface contains a collection of update methods for updating the data of a result set.

As with the get methods, there are two update methods for each data type –

- One that takes in a column name.
- One that takes in a column index.

For example, to update a String column of the current row of a result set, you would use one of the following updateString() methods –

S.N.	Methods & Description
1	<b>public void updateString(int columnIndex, String s) throws SQLException</b> Changes the String in the specified column to the value of s.
2	<b>public void updateString(String columnName, String s) throws SQLException</b>

Similar to the previous method, except that the column is specified by its name instead of its index.

There are update methods for the eight primitive data types, as well as String, Object, URL, and the SQL data types in the java.sql package.

Updating a row in the result set changes the columns of the current row in the ResultSet object, but not in the underlying database. To update your changes to the row in the database, you need to invoke one of the following methods.

S.N.	Methods & Description
1	<b>public void updateRow()</b> Updates the current row by updating the corresponding row in the database.
2	<b>public void deleteRow()</b> Deletes the current row from the database
3	<b>public void refreshRow()</b> Refreshes the data in the result set to reflect any recent changes in the database.
4	<b>public void cancelRowUpdates()</b> Cancels any updates made on the current row.
5	<b>public void insertRow()</b> Inserts a row into the database. This method can only be invoked when the cursor is pointing to the insert row.

## Networking– InetAddress class

**Java InetAddress** class represents an IP address. The java.net.InetAddress class provides methods to get the IP of any host name *for example* www.javatpoint.com, www.google.com, www.facebook.com, etc.

An IP address is represented by 32-bit or 128-bit unsigned number. An instance of InetAddress represents the IP address with its corresponding host name. There are two types of addresses: Unicast and Multicast. The Unicast is an identifier for a single interface whereas Multicast is an identifier for a set of interfaces.

Moreover, InetAddress has a cache mechanism to store successful and unsuccessful host name resolutions.

### IP Address

- An IP address helps to identify a specific resource on the network using a numerical representation.
- Most networks combine IP with TCP (Transmission Control Protocol). It builds a virtual bridge among the destination and the source.

There are two versions of IP address:

*1. IPv4*

IPv4 is the primary Internet protocol. It is the first version of IP deployed for production in the ARAPNET in 1983. It is a widely used IP version to differentiate devices on network using an addressing scheme. A 32-bit addressing scheme is used to store  $2^{32}$  addresses that is more than 4 million addresses.

**Features of IPv4:**

- It is a connectionless protocol.
- It utilizes less memory and the addresses can be remembered easily with the class based addressing scheme.
- It also offers video conferencing and libraries.

*2. IPv6*

IPv6 is the latest version of Internet protocol. It aims at fulfilling the need of more internet addresses. It provides solutions for the problems present in IPv4. It provides 128-bit address space that can be used to form a network of 340 undecillion unique IP addresses. IPv6 is also identified with a name IPng (Internet Protocol next generation).

**Features of IPv6:**

- It has a stateful and stateless both configurations.
- It provides support for quality of service (QoS).
- It has a hierarchical addressing and routing infrastructure.

**TCP/IP Protocol**

- TCP/IP is a communication protocol model used connect devices over a network via internet.
- TCP/IP helps in the process of addressing, transmitting, routing and receiving the data packets over the internet.
- The two main protocols used in this communication model are:
  1. TCP i.e. Transmission Control Protocol. TCP provides the way to create a communication channel across the network. It also helps in transmission of packets at sender end as well as receiver end.
  2. IP i.e. Internet Protocol. IP provides the address to the nodes connected on the internet. It uses a gateway computer to check whether the IP address is correct and the message is forwarded correctly or not.

**Java InetAddress Class Methods**

<b>Method</b>	<b>Description</b>
public static InetAddress getByName(String host) throws UnknownHostException	It returns the instance of InetAddress containing LocalHost IP and name.
public static InetAddress getLocalHost() throws UnknownHostException	It returns the instance of InetAddress containing local host name and address.
public String getHostName()	It returns the host name of the IP address.
public String getHostAddress()	It returns the IP address in string format.

### Example of Java InetAddress Class

Let's see a simple example of InetAddress class to get ip address of [www.javatpoint.com](http://www.javatpoint.com) website.

#### **InetDemo.java**

```

1. import java.io.*;
2. import java.net.*;
3. public class InetDemo{
4. public static void main(String[] args){
5. try{
6. InetAddress ip=InetAddress.getByName("www.javatpoint.com");
7.
8. System.out.println("Host Name: "+ip.getHostName());
9. System.out.println("IP Address: "+ip.getHostAddress());
10. }catch(Exception e){System.out.println(e);}
11. }
12. }
```

#### **Test it Now**

#### **Output:**

```

Host Name: www.javatpoint.com
IP Address: 172.67.196.82
```

*Program to demonstrate methods of InetAddress class*

#### **InetDemo2.java**

```

1. import java.net.Inet4Address;
2. import java.util.Arrays;
3. import java.net.InetAddress;
4. public class InetDemo2
5. {
```

```

6. public static void main(String[] arg) throws Exception
7. {
8. InetAddress ip = InetAddress.getByName("www.javatpoint.com");
9. InetAddress ip1[] = InetAddress.getAllByName("www.javatpoint.com");
10. byte addr[]={72, 3, 2, 12};
11. System.out.println("ip : "+ip);
12. System.out.print("\nip1 : "+ip1);
13. InetAddress ip2 = InetAddress.getByAddress(addr);
14. System.out.print("\nip2 : "+ip2);
15. System.out.print("\nAddress : " +Arrays.toString(ip.getAddress()));
16. System.out.print("\nHost Address : " +ip.getHostAddress());
17. System.out.print("\nisAnyLocalAddress : " +ip.isAnyLocalAddress());
18. System.out.print("\nisLinkLocalAddress : " +ip.isLinkLocalAddress());
19. System.out.print("\nisLoopbackAddress : " +ip.isLoopbackAddress());
20. System.out.print("\nisMCGlobal : " +ip.isMCGlobal());
21. System.out.print("\nisMCLinkLocal : " +ip.isMCLinkLocal());
22. System.out.print("\nisMCNodeLocal : " +ip.isMCNodeLocal());
23. System.out.print("\nisMCOrgLocal : " +ip.isMCOrgLocal());
24. System.out.print("\nisMCSiteLocal : " +ip.isMCSiteLocal());
25. System.out.print("\nisMulticastAddress : " +ip.isMulticastAddress());
26. System.out.print("\nisSiteLocalAddress : " +ip.isSiteLocalAddress());
27. System.out.print("\nhashCode : " +ip.hashCode());
28. System.out.print("\n Is ip1 == ip2 : " +ip.equals(ip2));
29. }
30. }

```

### **31. OUTPUT**

```

C:\Users\WIN 8.1>cd Desktop
C:\Users\WIN 8.1\Desktop>javac InetDemo2.java
C:\Users\WIN 8.1\Desktop>java InetDemo2
ip : www.javatpoint.com/172.67.196.82
ip1 : [Ljava.net.InetAddress;@7852e922
ip2 : /72.3.2.12
Address : [-84, 67, -60, 82]
Host Address : 172.67.196.82
isAnyLocalAddress : false
isLinkLocalAddress : false
isLoopbackAddress : false
isMCGlobal : false
isMCLinkLocal : false
isMCNodeLocal : false
isMCOrgLocal : false
isMCSiteLocal : false
isMulticastAddress : false
isSiteLocalAddress : false
hashCode : -1404844974
 Is ip1 == ip2 : false
C:\Users\WIN 8.1\Desktop>

```

## URL class

JDBC provides the URL to identify the database, so we can easily recognize the required driver and we can connect it. Basically JDBC URL we can use as database connection URL as per user requirement. When the driver loaded successfully we need to specify the required database connection URL to connect the database that the user wants. We know that the JDBC URL always starts with the JDBC keyword for the database connection; basically, the URL depends on the JDBC driver. We also need to provide the different parameters with the JDBC URL that is port number, hostname, database name, user name, and password, etc.

### Syntax:

```
specified protocol name//[specified host name]/[specified database name][username and password]
```

## How URL works in JDBC?

For establishing a connection with the database we need to follow the same step as follows.

**Import JDBC Packages:** First step we need to import the JDBC packages into the Java program that we require the class in code.

**Register the JDBC Driver:** After importing the class we need to load the JVM to fulfill that is it loaded the required driver as well as memory for JDBC request.

**Database URL Formation:** In this step, we need to provide the correct parameter to connect the specified database that we already discussed in the above point.

**Create the Connection Object:** After the formation of the URL, we need to create the object of connection that means we can call the DriverManager with getConnection() methods to establish the connection with a specified database name.

Now let's see in detail how we can import the JDBC Driver as follows.

Basically, the import statement is used to compile the java program and is also used to find the classes that are helpful to implement the source code as per user requirements. By using these standard packages, we can perform different operations such as insertion, delete and update as per user requirements.

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

Now let's see how we can register the JDBC Driver as follows.

We just need to import the driver before using it. Enlisting the driver is the cycle by which the Oracle driver's class document is stacked into the memory, so it tends to be used as an execution of the JDBC interfaces.

You need to do this enrollment just a single time in your program. You can enlist a driver in one of two different ways.

### **1. By using Class.forName():**

The most widely recognized way to deal with registering a driver is to utilize Java's Class.forName() technique, to progressively stack the driver's class document into memory, which naturally enlists it. This technique is ideal since it permits you to make the driver enrollment configurable and compact.

### **2. By using DriverManager.registerDriver():**

The second methodology you can use to enroll a driver is to utilize the static DriverManager.registerDriver() strategy.

You should utilize the registerDriver() technique in case you are utilizing a non-JDK agreeable JVM, for example, the one given by Microsoft.

After you've stacked the driver, you can set up an association utilizing the DriverManager.getConnection() technique. JDBC provides the different JDBC drivers for the different database systems and we can utilize them as per the user requirement.

### ***1. MySQL JDBC URL format:***

This is the first JDBC URL format that can be used in MySQL to establish the connection with the required database name. The format of this URL is as follows.

```
(Connection con_obj = DriverManager.getConnection(specified_jdbcUrl, user defined username, user defined
password))
```

### **Explanation**

In the above format, we use DriverManager.getConnection method to establish the connection with the database; here we need to pass the specified JDBC URL as well as we need to pass the username and password. The username and password fields are depending on the user. In JDBC URL we need to pass all parameters that we require to make the connection such as database name, protocol, etc.

### ***2. Microsoft SQL Server URL format:***

This is another famous URL format for the database system. Suppose we need to connect to the Microsoft SQL Server from a Java application at that time we can use the below-mentioned format as follows.

```
jdbc:sqlserver://[specified serverName[\ specified instanceName][:required portNumber]][;property(that user
defined properties)]
```

### **Explanation**

In the above syntax, we need to mention the server name that is the address of the server, or we can say that domain name or IP address. Also, we need to mention the instance name for server connection if we leave then it uses the default. In the same way, we can use port numbers and properties.

### **3. PostgreSQL JDBC URL format:**

PostgreSQL is a famous open-source database system. So we can use the below-mentioned JDBC format as follows.

Jdbc:postgresql://hostname:port number/specify database name and properties.

#### **Examples**

Now let's see different examples of JDBC URLs for better understanding as follows.

```
import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.Statement;

public class connection_t {

 public static void main(String args[]){

 String m_url = " jdbc:mysql://localhost ";

 Connection con_obj = DriverManager.getConnection(m_url, "root", "root");

 System.out.println("Connection successfully established with database. . .");

 }

}
```

#### **Explanation**

In the above example, we import the dependencies that are required to establish the connection with the database such as SQL. connection, SQL.DriverManger etc. After that, we import the class as shown. Here we also mentioned a connection string with connection parameters such as DriverManager.getConnection() method as shown. The final output or end result of the above example we illustrated by using the following screenshot as follows.

In the same way, we can connect to the Microsoft server and PostgreSQL as per our requirements

**Program** A program is an executable file residing on a disk in a directory. A program is read into memory and is executed by the kernel as a result of an `exec()` function. The `exec()` has six variants, but we only consider the simplest one (`exec()`) in this course.

**Process** An executing instance of a program is called a *process*. Sometimes, *task* is used instead of process with the same meaning. UNIX guarantees that every process has a unique identifier called the *process ID*. The process ID is always a non-negative integer.

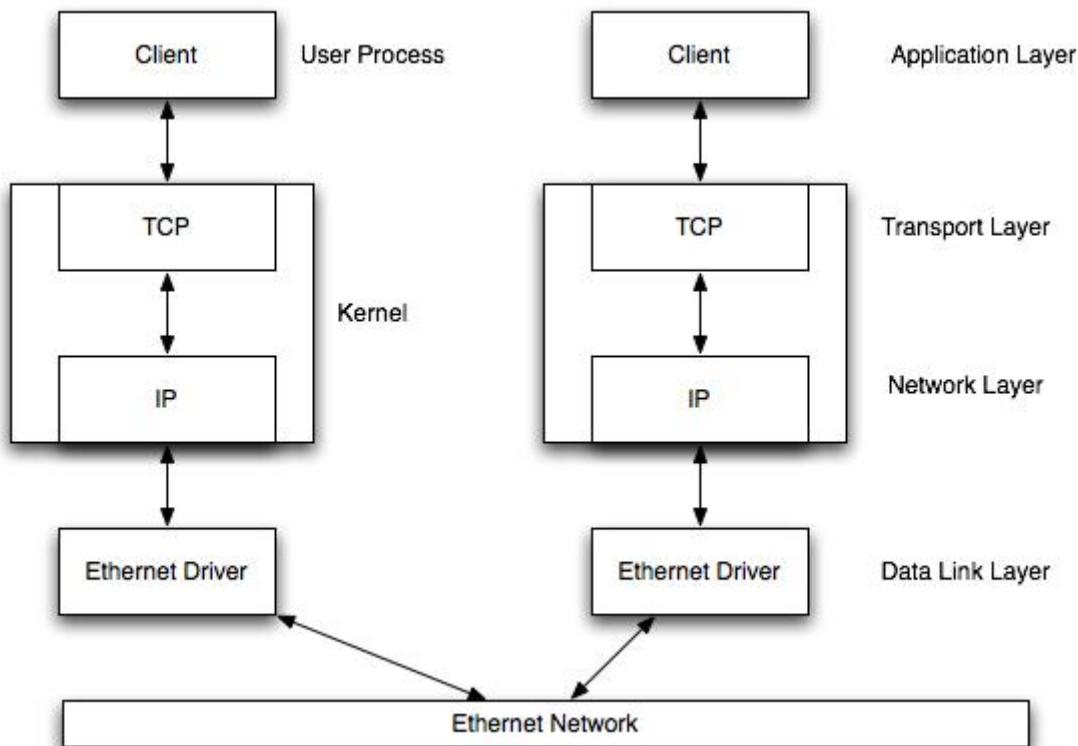
**File descriptors** File descriptors are normally small non-negative integers that the kernel uses to identify the files being accessed by a particular process. Whenever it opens an existing file or creates a new file, the kernel returns a file descriptor that is used to read or write the file. As we will see in this course, sockets are based on a very similar mechanism (socket descriptors).

#### *The client-server model*

The client-server model is one of the most used communication paradigms in networked systems. Clients normally communicate with one server at a time. From a server's perspective, at any point in time, it is not unusual for a server to be communicating with multiple clients. Client need to know of the existence of and the address of the server, but the server does not need to know the address of (or even the existence of) the client prior to the connection being established

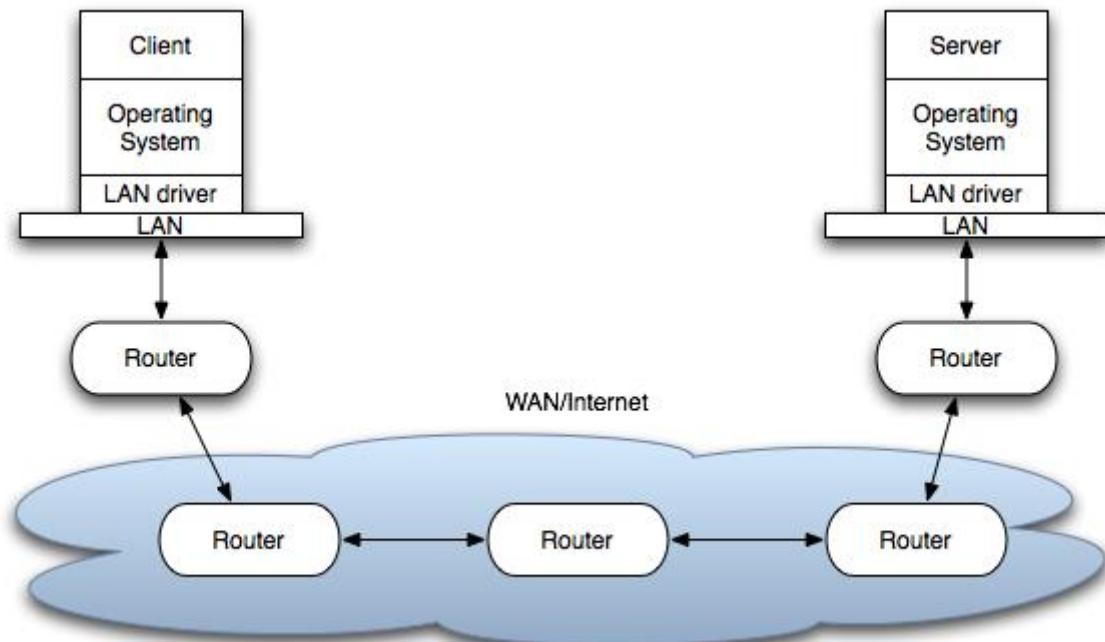
Client and servers communicate by means of multiple layers of network protocols. In this course we will focus on the TCP/IP protocol suite.

The scenario of the client and the server on the same local network (usually called LAN, Local Area Network) is shown in Figure 1



**Figure 1:** Client and server on the same Ethernet communicating using TCP/IP

The client and the server may be in different LANs, with both LANs connected to a Wide Area Network (WAN) by means of *routers*. The largest WAN is the Internet, but companies may have their own WANs. This scenario is depicted in Figure 2.



**Figure 2:** Client and server on different LANs connected through WAN/Internet.

The flow of information between the client and the server goes down the protocol stack on one side, then across the network and then up the protocol stack on the other side.

***User Datagram Protocol (UDP)***

UDP is a simple transport-layer protocol. The application writes a message to a UDP socket, which is then encapsulated in a UDP datagram, which is further encapsulated in an IP datagram, which is sent to the destination.

There is no guarantee that a UDP will reach the destination, that the order of the datagrams will be preserved across the network or that datagrams arrive only once.

The problem of UDP is its lack of reliability: if a datagram reaches its final destination but the checksum detects an error, or if the datagram is dropped in the network, it is not automatically retransmitted.

Each UDP datagram is characterized by a length. The length of a datagram is passed to the receiving application along with the data.

No connection is established between the client and the server and, for this reason, we say that UDP provides a *connection-less service*.

It is described in RFC 768.

***Transmission Control Protocol (TCP)***

TCP provides a *connection oriented service*, since it is based on connections between clients and servers.

TCP provides reliability. When a TCP client send data to the server, it requires an acknowledgement in return. If an acknowledgement is not received, TCP automatically retransmit the data and waits for a longer period of time.

We have mentioned that UDP datagrams are characterized by a length. TCP is instead a byte-stream protocol, without any boundaries at all.

TCP is described in RFC 793, RFC 1323, RFC 2581 and RFC 3390.

**Socket addresses**

IPv4 socket address structure is named `sockaddr_in` and is defined by including the `<netinet/in.h>` header.

The POSIX definition is the following:

```
struct in_addr{
in_addr_t s_addr; /*32 bit IPv4 network byte ordered address*/
};
```

```
struct sockaddr_in {
 uint8_t sin_len; /* length of structure (16)*/
 sa_family_t sin_family; /* AF_INET*/
 in_port_t sin_port; /* 16 bit TCP or UDP port number */
 struct in_addr sin_addr; /* 32 bit IPv4 address*/
 char sin_zero[8]; /* not used but always set to zero */
};
```

The `uint8_t` datatype is unsigned 8-bit integer.

### **Generic Socket Address Structure**

A socket address structure is always passed by reference as an argument to any socket functions. But any socket function that takes one of these pointers as an argument must deal with socket address structures from any of the supported protocol families.

A problem arises in declaring the type of pointer that is passed. With ANSI C, the solution is to use `void *` (the generic pointer type). But the socket functions predate the definition of ANSI C and the solution chosen was to define a generic socket address as follows:

```
struct sockaddr {
 uint8_t sa_len;
 sa_family_t sa_family; /* address family: AD_xxx value */
 char sa_data[14];
};
```

### ***Host Byte Order and Network Byte Order Conversion***

There are two ways to store two bytes in memory: with the lower-order byte at the starting address (*little-endian* byte order) or with the high-order byte at the starting address (*big-endian* byte order). We call them collectively *host byte order*. For example, an Intel processor stores the 32-bit integer as four consecutive bytes in memory in the order 1-2-3-4, where 1 is the most significant byte. IBM PowerPC processors would store the integer in the byte order 4-3-2-1.

Networking protocols such as TCP are based on a specific *network byte order*. The Internet protocols use big-endian byte ordering.

### **The htons(), htonl(), ntohs(), and ntohl() Functions**

The following functions are used for the conversion:

```
#include <netinet/in.h>

uint16_t htons(uint16_t host16bitvalue);

uint32_t htonl(uint32_t host32bitvalue);

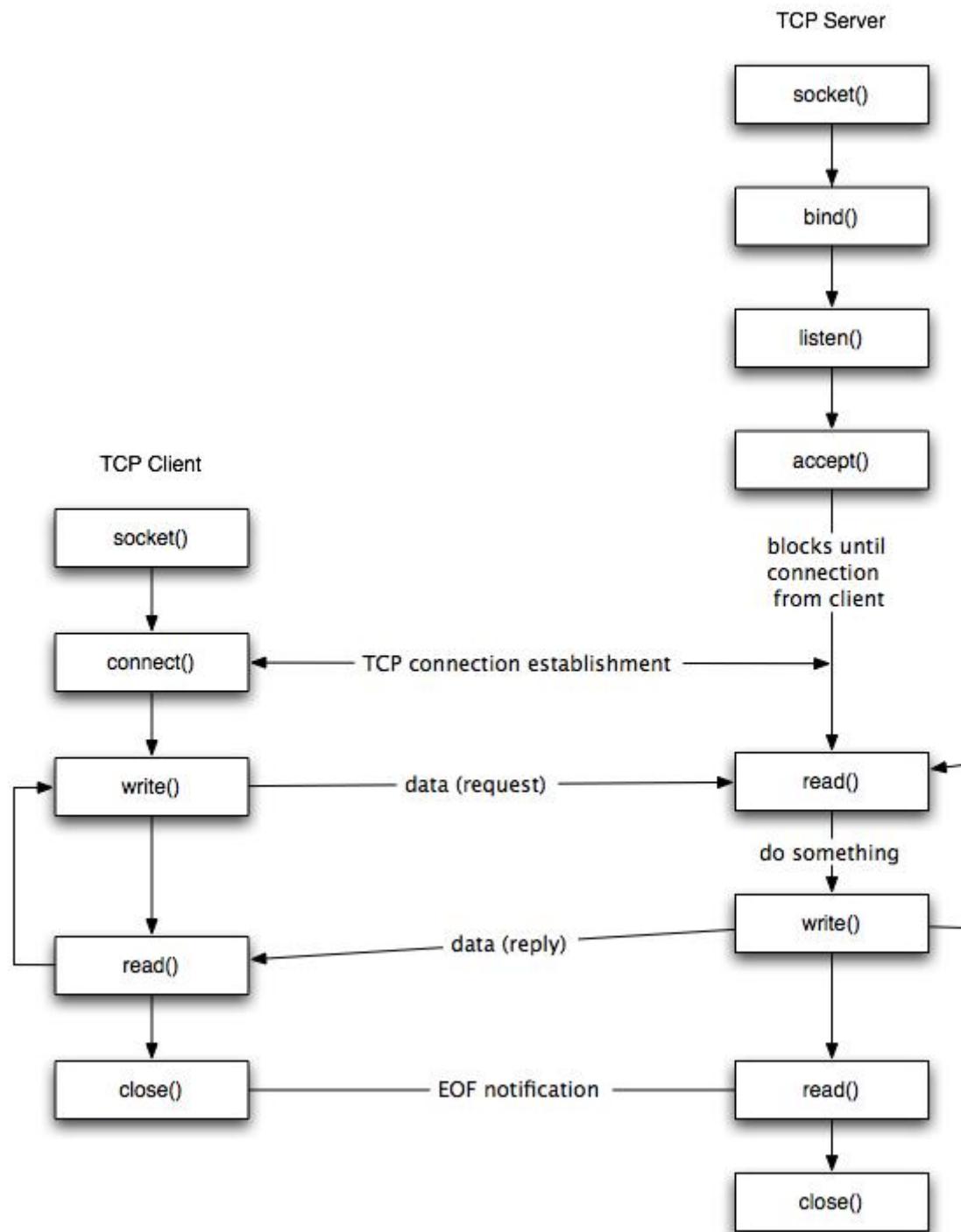
uint16_t ntohs(uint16_t net16bitvalue);
```

```
uint32_t ntohs(uint32_t net32bitvalue);
```

The first two return the value in network byte order (16 and 32 bit, respectively). The latter return the value in host byte order (16 and 32 bit, respectively).

### **TCP Socket API**

The sequence of function calls for the client and a server participating in a TCP connection is presented in Figure 3.



**Figure 3:** TCP client-server.

As shown in the figure, the steps for establishing a TCP socket on the client side are the following:

- Create a socket using the `socket()` function;
- Connect the socket to the address of the server using the `connect()` function;
- Send and receive data by means of the `read()` and `write()` functions.
- Close the connection by means of the `close()` function.

The steps involved in establishing a TCP socket on the server side are as follows:

- Create a socket with the `socket()` function;
- Bind the socket to an address using the `bind()` function;
- Listen for connections with the `listen()` function;
- Accept a connection with the `accept()` function system call. This call typically blocks until a client connects with the server.
- Send and receive data by means of `send()` and `receive()`.
- Close the connection by means of the `close()` function.

**The `socket()` Function**

The first step is to call the `socket` function, specifying the type of communication protocol (TCP based on IPv4, TCP based on IPv6, UDP).

The function is defined as follows:

```
#include <sys/socket.h>

int socket (int family, int type, int protocol);
```

where `family` specifies the protocol family (`AF_INET` for the IPv4 protocols), `type` is a constant described the type of socket (`SOCK_STREAM` for stream sockets and `SOCK_DGRAM` for datagram sockets).

The function returns a non-negative integer number, similar to a file descriptor, that we define *socket descriptor* or -1 on error.

**The `connect()` Function**

The `connect()` function is used by a TCP client to establish a connection with a TCP server/

The function is defined as follows:

```
#include <sys/socket.h>

int connect (int sockfd, const struct sockaddr *servaddr, socklen_t addrlen);
```

where `sockfd` is the socket descriptor returned by the `socket` function.

The function returns 0 if it succeeds in establishing a connection (i.e., successful TCP three-way handshake, -1 otherwise).

The client does not have to call bind() in Section before calling this function: the kernel will choose both an ephemeral port and the source IP if necessary.

### **The bind() Function**

The bind() assigns a local protocol address to a socket. With the Internet protocols, the address is the combination of an IPv4 or IPv6 address (32-bit or 128-bit) address along with a 16 bit TCP port number.

The function is defined as follows:

```
#include <sys/socket.h>

int bind(int sockfd, const struct sockaddr *servaddr, socklen_t addrlen);
```

where sockfd is the socket descriptor, servaddr is a pointer to a protocol-specific address and addrlen is the size of the address structure.

bind() returns 0 if it succeeds, -1 on error.

This use of the generic socket address sockaddr requires that any calls to these functions must cast the pointer to the protocol-specific address structure. For example for an IPv4 socket structure:

```
struct sockaddr_in serv; /* IPv4 socket address structure */

bind(sockfd, (struct sockaddr*) &serv, sizeof(serv))
```

A process can bind a specific IP address to its socket: for a TCP client, this assigns the source IP address that will be used for IP datagrams sent on the sockets. For a TCP server, this restricts the socket to receive incoming client connections destined only to that IP address.

Normally, a TCP client does not bind an IP address to its socket. The kernel chooses the source IP socket is connected, based on the outgoing interface that is used. If a TCP server does not bind an IP address to its socket, the kernel uses the destination IP address of the incoming packets as the server's source address.

bind() allows to specify the IP address, the port, both or neither.

The table below summarizes the combinations for IPv4.

IP Address	IP Port	Result
INADDR_ANY	0	Kernel chooses IP address and port
INADDR_ANY	non zero	Kernel chooses IP address, process specifies port
Local IP address	0	Process specifies IP address, kernel chooses port
Local IP address	non zero	Process specifies IP address and port

Note, the local host address is 127.0.0.1; for example, if you wanted to run your echoServer (see later) on your local machine the your client would connect to 127.0.0.1 with the suitable port.

## The listen() Function

The listen() function converts an unconnected socket into a passive socket, indicating that the kernel should accept incoming connection requests directed to this socket. It is defined as follows:

```
#include <sys/socket.h>
int listen(int sockfd, int backlog);
```

where sockfd is the socket descriptor and backlog is the maximum number of connections the kernel should queue for this socket. The backlog argument provides an hint to the system of the number of outstanding connect requests that it should enqueue on behalf of the process. Once the queue is full, the system will reject additional connection requests. The backlog value must be chosen based on the expected load of the server.

The function listen() return 0 if it succeeds, -1 on error.

## The accept() Function

The accept() is used to retrieve a connect request and convert that into a request. It is defined as follows:

```
#include <sys/socket.h>
int accept(int sockfd, struct sockaddr *cliaddr,
socklen_t *addrlen);
```

where sockfd is a new file descriptor that is connected to the client that called the connect(). The cliaddr and addrlen arguments are used to return the protocol address of the client. The new socket descriptor has the same socket type and address family of the original socket. The original socket passed to accept() is not associated with the connection, but instead remains available to receive additional connect requests. The kernel creates one connected socket for each client connection that is accepted.

If we don't care about the client's identity, we can set the cliaddr and addrlen to NULL. Otherwise, before calling the accept function, the cliaddr parameter has to be set to a buffer large enough to hold the address and set the interger pointed by addrlen to the size of the buffer.

## The send() Function

Since a socket endpoint is represented as a file descriptor, we can use read and write to communicate with a socket as long as it is connected. However, if we want to specify options we need another set of functions.

For example, send() is similar to write() but allows to specify some options. send() is defined as follows:

```
#include <sys/socket.h>
ssize_t send(int sockfd, const void *buf, size_t nbytes, int flags);
```

where buf and nbytes have the same meaning as they have with write. The additional argument flags is used to specify how we want the data to be transmitted. We will not consider the possible options in this course. We will assume it equal to 0.

The function returns the number of bytes if it succeeds, -1 on error.

### **The receive() Function**

The recv() function is similar to read(), but allows to specify some options to control how the data are received. We will not consider the possible options in this course. We will assume it equal to 0.

receive is defined as follows:

```
#include <sys/socket.h>
ssize_t recv(int sockfd, void *buf, size_t nbytes, int flags);
```

The function returns the length of the message in bytes, 0 if no messages are available and peer had done an orderly shutdown, or -1 on error.

### **The close() Function**

The normal close() function is used to close a socket and terminate a TCP socket. It returns 0 if it succeeds, -1 on error. It is defined as follows:

```
#include <unistd.h>
int close(int sockfd);
```

### ***UDP Socket API***

There are some fundamental differences between TCP and UDP sockets. UDP is a connection-less, unreliable, datagram protocol (TCP is instead connection-oriented, reliable and stream based). There are some instances when it makes to use UDP instead of TCP. Some popular applications built around UDP are DNS, NFS, SNMP and for example, some Skype services and streaming media.

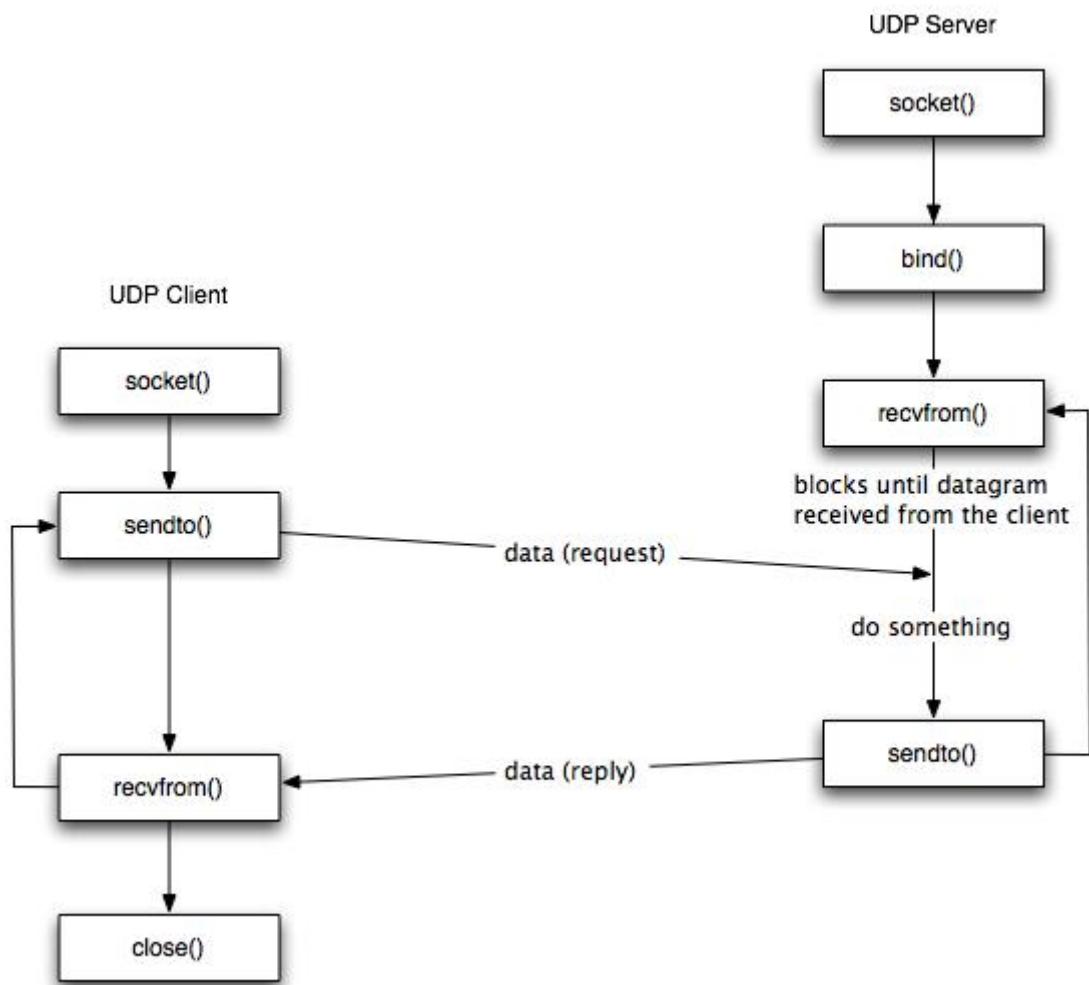
Figure 4 shows the interaction between a UDP client and server. First of all, the client does not establish a connection with the server. Instead, the client just sends a datagram to the server using the sendto function which requires the address of the destination as a parameter. Similarly, the server does not accept a connection from a client. Instead, the server just calls the recvfrom function, which waits until data arrives from some client. recvfrom returns the IP address of the client, along with the datagram, so the server can send a response to the client.

As shown in the Figure, the steps of establishing a UDP socket communication on the client side are as follows:

- Create a socket using the socket() function;
- Send and receive data by means of the recvfrom() and sendto() functions.

The steps of establishing a UDP socket communication on the server side are as follows:

- Create a socket with the socket() function;
- Bind the socket to an address using the bind() function;
- Send and receive data by means of recvfrom() and sendto().



**Figure 4:** UDP client-server.

In this section, we will describe the two new functions `recvfrom()` and `sendto()`.

### The `recvfrom()` Function

This function is similar to the `read()` function, but three additional arguments are required. The `recvfrom()` function is defined as follows:

```
#include <sys/socket.h>

ssize_t recvfrom(int sockfd, void* buff, size_t nbytes,
 int flags, struct sockaddr* from,
 socklen_t *addrlen);
```

The first three arguments `sockfd`, `buff`, and `nbytes`, are identical to the first three arguments of `read` and `write`. `sockfd` is the socket descriptor, `buff` is the pointer to read into, and `nbytes` is number of bytes to read. In our examples we will set all the values of the `flags` argument to 0. The `recvfrom` function fills in the

socket address structure pointed to by from with the protocol address of who sent the datagram. The number of bytes stored in the socket address structure is returned in the integer pointed by addrlen.

The function returns the number of bytes read if it succeeds, -1 on error.

### **The sendto() Function**

This function is similar to the send() function, but three additional arguments are required. The sendto() function is defined as follows:

```
#include <sys/socket.h>
ssize_t sendto(int sockfd, const void *buff, size_t nbytes,
 int flags, const struct sockaddr *to,
 socklen_t addrlen);
```

The first three arguments sockfd, buff, and nbytes, are identical to the first three arguments of recv. sockfd is the socket descriptor, buff is the pointer to write from, and nbytes is number of bytes to write. In our examples we will set all the values of the flags argument to 0. The to argument is a socket address structure containing the protocol address (e.g., IP address and port number) of where the data is sent. addlen specified the size of this socket.

The function returns the number of bytes written if it succeeds, -1 on error.

### ***Concurrent Servers***

There are two main classes of servers, iterative and concurrent. An *iterative* server iterates through each client, handling it one at a time. A *concurrent* server handles multiple clients at the same time. The simplest technique for a concurrent server is to call the fork function, creating one child process for each client. An alternative technique is to use *threads* instead (i.e., light-weight processes). We do not consider this kind of servers in this course.

### **The fork() function**

The fork() function is the only way in Unix to create a new process. It is defined as follows:

```
#include <unistd.h>
pid_t fork(void);
```

The function returns 0 if in child and the process ID of the child in parent; otherwise, -1 on error.

In fact, the function fork() is called once but returns *twice*. It returns once in the calling process (called the parent) with the process ID of the newly created process (its child). It also returns in the child, with a return value of 0. The return value tells whether the current process is the parent or the child.

### **Example**

A typical concurrent server has the following structure:

```
pid_t pid;
int listenfd, connfd;
listenfd = socket(...);

/***fill the socket address with server's well known port***/

bind(listenfd, ...);
listen(listenfd, ...);

for (; ;) {

 connfd = accept(listenfd, ...); /* blocking call */

 if ((pid = fork()) == 0) {

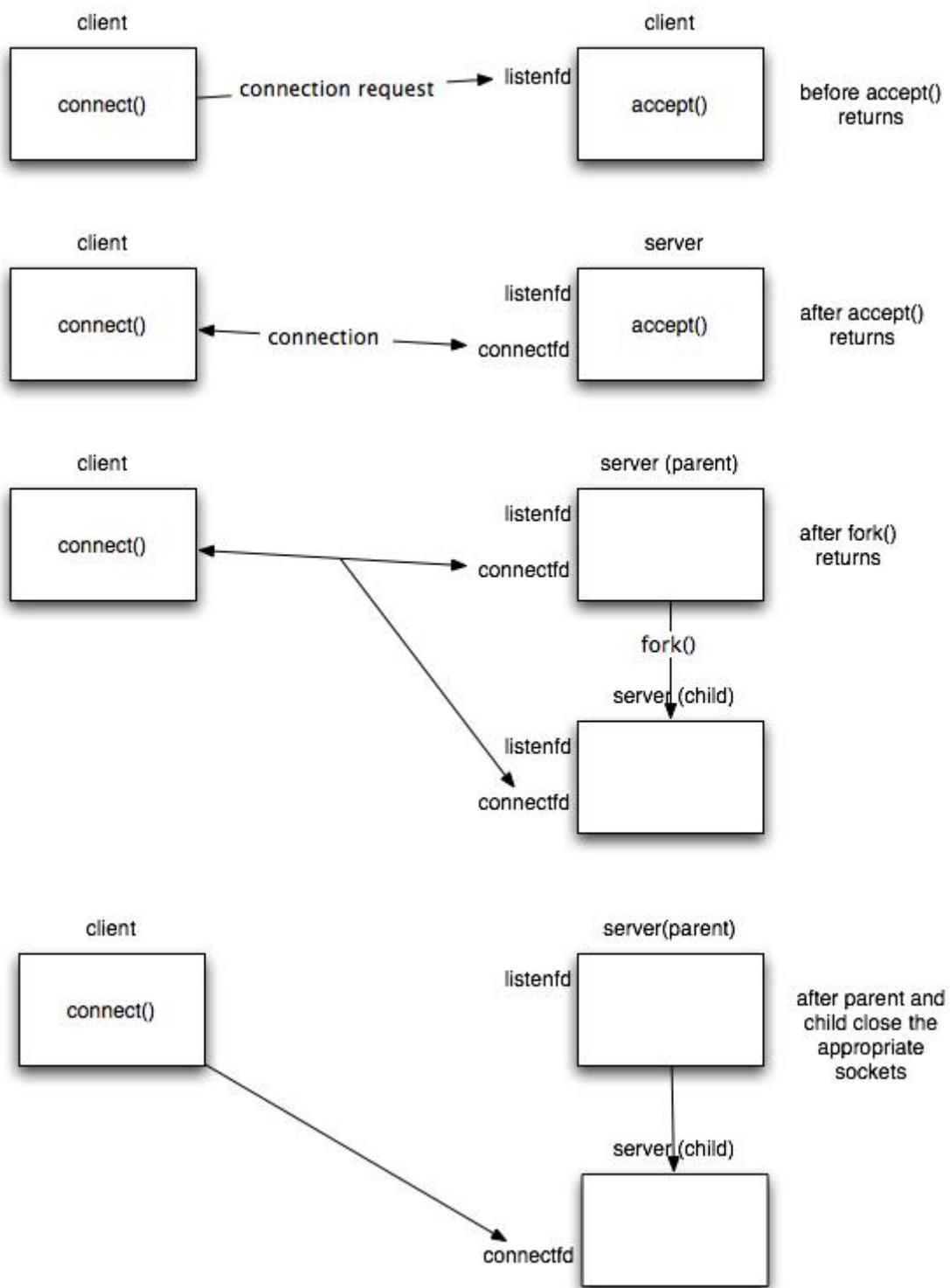
 close(listenfd); /* child closes listening socket */

 /***process the request doing something using connfd ***/
 /* */

 close(connfd);
 exit(0); /* child terminates
 }
 close(connfd); /*parent closes connected socket*/
}
}
```

When a connection is established, accept returns, the server calls fork, and the child process services the client (on the connected socket connfd). The parent process waits for another connection (on the listening socket listenfd). The parent closes the connected socket since the child handles the new client. The interactions among client and server are presented in Figure 5.

---



**Figure 5:** Example of interaction among a client and a concurrent server.

### **TCP Client/Server Examples**

We now present a complete example of the implementation of a TCP based echo server to summarize the concepts presented above. We present an iterative and a concurrent implementation of the server.

**echoClient.c source:** [echoClient.c](#)

**TCP Echo Client**

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>

#define MAXLINE 4096 /*max text line length*/
#define SERV_PORT 3000 /*port*/

int
main(int argc, char **argv)
{
 int sockfd;
 struct sockaddr_in servaddr;
 char sendline[MAXLINE], recvline[MAXLINE];

 //basic check of the arguments
 //additional checks can be inserted
 if (argc !=2) {
 perror("Usage: TCPClient <IP address of the server");
 exit(1);
 }

 //Create a socket for the client
 //If sockfd<0 there was an error in the creation of the socket
 if ((sockfd = socket (AF_INET, SOCK_STREAM, 0)) <0) {
 perror("Problem in creating the socket");
 exit(2);
 }

 //Creation of the socket
 memset(&servaddr, 0, sizeof(servaddr));
 servaddr.sin_family = AF_INET;
 servaddr.sin_addr.s_addr= inet_addr(argv[1]);
 servaddr.sin_port = htons(SERV_PORT); //convert to big-endian order

 //Connection of the client to the socket
 if (connect(sockfd, (struct sockaddr *) &servaddr, sizeof(servaddr))<0) {
 perror("Problem in connecting to the server");
 exit(3);
 }

 while (fgets(sendline, MAXLINE, stdin) != NULL) {

 send(sockfd, sendline, strlen(sendline), 0);

 if (recv(sockfd, recvline, MAXLINE,0) == 0){

```

```

//error: server terminated prematurely
 perror("The server terminated prematurely");
 exit(4);
}
printf("%s", "String received from the server: ");
fputs(recvline, stdout);
}

exit(0);
}

```

**echoServer.c source:** [echoServer.c](#)***TCP Iterative Server***

```

#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>

#define MAXLINE 4096 /*max text line length*/
#define SERV_PORT 3000 /*port*/
#define LISTENQ 8 /*maximum number of client connections */

int main (int argc, char **argv)
{
 int listenfd, connfd, n;
 socklen_t clilen;
 char buf[MAXLINE];
 struct sockaddr_in cliaddr, servaddr;

 //creation of the socket
 listenfd = socket (AF_INET, SOCK_STREAM, 0);

 //preparation of the socket address
 servaddr.sin_family = AF_INET;
 servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
 servaddr.sin_port = htons(SERV_PORT);

 bind (listenfd, (struct sockaddr *) &servaddr, sizeof(servaddr));

 listen (listenfd, LISTENQ);

 printf("%s\n","Server running...waiting for connections.");

 for (; ;) {

 clilen = sizeof(cliaddr);

```

```

connfd = accept (listenfd, (struct sockaddr *) &cliaddr, &clilen);
printf("%s\n","Received request...");

while ((n = recv(connfd, buf, MAXLINE,0)) > 0) {
 printf("%s","String received from and resent to the client:");
 puts(buf);
 send(connfd, buf, n, 0);
}

if (n < 0) {
 perror("Read error");
 exit(1);
}
close(connfd);

}

//close listening socket
close (listenfd);
}

```

**conEchoServer.c source:** [conEchoServer.c](#)

### **TCP Concurrent Echo Server**

```

#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>

#define MAXLINE 4096 /*max text line length*/
#define SERV_PORT 3000 /*port*/
#define LISTENQ 8 /*maximum number of client connections*/

int main (int argc, char **argv)
{
int listenfd, connfd, n;
pid_t childpid;
socklen_t clilen;
char buf[MAXLINE];
struct sockaddr_in cliaddr, servaddr;

//Create a socket for the socket
//If sockfd<0 there was an error in the creation of the socket
if ((listenfd = socket (AF_INET, SOCK_STREAM, 0)) <0) {
 perror("Problem in creating the socket");
 exit(2);
}

```

```

//preparation of the socket address
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(SERV_PORT);

//bind the socket
bind (listenfd, (struct sockaddr *) &servaddr, sizeof(servaddr));

//listen to the socket by creating a connection queue, then wait for clients
listen (listenfd, LISTENQ);

printf("%s\n","Server running...waiting for connections.");

for (; ;) {

 clilen = sizeof(cliaddr);
 //accept a connection
 connfd = accept (listenfd, (struct sockaddr *) &cliaddr, &clilen);

 printf("%s\n","Received request...");

 if ((childpid = fork ()) == 0) {//if it's 0, it's child process

 printf ("%s\n","Child created for dealing with client requests");

 //close listening socket
 close (listenfd);

 while ((n = recv(connfd, buf, MAXLINE,0)) > 0) {
 printf("%s","String received from and resent to the client:");
 puts(buf);
 send(connfd, buf, n, 0);
 }

 if (n < 0)
 printf("%s\n", "Read error");
 exit(0);
 }
 //close socket of the server
 close(connfd);
}
}

```

## **Java RMI**

RMI stands for **Remote Method Invocation**. It is a mechanism that allows an object residing in one system (JVM) to access/invoke an object running on another JVM.

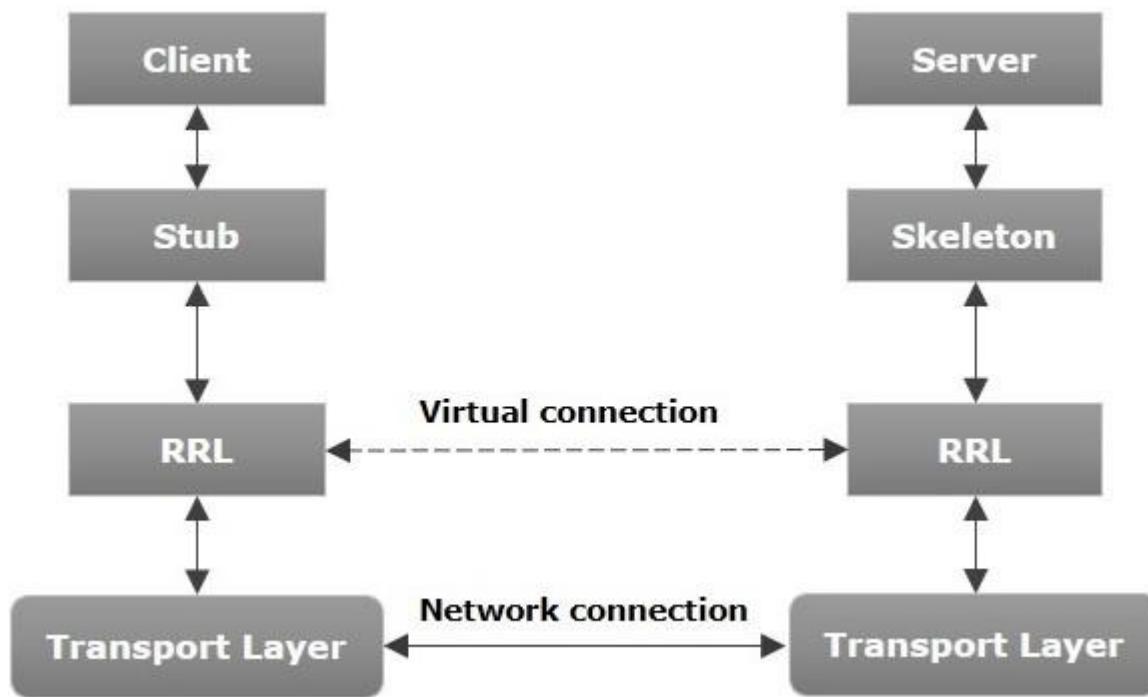
RMI is used to build distributed applications; it provides remote communication between Java programs. It is provided in the package **java.rmi**.

## Architecture of an RMI Application

In an RMI application, we write two programs, a **server program** (resides on the server) and a **client program** (resides on the client).

- Inside the server program, a remote object is created and reference of that object is made available for the client (using the registry).
- The client program requests the remote objects on the server and tries to invoke its methods.

The following diagram shows the architecture of an RMI application.



Let us now discuss the components of this architecture.

- **Transport Layer** – This layer connects the client and the server. It manages the existing connection and also sets up new connections.
- **Stub** – A stub is a representation (proxy) of the remote object at client. It resides in the client system; it acts as a gateway for the client program.
- **Skeleton** – This is the object which resides on the server side. **stub** communicates with this skeleton to pass request to the remote object.
- **RRL(Remote Reference Layer)** – It is the layer which manages the references made by the client to the remote object.

## Working of an RMI Application

The following points summarize how an RMI application works –

- When the client makes a call to the remote object, it is received by the stub which eventually passes this request to the RRL.
- When the client-side RRL receives the request, it invokes a method called **invoke()** of the object **remoteRef**. It passes the request to the RRL on the server side.
- The RRL on the server side passes the request to the Skeleton (proxy on the server) which finally invokes the required object on the server.

- The result is passed all the way back to the client.

## Marshalling and Unmarshalling

Whenever a client invokes a method that accepts parameters on a remote object, the parameters are bundled into a message before being sent over the network. These parameters may be of primitive type or objects. In case of primitive type, the parameters are put together and a header is attached to it. In case the parameters are objects, then they are serialized. This process is known as **marshalling**.

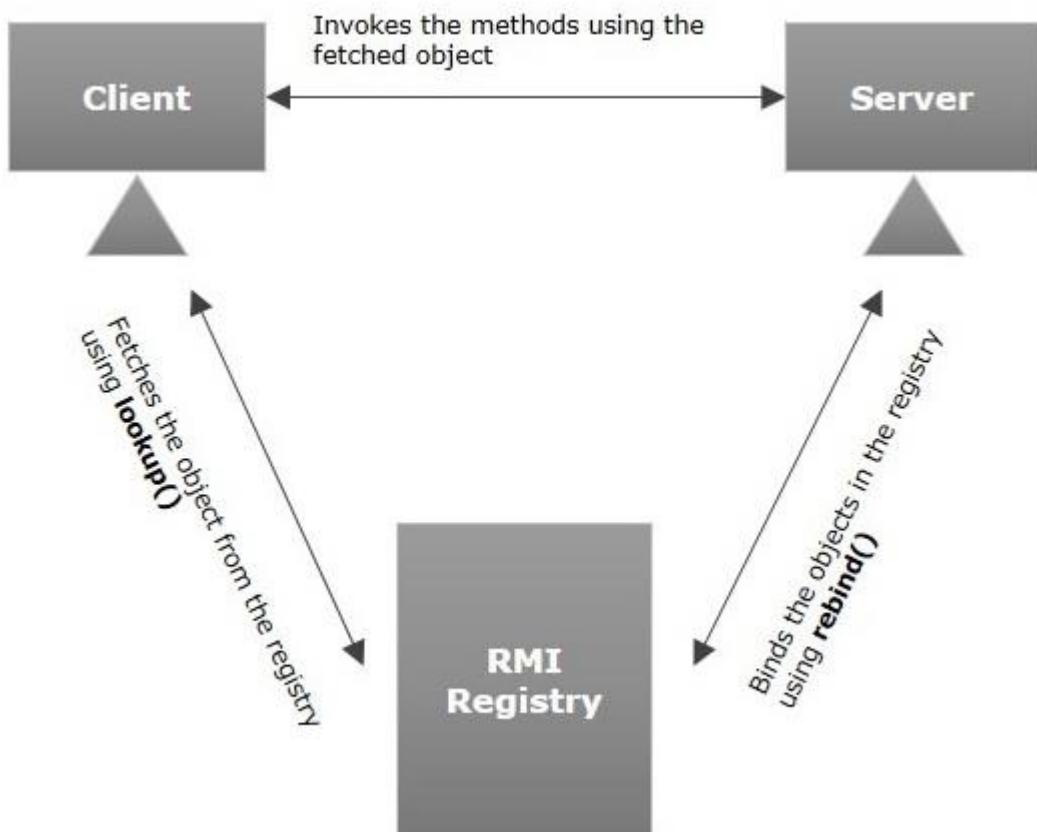
At the server side, the packed parameters are unbundled and then the required method is invoked. This process is known as **unmarshalling**.

## RMI Registry

RMI registry is a namespace on which all server objects are placed. Each time the server creates an object, it registers this object with the RMI registry (using **bind()** or **reBind()** methods). These are registered using a unique name known as **bind name**.

To invoke a remote object, the client needs a reference of that object. At that time, the client fetches the object from the registry using its bind name (using **lookup()** method).

The following illustration explains the entire process –



## Goals of RMI

Following are the goals of RMI –

- To minimize the complexity of the application.
- To preserve type safety.
- Distributed garbage collection.

- Minimize the difference between working with local and remote objects.

**UNIT – IV****APPLETS**

Java applets- Life cycle of an applet – Adding images to an applet – Adding sound to an applet. Passing parameters to an applet. Event Handling. Introducing AWT: Working with Windows Graphics and Text. Using AWT Controls, Layout Managers and Menus. Servlet – life cycle of a servlet. The Servlet API, Handling HTTP Request and Response, using Cookies, Session Tracking Introduction to JSP.

The Servlet technology and JavaServer Pages (JSP) are the two main technologies for developing java Web applications. When first introduced by Sun Microsystems in 1996, the Servlet technology was considered superior to the reigning Common Gateway Interface (CGI) because servlets stay in memory after they service the first requests. Subsequent requests for the same servlet do not require instantiation of the servlet's class therefore enabling better response time.

Servlets are Java classes that implement the javax.servlet.Servlet interface. They are compiled and deployed in the web server. The problem with servlets is that you embed HTML in Java code. If you want to modify the cosmetic look of the page or you want to modify the structure of the page, you have to change code. Generally speaking, this is left to the better hands (and brains) of a web page designer and not to a Java developer.

```
PrintWriter pw = response.getWriter();
pw.println("<html><head><title>Testing</title></head>"); pw.println("<body
bgcolor=\"# ffddcc\">");
```

As seen from the example above this method presents several difficulties to the web developer:

1. The code for a servlet becomes difficult to understand for the programmer.
2. The HTML content of such a page is difficult if not impossible for a web designer to understand or design.
3. This is hard to program and even small changes in the presentation, such as the page's background color, will require the servlet to be recompiled. Any changes in the HTML content require the recompiling of the whole servlet.
4. It's hard to take advantage of web-page development tools when designing the application interface. If such tools are used to develop the web page layout, the generated HTML must then be manually embedded into the servlet code, a process which is time consuming, error prone, and extremely boring.
5. In many Java servlet-based applications, processing the request and generating the response are both handled by a single servlet class.
6. The servlet contains request processing and business logic (implemented by methods), and also generates the response HTML code, are embedded directly in the servlet code.

JSP solves these problems by giving a way to include java code into an HTML page using scriptlets. This way the HTML code remains intact and easily accessible to web designers, but the page can still perform its task.

In late 1999, Sun Microsystems added a new element to the collection of Enterprise Java tools: JavaServer Pages(JSP). JavaServerPages are built on top of Java servlets and

designed to increase the efficiency in which programmers, and even nonprogrammers, can create web content.

Instead of embedding HTML in the code, you place all static HTML in a JSP page, just as in a regular web page, and add a few JSP elements to generate the dynamic parts of the page. The request processing can remain the domain of the servlet, and the business logic can be handled by JavaBeans and EJB components.

A JSP page is handled differently compared to a servlet by the web server. When a servlet is deployed into a web server in compiled (bytecode) form, then a JSP page is deployed in its original, human-readable form.

When a user requests the specific page, the web server compiles the page into a servlet and from there on handles it as a standard servlet.

This accounts for a small delay, when a JSP page is first requested, but any subsequent requests benefit from the same speed effects that are associated with servlets.

### **The Problem with Servlet**

- Servlets are difficult to code which are overcome in JSP. Other way, we can say, JSP is almost a replacement of Servlets, (by large, the better word is extension of Servlets), where coding decreases more than half.
- In Servlets, both static code and dynamic code are put together. In JSP, they are separated. For example, In Servlets:

```
out.println("Hello Mr." + str + " you are great man");
```

where str is the name of the client which changes for each client and is known as dynamic content. The strings, "Hello Mr." and "you are great man" are static content which is the same irrespective of client. In Servlets, in `println()`, both are put together.

- In JSP, the static content and dynamic content is separated. Static content is written in HTML and dynamic content in JSP. As much of the response comprises of static content (nearly 70%) only, the JSP file more looks as a HTML file.
- Programmer inserts, here and there, chunks of JSP code in a running HTML developed by Designer. As much of the response delivered to client by server comprises of static content (nearly 70%), the JSP file more looks like a HTML file. Other way we can say, JSP is nothing but Java in HTML (servlets are HTML)
- in Java); java code embedded in HTML.
- When the roles of Designer and Programmer are nicely separated, the product development becomes cleaner and fast. Cost of developing Web site becomes cheaper as Designers are much paid less than Programmers, especially should be thought in the present competitive world.
- Both presentation layer and business logic layer put together in Servlets. In JSP, they can be separated with the usage of JavaBeans.
- The objects of PrintWriter, ServletConfig, ServletContext, HttpSession and RequestDispatcher etc. are created by the Programmer in Servlets and used. But in JSP, they are built-in and are known as "implicit objects". That is, in JSP, Programmer never creates these objects and straightforwardly uses them as they are implicitly created and given by JSP container. This decreases lot of coding.
- JSP can easily be integrated with JavaBeans.
- JSP is much used in frameworks like Struts.
- With JSP, Programmer can build custom tags that can be called in JavaBeans directly. Servlets do not have this advantage. Reusability increases with tag libraries and JavaBeans etc.
- Writing alias name in <url-pattern> tag of web.xml is optional in JSP but mandatory in Servlets.
- A Servlet is simply a Java class with extension .java written in normal Java code.

- A Servlet is a Java class. It is written like a normal Java. JSP is comes with some elements that are easy to write.
- JSP needs no compilation by the Programmer. Programmer deploys directly a JSP source code file in server where as incase of Servlets, the Programmer compiles manually a Servlet file and deploys a .class file in server.
- JSP is so easy even a Web Designer can put small interactive code (not knowing much of Java) in static Webpages.
- First time when JSP is called it is compiled to a Servlet. Subsequent calls to the same JSP will call the same compiled servlet (instead of converting the JSP to servlet), Ofcourse, the JSP code would have not modified. This increases performance.

### Anatomy of JSP

## Anatomy of a jsp page

```
<%@page contentType = "text/html" language = "java%">
<%@page import = "java.util.Date" session = "false%">}
```

Jsp elements

**%@ is jsp directive**

```
<html>
<head>
<title> simple jsp page demo</title>
</head>
<body>
<h3> current time is : </h3>
```



Template data

**<%= new Date()%> -- > jsp elements**

**%= is jsp element**

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

Template data

### JSP Processing

Once you have a JSP capable web-server or application server, you need to know the following information about it:

- Where to place the files
- How to access the files from your browser (with an http: prefix, not asfile:)

You should be able to create a simple file, such as

```
<HTML>
<BODY>
Hello, world
</BODY></HTML>
```

Know where to place this file and how to see it in your browser with an http:// prefix.

Since this step is different for each web-server, you would need to see the web-server documentation to find out how this is done. Once you have completed this step, proceed to the next.

### **Your first JSP**

JSP simply puts Java inside HTML pages. You can take any existing HTML page and change its extension to ".jsp" instead of ".html". In fact, this is the perfect exercise for your first JSP.

Take the HTML file you used in the previous exercise. Change its extension from ".html" to ".jsp". Now load the new file, with the ".jsp" extension, in your browser.

**You will see the same output, but it will take longer! But only the first time. If you reload it again, it will load normally.**

What is happening behind the scenes is that your JSP is being turned into a Java file, compiled and loaded. This compilation only happens once, so after the first load, the file doesn't take long to load anymore. (But everytime you change the JSP file, it will be recompiled again.)

Of course, it is not very useful to just write HTML pages with a .jsp extension! We now proceed to see what makes JSP souseful

Adding dynamic content viaexpressions

As we saw in the previous section, any HTML file can be turned into a JSP file by changing its extension to .jsp. Of course, what makes JSP useful is the ability to embed Java. Put the following text in a file with .jsp extension (let us call it hello.jsp), place it in your JSP directory, and view it in a browser.

```
<HTML>
<BODY>
Hello! The time is now <%= new java.util.Date() %>
</BODY>
</HTML>
```

Notice that each time you reload the page in the browser, it comes up with the current time. The character sequences

<%= and %> enclose Java expressions, which are evaluated at run time.

This is what makes it possible to use JSP to generate dyamic HTML pages that change in response to user actions or vary from user to user.

### **Explain about JSP Elements**

In this lesson we will learn about the various elements available in JSP with suitable examples. InJSP elements can be divided into 4 different types.

**Theseare:**

#### **1. Expressions**

We can use this tag to output any data on the generated page. These data are automatically converted to string and printed on the outputstream.

Syntax of JSP Expressions are: <%="Any thing"%>

JSP Expressions start with Syntax of JSP Scriptles are with `<%=` and ends with `%>`. Between these this you can put anything and that will convert to the String and that will be displayed.

**Example:** `<%"Hello World!" %>` Above code will display 'HelloWorld!'

## 2. Scriptlets

In this tag we can insert any amount of valid java code and these codes are placed in `_jspService` method by the JSP engine.

### Syntax of JSP Scriptlets are:

`<% //java codes`

`%>`

JSP Scriptlets begins with `<%` and ends `%>`. We can embed any amount of java code in the JSP Scriptlets. JSP Engine places these code in the `_jspService()` method. Variables available to the JSP Scriptlets are:

**a. Request:** Request represents the clients request and is a subclass of `HttpServletRequest`. Use this variable to retrieve the data submitted along therequest.

Example: `<% //javacodes`

`String userName=null; userName=request.getParameter("userName");`

`%>`

**b. Response:** Response represents the server response and is a subclass of `HttpServletResponse`.

`<% response.setContentType("text/html");%>`

**c. Session:** represents the HTTP sessionobject associated with the request. Your Session ID: `<%= session.getId()%>`

**d. Out:** out is an object of output stream and is used to send any output to theclient.

## 3. Directives

A JSP "directive" starts with `<%@` characters. In the directives we can import packages, define error handling pages or the session information of the JSP page.

### Syntax of JSP directives is:

`<%@directive attribute="value" %>`

**a. page:** page is used to provide the information about it. Example: `<%@page language="java"%>`

**b. include:** include is used to include a file in the JSP page. Example: `<%@ include file="/header.jsp"%>`

**c. taglib:** taglib is used to use the custom tags in the JSP pages (custom tags allows us to defined our own tags). Example: `<%@ taglib uri="tlds/taglib.tld" prefix="mytag"%>`

Page tag attributes are:

**a. language="java"**

This tells the server that the page is using the java language. Current JSP specification supports only java language. Example: `<%@page language="java"%>`

**b. extends="mypackage.myclass"**

This attribute is used when we want to extend any class. We can use comma(,) to import more than one packages. Example: `%@page language="java" import="java.sql.*"%`

#### c. **session="true"**

When this value is true session data is available to the JSP page otherwise not. By default this value is true.

Example: `<%@page language="java" session="true" %>`

#### d. **errorPage="error.jsp"**

errorPage is used to handle the un-handled exceptions in the page. Example: `<%@page session="true" errorPage="error.jsp"%>`

#### e. **contentType="text/html;charset=ISO-8859-1"**

Use this attribute to set the mime type and character set of the JSP. Example: `<%@page contentType="text/html;charset=ISO-8859-1"%>`

### **4. Declarations**

This tag is used for defining the functions and variables to be used in the JSP. Syntax of JSP Declaratives are:

```
<%!
//java codes
%>
```

JSP Declaratives begins with `<%!` and ends `%>` with . We can embed any amount of java code in the JSP Declaratives. Variables and functions defined in the declaratives are class level and can be used anywhere in the JSP page.

#### **Example**

```
<%@ page import="java.util.*" %>
<HTML>
<BODY>
<%!
Date theDate = new Date(); Date getDate()
{
System.out.println("In getDate() method"); return theDate;
}
%>
Hello! The time is now <%= getDate() %>
</BODY>
</HTML>
```

### **Explaining about Jsp programs?**

A Web Page with JSP code

```
<HTML>
<HEAD>
<TITLE>A Web Page</TITLE>
</HEAD>
<BODY>
```

```
<% out.println("Hello there!"); %>
</BODY>
</HTML>
```

### **Using a Literal**

```
<HTML>
<HEAD>
<TITLE>Using a Literal</TITLE>
</HEAD>
<BODY>
<H1>Using a Literal</H1>
<%
out.println("Number of days = "); out.println(365);
%>
</BODY>
</html>
```

### **Declaration Tag Example**

```
<%!
String name = "Joe";
String date = "8th April, 2002";
%>
<HTML>
<TITLE>Declaration Tag Example</TITLE>
<BODY>
This page was last modified on <%= date %> by <%= name %>.
</BODY>
</HTML>
```

### **Embedding Code**

```
<%!
String[] names = {"A", "B", "C", "D"};
%>
<HTML>
<HEAD><TITLE>Embedding Code</TITLE></HEAD>
<BODY>
<H1>List of people</H1>
<TABLE BORDER="1">
<TH>Name</TH>
<% for (int i=0; i<names.length; i++) { %>
<TR><TD><%=names[i]%></TD></TR>
```

```
<% } %>
</TABLE>
</BODY>
</HTML>
```

### **Use out**

```
<%@ page language="java" %>
<HTML>
<HEAD><TITLE>JSP Example</TITLE></HEAD>
<BODY>
<H1>Quadratic Equation: y = x^2</H1>
<TABLE BORDER="1">
<TH>x</TH><TH>y</TH>
<%
for (int i=0; i<10; i++)
out.print("<TR><TD WIDTH='100'>" + i + "</TD><TD WIDTH='100'>" + (i*i) +
"</TD></TR>");
%>
</TABLE>
</BODY>
</HTML>
Casting to a New Type
<HTML>
<HEAD>
<TITLE>Casting to a New Type</TITLE>
</HEAD>
<BODY>
<H1>Casting to a New Type</H1>
<%
float float1;
double double1 = 1;
float1 = (float) double1;

out.println("float1 = " + float1);
%>
</BODY>
</HTML>
```

### **Creating a String**

```
<HTML>
<HEAD>
```

```
<TITLE>Creating a String</TITLE>
</HEAD>
```

```
<BODY>
<H1>Creating a String</H1>
<%
String greeting = "Hello from JSP!";
out.println(greeting);
%>
</BODY>
</HTML>
```

### **Use for loop to display string array**

```
<%@ page session="false" %>
<%
String[] colors = { "red", "green", "blue" };
for (int i = 0; i < colors.length; i++) { out.print("<P>" + colors[i] + "</p>");
}
%>
```

### **Creating an Array**

```
<HTML>
<HEAD>
<TITLE>Creating an Array</TITLE>
</HEAD>
<BODY>
<H1>Creating an Array</H1>
<%
double accounts[];
accounts = new double[100]; accounts[3] = 119.63;
out.println("Account 3 holds $" + accounts[3]);
%>
</BODY>
</HTML>
```

### **Using Multidimensional Arrays**

```
<HTML>
<HEAD>
<TITLE>Using Multidimensional Arrays</TITLE>
</HEAD>
<BODY>
<H1>Using Multidimensional Arrays</H1>
```

```

<%
double accounts[][] = new double[2][100];
accounts[0][3] = 119.63;
accounts[1][3] = 194.07;
out.println("Savings Account 3 holds $" + accounts[0][3] + "
"); out.println("Checking
Account 3 holds $" + accounts[1][3]);
%>
</BODY>
</HTML>

```

### **Finding a Factorial**

```

<HTML>
<HEAD>
<TITLE>Finding a Factorial</TITLE>
</HEAD>
<BODY>
<H1>Finding a Factorial</H1>
<%
int value = 6, factorial = 1, temporaryValue = value;
while (temporaryValue > 0) { factorial *= temporaryValue; temporaryValue--;
}
out.println("The factorial of " + value + " is " + factorial + ".");
%>
</BODY>
</HTML>

```

### **Get Form Button Value**

```

<HTML>
<HEAD>
<TITLE>Using Buttons</TITLE>
</HEAD>
<BODY>
<H1>Using Buttons</H1>
<FORM NAME="form1" ACTION="basic.jsp" METHOD="POST">
<INPUT TYPE="HIDDEN" NAME="buttonName">
<INPUT TYPE="BUTTON" VALUE="Button 1" ONCLICK="button1()">
<INPUT TYPE="BUTTON" VALUE="Button 2" ONCLICK="button2()">
<INPUT TYPE="BUTTON" VALUE="Button 3" ONCLICK="button3()">
</FORM>
<SCRIPT LANGUAGE="JavaScript">
<!--

```

```

function button1()
{
document.form1.buttonName.value = "button 1" form1.submit()
}
function button2()
{
document.form1.buttonName.value = "button 2" form1.submit()
}
function button3()
{
document.form1.buttonName.value = "button 3" form1.submit()
}
// -->
</SCRIPT>
</BODY>
</HTML>

```

**basic.jsp**

```

<HTML>
<HEAD>
<TITLE>Determining Which Button Was Clicked</TITLE>
</HEAD>
<BODY>
<H1>Determining Which Button Was Clicked</H1> You clicked
<%= request.getParameter("buttonName") %>
</BODY>
</HTML>

```

**Read Form Checkboxes****index.jsp**

```

<HTML>
<HEAD>
<TITLE>Submitting Check Boxes</TITLE>
</HEAD>
<BODY>
<H1>Submitting Check Boxes</H1>
<FORM ACTION="basic.jsp" METHOD="post">
<INPUT TYPE="CHECKBOX" NAME="check1" VALUE="check1" CHECKED>
Checkbox 1

<INPUT TYPE="CHECKBOX" NAME="check2" VALUE="check2">

```

```

Checkbox 2

<INPUT TYPE="CHECKBOX" NAME="check3" VALUE="check3">
Checkbox 3

<INPUT TYPE="SUBMIT" VALUE="Submit">
</FORM>
</BODY>
</HTML>
```

**basic.jsp**

```

<HTML>
<HEAD>
<TITLE>Reading Checkboxes</TITLE>
</HEAD>
<BODY>
<H1>Reading Checkboxes</H1>
<%
if(request.getParameter("check1") != null) { out.println("Checkbox 1 was checked.
"); }
else {
out.println("Checkbox 1 was not checked.
"); }
if(request.getParameter("check2") != null) { out.println("Checkbox 2 was checked.
"); }
else {
out.println("Checkbox 2 was not checked.
"); }
if(request.getParameter("check3") != null) { out.println("Checkbox 3 was checked.
"); }
else {
out.println("Checkbox 3 was not checked.
"); }
%>
</BODY>
</HTML>
```

**Model View Controller**

JSP technology can play a part in everything from the simplest web application to complex enterprise applications. How large a part JSP plays differs in each case, of course. Let's introduce a design model called Model-View-Controller (MVC), suitable for both simple and complex applications.

MVC was first described by Xerox in a number of papers published in the late 1980s. The key point of using MVC is to separate logic into three distinct units: the Model, the View, and the Controller. In a server application, we commonly classify the parts of the application as business logic, presentation, and requestprocessing.

Business logic is the term used for the manipulation of an application's data, such as customer, product, and order information. Presentation refers to how the application data is displayed to the user, for example, position, font, and size. And finally, request processing is what ties the business logic and presentation partstogether.

In MVC terms, presentation should be separated from the business logic. Presentation of that data (the View) changes fairly often. Just look at all the face-lifts many web sites go through to keep up with the latest fashion in web design. Some sites may want to present the data in different languages or present different subsets of the data to internal and external users.

### **cookies:**

- A **cookie** is a small piece of information created by a JSP program that is stored in theclient's hard disk by the browser. Cookies are used to store various kind of information such as username, password, and user preferences,etc.
- **Different methods in cookie class are:**
  1. **String getName()**- Returns a name ofcookie
  2. **String getValue()**-Returns a value ofcookie
  3. **int getMaxAge()**-Returns a maximum age of cookie inmillisecond
  4. **String getDomain()**-Returns adomain
  5. **boolean getSecure()**-Returns true if cookie is secure otherwisefalse
  6. **String getPath()**-Returns a path ofcookie
  7. **void setPath(Sting)**- set the path ofcookie
  8. **void setDomain(String)**-set the domain of cookie
  9. **void setMaxAge(int)**-set the maximum age of cookie
  10. **void setSecure(Boolean)**-set the secure ofcookie.

### **Creating cookie:**

Cookie are created using cookie class constructor.

Content of cookies are added the browser using addCookies() method.

### **Reading cookies:**

Reading the cookie information from the browser using getCookies() method.

Find the length of cookie class.

Retrive the information using different method belongs the cookie class

**PROGRAM: To create and read the cookie for the given cookie name as “EMPID” and its value as”AN2356”.**

### **JSP program to create a cookie**

```
<%!
Cookie c=new Cookie("EMPID","AN2356");
response.addCookie(c);
%>
```

**JSP program to read a cookie**

```
<%!
Cookie c[] = request.getCookies();
for(i=0;i<c.length;i++)
{
String name=c[i].getName();
String value=c[i].getValue();
out.println("name=" + name);
out.println("value=" + value);
}
%>
```

**Session object(session tracking or session uses)**

- The HttpSession object associated to therequest
- Session object has a session scope that is an instance of javax.servlet.http.HttpSession class. Perhaps it is the most commonly used object to manage the statecontexts.
- This object persist information across multiple userconnection.
- Created automaticallyby
- Different methods of HttpSession interface are asfollows:

1. **object getAttribute(String)**-Returns the value associated with the name passed as argument.
2. **long getCreationTime()**-Returns the time when session created.
3. **String getID()**-Returns the sessionID
4. **long getLastAccessedTime()**-returns the time when client last made a request for this session.
5. **void setAttribute(String,object)**-Associates the values passed in the object namepassed.

**Program:**

```
<%!
HttpSession h=req.getSession(true);
Date d=(Date) h.getAttribute("Date");
out.println("last date and time=" + d);
Date d1=new Date();
d1=h.setAttribute("date",d1);
out.println("current date and time=" + d1);
%>
```

## JavaBeans

### **JavaBeans:**

JavaBeans is architecture for both using and building components in Java. This architecture supports the features of software reuse, component models, and object orientation. One of the most important features of JavaBeans is that it does not alter the existing Java language.

Although Beans are intended to work in a visual application development tool, they don't necessarily have a visual representation at run-time (although many will). What this does mean is that Beans must allow their property values to be changed through some type of visual interface, and their methods and events should be exposed so that the development tool can write code capable of manipulating the component when the application is executed.

**Bean Development Kit (BDK)** is a tool for testing whether your JavaBeans meets the JavaBean specification.

### **Features of JavaBeans**

**Compact and Easy:** JavaBeans components are simple to create and easy to use. This is an important goal of the JavaBeans architecture. It doesn't take very much to write a simple Bean, and such a Bean is lightweight, it doesn't have to carry around a lot of inherited baggage just to support the Beans environment.

**Portable:** Since JavaBeans components are built purely in Java, they are fully portable to any platform that supports the Java run-time environment. All platform specifics, as well as support for JavaBeans, are implemented by the Java virtual machine.

**Introspection:** Introspection is the process of exposing the properties, methods, and events that a JavaBean component supports. This process is used at run-time, as well as by a visual development tool at design-time. The default behavior of this process allows for the automatic introspection of any Bean. A low-level reflection mechanism is used to analyze the Bean's class to determine its methods. Next it applies some simple design patterns to determine the properties and events that are supported. To take advantage of reflection, you only need to follow a coding style that matches the design pattern. This is an important feature of JavaBeans. It means that you don't have to do anything more than code your methods using a simple convention. If you do, your Beans will automatically support introspection without you having to write any extracode.

**Customization:** When you are using a visual development tool to assemble components into applications, you will be presented with some sort of user interface for customizing Bean attributes. These attributes may affect the way the Bean operates or the way it looks on the screen. The application tool you use will be able to determine the properties that a Bean supports and build a property sheet dynamically. This property sheet will contain editors for each of the properties supported by the Bean, which you can use to customize the Bean to your liking. The Beans class library comes with a number of property editors for common types such as float, boolean, and String. If you are using custom classes for properties, you will have to create custom property editors to associate with them.

**Persistence:** It is necessary that Beans support a large variety of storage mechanisms. This way, Beans can participate in the largest number of applications. The simplest way to support persistence is to take advantage of Java Object Serialization. This is an automatic mechanism for saving and restoring the state of an object. Java Object Serialization is the best way to make sure that your Beans are fully portable, because you take advantage of a standard feature supported by the core Java platform. This, however, is not always desirable. There

may be cases where you want your Bean to use other file formats or mechanisms to save and restore state. In the future, JavaBeans will support an alternative externalization mechanism that will allow the Bean to have complete control of its persistence mechanism.

### Deploying java beans in jsp:

A JavaBean can be defined as a reusable software component. A Java Bean is a java class that should follow following conventions:

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

Java beans- development phases

1. The ConstructionPhase
2. The BuildPhase
3. The ExecutionPhase

JavaBeans component design conventions govern the properties of the class, and the public methods that give access to the properties.

### A JavaBeans component property can be:

Read/write, read-only, or write-only.

It means it contains a single value, or indexed, i.e. it represents an array of values.

There is no requirement that a property be implemented by an instance variable; the property must simply be accessible using public methods that conform to certain conventions:

For each readable property, the bean must have a method of the form:

PropertyClass getProperty () { ... }

For each writable property, the bean must have a method of the form:

setProperty (PropertyClass pc) { ... }

In addition to the property methods, a JavaBeans component must define a constructor that takes no parameters.

### Steps to deploy and run this JSP using JavaBeanProject

1. Write a java file and name it as **FindAuthor.java**
2. Write a jsp file and name it as **GetAuthorName.jsp**
3. Write a html file and name it as **WelcomePage.html**

**UNIT – IV****APPLETS**

Java applets- Life cycle of an applet – Adding images to an applet – Adding sound to an applet. Passing parameters to an applet. Event Handling. Introducing AWT: Working with Windows Graphics and Text. Using AWT Controls, Layout Managers and Menus. Servlet – life cycle of a servlet. The Servlet API, Handling HTTP Request and Response, using Cookies, Session Tracking Introduction to JSP.

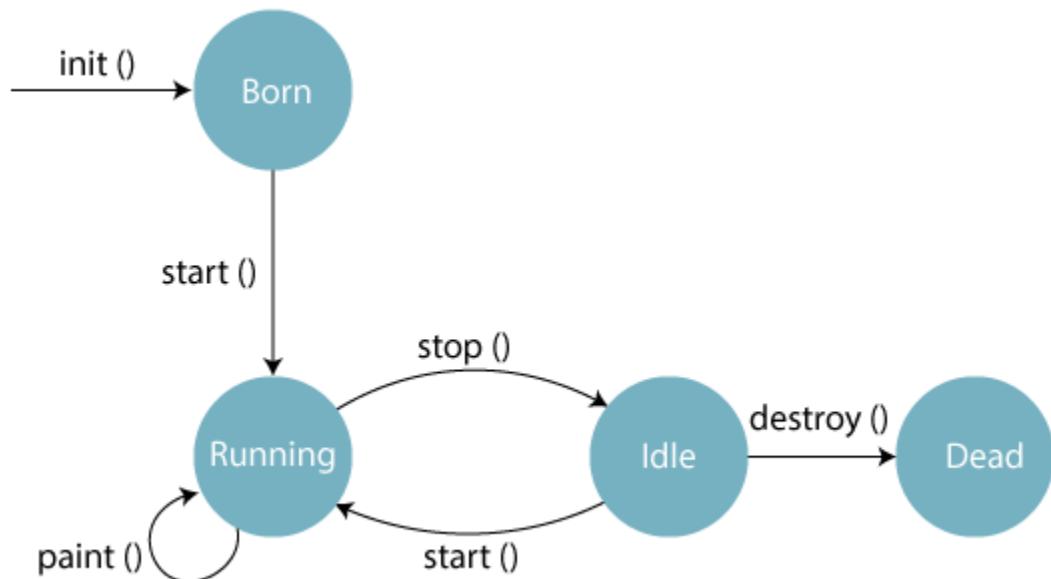
**Applet Life Cycle in Java**

In Java, an applet is a special type of program embedded in the web page to generate dynamic content. Applet is a class in Java.

The applet life cycle can be defined as the process of how the object is created, started, stopped, and destroyed during the entire execution of its application. It basically has five core methods namely init(), start(), stop(), paint() and destroy(). These methods are invoked by the browser to execute.

Along with the browser, the applet also works on the client side, thus having less processing time.

Methods of Applet Life Cycle



There are five methods of an applet life cycle, and they are:

- **init():** The init() method is the first method to run that initializes the applet. It can be invoked only once at the time of initialization. The web browser creates the initialized

objects, i.e., the web browser (after checking the security settings) runs the init() method within the applet.

- **start():** The start() method contains the actual code of the applet and starts the applet. It is invoked immediately after the init() method is invoked. Every time the browser is loaded or refreshed, the start() method is invoked. It is also invoked whenever the applet is maximized, restored, or moving from one tab to another in the browser. It is in an inactive state until the init() method is invoked.
- **stop():** The stop() method stops the execution of the applet. The stop () method is invoked whenever the applet is stopped, minimized, or moving from one tab to another in the browser, the stop() method is invoked. When we go back to that page, the start() method is invoked again.
- **destroy():** The destroy() method destroys the applet after its work is done. It is invoked when the applet window is closed or when the tab containing the webpage is closed. It removes the applet object from memory and is executed only once. We cannot start the applet once it is destroyed.
- **paint():** The paint() method belongs to the Graphics class in Java. It is used to draw shapes like circle, square, trapezium, etc., in the applet. It is executed after the start() method and when the browser or applet windows are resized.

### **Sequence of method execution when an applet is executed:**

1. init()
2. start()
3. paint()

### **Sequence of method execution when an applet is executed:**

1. stop()
2. destroy()

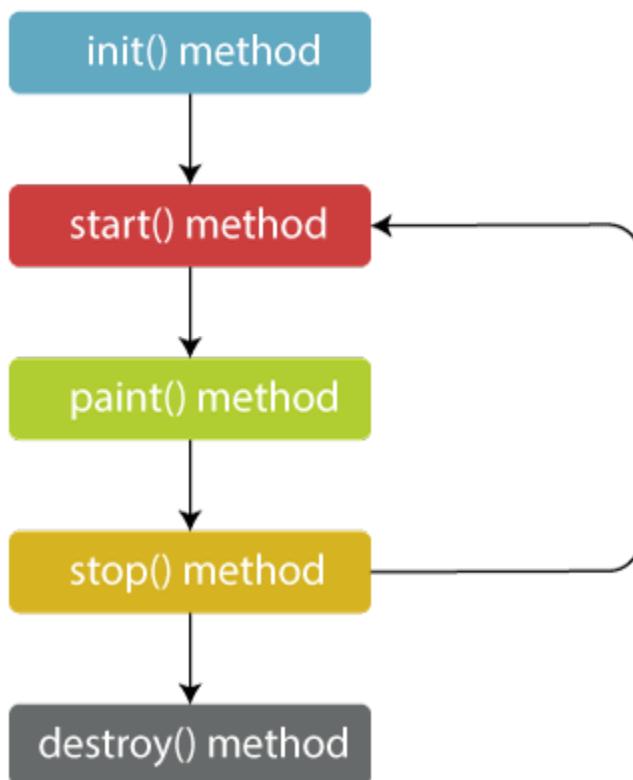
#### **Applet Life Cycle Working**

- The Java plug-in software is responsible for managing the life cycle of an applet.
- An applet is a Java application executed in any web browser and works on the client-side. It doesn't have the main() method because it runs in the browser. It is thus created to be placed on an HTML page.

- The `init()`, `start()`, `stop()` and `destroy()` methods belongs to the **applet.Applet** class.
- The `paint()` method belongs to the **awt.Component** class.
- In Java, if we want to make a class an Applet class, we need to extend the **Applet**
- Whenever we create an applet, we are creating the instance of the existing Applet class. And thus, we can use all the methods of that class.

Flow of Applet Life Cycle:

These methods are invoked by the browser automatically. There is no need to call them explicitly.



Syntax of entire Applet Life Cycle in Java

```

1. class TestAppletLifeCycle extends Applet {
2. public void init() {
3. // initialized objects
4. }
5. public void start() {
6. // code to start the applet
7. }

```

```

8. public void paint(Graphics graphics) {
9. // draw the shapes
10. }
11. public void stop() {
12. // code to stop the applet
13. }
14. public void destroy() {
15. // code to destroy the applet
16. }
17. }
```

### **Adding images to an applet**

#### **Displaying Image in Applet**

Applet is mostly used in games and animation. For this purpose image is required to be displayed. The java.awt.Graphics class provide a method drawImage() to display the image.

Syntax of drawImage() method:

1. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.

How to get the object of Image:

The java.applet.Applet class provides getImage() method that returns the object of Image. Syntax:

1. **public Image getImage(URL u, String image){}**

Other required methods of Applet class to display image:

1. **public URL getDocumentBase():** is used to return the URL of the document in which applet embedded.
2. **public URL getCodeBase():** is used to return the base URL.

Example of displaying image in applet:

1. **import** java.awt.\*;
2. **import** java.applet.\*;
- 3.
- 4.

```

5. public class DisplayImage extends Applet {
6.
7. Image picture;
8.
9. public void init() {
10. picture = getImage(getDocumentBase(),"sonoo.jpg");
11. }
12.
13. public void paint(Graphics g) {
14. g.drawImage(picture, 30,30, this);
15. }
16.
17. }
```

In the above example, `drawImage()` method of `Graphics` class is used to display the image. The 4th argument of `drawImage()` method is `ImageObserver` object. The `Component` class implements `ImageObserver` interface. So the current class object would also be treated as `ImageObserver` because `Applet` class indirectly extends the `Component` class.

*myapplet.html*

1. <html>
2. <body>
3. <applet code="DisplayImage.class" width="300" height="300">
4. </applet>
5. </body>
6. </html>

### **Adding sound to an applet**

A Java applet is a special kind of Java program that a browser enabled with Java technology can download from the internet and run. An applet is typically embedded inside a web page and runs in the context of a browser. An applet must be a subclass of the [java.applet.Applet](#) class. The [Applet](#) class provides the standard interface between the applet and the browser environment. To play audio in an Applet one should perform the following steps:

- Create a class that extends the [Applet](#), such as [PlayAudioInApplet](#) class in the example.

- Use `init()` API method of Applet. This method is called by the browser or applet viewer to inform this applet that it has been loaded into the system. In this method call the `getAudioClip(URL url)` API method to get the `AudioClip` object specified by URL and name arguments.
- In `paint(Graphics g)` method call `play()` API method of `AudioClip` to start playing this audio clip. Call `stop()` API method of `AudioClip` to stop playing this audio clip. Call `loop()` API method of `AudioClip` to start playing this audio clip in a loop, as described in the code snippet below.

```
package com.javacodegeeks.snippets.core;
```

```
import java.applet.Applet;
import java.applet.AudioClip;
import java.awt.Graphics;
```

```
public class PlayAudioInApplet extends Applet {
```

```
 private static final long serialVersionUID = 2530894095587089544L;
```

```
 private AudioClip clip;
```

```
 // Called by the browser or applet viewer to inform
 // this applet that it has been loaded into the system.
 public void init() {
```

```
 clip = getAudioClip(getDocumentBase(), "http://www.myserver.com/clip.au");
```

```
}
```

```
 // Paints the container. This forwards the paint to any
```

```
// lightweight components that are children of this container.
public void paint(Graphics g) {

 // Start playing this audio clip. Each time this method is called,
 // the clip is restarted from the beginning.
 clip.play();

 // Stops playing this audio clip.
 clip.stop();

 // Starts playing this audio clip in a loop.
 clip.loop();

}
}
```

This was an example of how to play audio in Applet in Java

## Passing parameters to an applet

### Parameter in Applet

We can get any information from the HTML file as a parameter. For this purpose, Applet class provides a method named `getParameter()`. Syntax:

```
public String getParameter(String parameterName)
Example of using parameter in Applet:
```

```
import java.applet.Applet;
import java.awt.Graphics;
```

```
public class UseParam extends Applet{
 public void paint(Graphics g){
 String str=getParameter("msg");
 g.drawString(str,50, 50);
 }

}
myapplet.html
<html>
<body>
<applet code="UseParam.class" width="300" height="300">
<param name="msg" value="Welcome to applet">
</applet>
</body>
</html>
```

Java AWT

**Java AWT** (Abstract Window Toolkit) is *an API to develop Graphical User Interface (GUI) or windows-based applications* in Java.

Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavy weight i.e. its components are using the resources of underlying operating system (OS).

The `java.awt` package provides classes for AWT API such as `TextField`, `Label`, `TextArea`, `RadioButton`, `CheckBox`, `Choice`, `List` etc.

The AWT tutorial will help the user to understand Java GUI programming in simple and easy steps.

### **Why AWT is platform independent?**

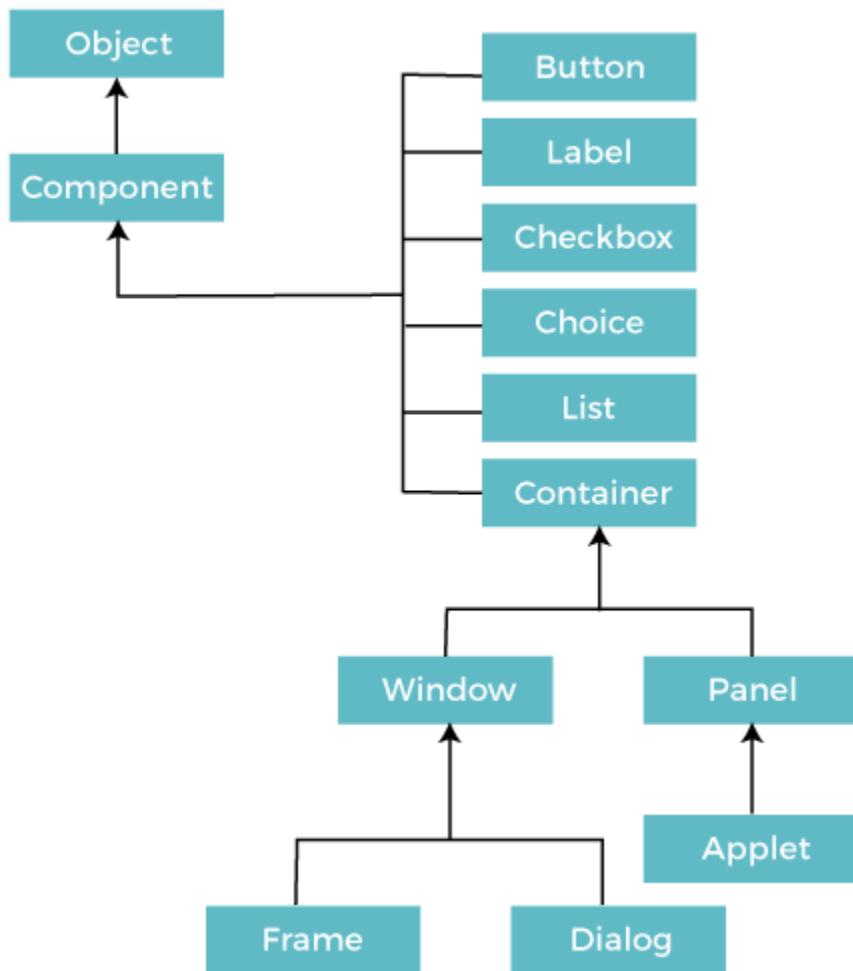
Java AWT calls the native platform calls the native platform (operating systems) subroutine for creating API components like `TextField`, `ChechBox`, button, etc.

For example, an AWT GUI with components like `TextField`, label and button will have different look and feel for the different platforms like Windows, MAC OS, and Unix. The reason for this is the platforms have different view for their native components and AWT directly calls the native subroutine that creates those components.

In simple words, an AWT application will look like a windows application in Windows OS whereas it will look like a Mac application in the MAC OS.

### Java AWT Hierarchy

The hierarchy of Java AWT classes are given below.



### *Components*

All the elements like the button, text fields, scroll bars, etc. are called components. In Java AWT, there are classes for each component as shown in above diagram. In order to place every component in a particular position on a screen, we need to add them to a container.

### *Container*

The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as **Frame**, **Dialog** and **Panel**.

It is basically a screen where the components are placed at their specific locations. Thus it contains and controls the layout of components.

*Note: A container itself is a component (see the above diagram), therefore we can add a container inside container.*

### Types of containers:

There are four types of containers in Java AWT:

1. Window
2. Panel
3. Frame
4. Dialog

#### *Window*

The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window. We need to create an instance of Window class to create this container.

#### *Panel*

The Panel is the container that doesn't contain title bar, border or menu bar. It is generic container for holding the components. It can have other components like button, text field etc. An instance of Panel class creates a container, in which we can add components.

#### *Frame*

The Frame is the container that contain title bar and border and can have menu bars. It can have other components like button, text field, scrollbar etc. Frame is most widely used container while developing an AWT application.

#### Useful Methods of Component Class

Method	Description
public void add(Component c)	Inserts a component on this component.
public void setSize(int width,int height)	Sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	Defines the layout manager for the component.
public void setVisible(boolean status)	Changes the visibility of the component, by default false.

## Java AWT Example

To create simple AWT example, you need a frame. There are two ways to create a GUI using Frame in AWT.

1. By extending Frame class (**inheritance**)
2. By creating the object of Frame class (**association**)

### AWT Example by Inheritance

Let's see a simple example of AWT where we are inheriting Frame class. Here, we are showing Button component on the Frame.

#### **AWTExample1.java**

```
// importing Java AWT class
import java.awt.*;

// extending Frame class to our class AWTEmployee
public class AWTEmployee extends Frame {

 // initializing using constructor
 AWTEmployee() {

 // creating a button
 Button b = new Button("Click Me!!");

 // setting button position on screen
 b.setBounds(30,100,80,30);

 // adding button into frame
 add(b);

 // frame size 300 width and 300 height
 setSize(300,300);

 // setting the title of Frame
 setTitle("Java AWT Example");
 }

 // overriding the paint method
 public void paint(Graphics g) {
 super.paint(g);
 g.drawString("Hello World", 100, 100);
 }
}
```

```
setTitle("This is our basic AWT example");

// no layout manager
setLayout(null);

// now frame will be visible, by default it is not visible
setVisible(true);
}

// main method
public static void main(String args[]) {

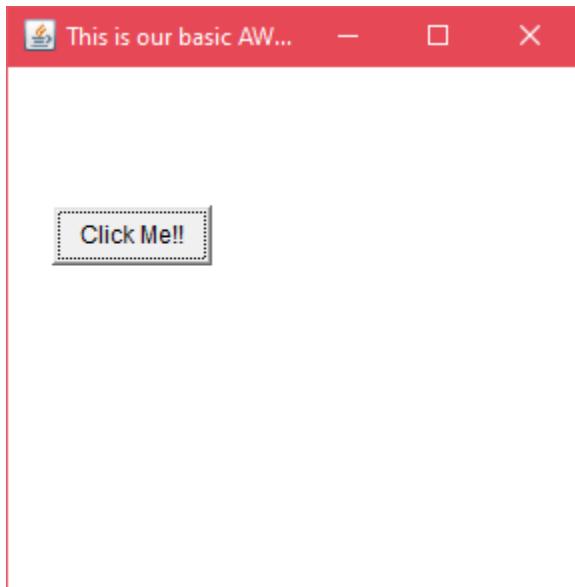
// creating instance of Frame class
AWTExample1 f = new AWTExample1();

}

download this example
```

The `setBounds(int x-axis, int y-axis, int width, int height)` method is used in the above example that sets the position of the awt button.

#### **Output:**



*AWT Example by Association*

Let's see a simple example of AWT where we are creating instance of Frame class. Here, we are creating a TextField, Label and Button component on the Frame.

### **AWTExample2.java**

```
// importing Java AWT class
import java.awt.*;

// class AWTExample2 directly creates instance of Frame class
class AWTExample2 {

 // initializing using constructor
 AWTExample2() {

 // creating a Frame
 Frame f = new Frame();

 // creating a Label
 Label l = new Label("Employee id:");

 // creating a Button
 Button b = new Button("Submit");
 }
}
```

```
// creating a TextField
TextField t = new TextField();

// setting position of above components in the frame
l.setBounds(20, 80, 80, 30);
t.setBounds(20, 100, 80, 30);
b.setBounds(100, 100, 80, 30);

// adding components into frame
f.add(b);
f.add(l);
f.add(t);

// frame size 300 width and 300 height
f.setSize(400,300);

// setting the title of frame
f.setTitle("Employee info");

// no layout
f.setLayout(null);

// setting visibility of frame
f.setVisible(true);

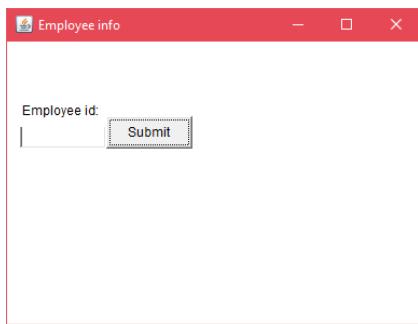
}

// main method
public static void main(String args[]) {

// creating instance of Frame class
AWTExample2 awt_obj = new AWTExample2();

}
```

}

**Output:****Layout Managers and Menus****BorderLayout (LayoutManagers)***Java LayoutManagers*

The LayoutManagers are used to arrange components in a particular manner. The **Java LayoutManagers** facilitates us to control the positioning and size of the components in GUI forms. LayoutManager is an interface that is implemented by all the classes of layout managers. There are the following classes that represent the layout managers:

1. java.awt.BorderLayout
2. java.awt.FlowLayout
3. java.awt.GridLayout
4. java.awt.CardLayout
5. java.awt.GridBagLayout
6. javax.swing.BoxLayout
7. javax.swing.GroupLayout
8. javax.swing.ScrollPaneLayout
9. javax.swing.SpringLayout etc.

*Java BorderLayout*

The BorderLayout is used to arrange the components in five regions: north, south, east, west, and center. Each region (area) may contain one component only. It is the default layout of a frame or window. The BorderLayout provides five constants for each region:

1. **public static final int NORTH**
2. **public static final int SOUTH**
3. **public static final int EAST**
4. **public static final int WEST**
5. **public static final int CENTER**

*Constructors of BorderLayout class:*

- o **BorderLayout():** creates a border layout but with no gaps between the components.
- o **BorderLayout(int hgap, int vgap):** creates a border layout with the given horizontal and vertical gaps between the components.

*Example of BorderLayout class: Using BorderLayout() constructor*

**FileName:** Border.java

```
import java.awt.*;
import javax.swing.*;

public class Border
{
 JFrame f;
 Border()
 {
 f = new JFrame();

 // creating buttons
 JButton b1 = new JButton("NORTH"); // the button will be labeled as NORTH
 JButton b2 = new JButton("SOUTH"); // the button will be labeled as SOUTH
 JButton b3 = new JButton("EAST"); // the button will be labeled as EAST
 JButton b4 = new JButton("WEST"); // the button will be labeled as WEST
 JButton b5 = new JButton("CENTER"); // the button will be labeled as CENTER

 f.add(b1, BorderLayout.NORTH); // b1 will be placed in the North Direction
 f.add(b2, BorderLayout.SOUTH); // b2 will be placed in the South Direction
 f.add(b3, BorderLayout.EAST); // b2 will be placed in the East Direction
```

```

f.add(b4, BorderLayout.WEST); // b2 will be placed in the West Direction
f.add(b5, BorderLayout.CENTER); // b2 will be placed in the Center

f.setSize(300, 300);
f.setVisible(true);
}

public static void main(String[] args) {
 new Border();
}
}

```

**Output:**



*Example of BorderLayout class: Using BorderLayout(int hgap, int vgap) constructor*

The following example inserts horizontal and vertical gaps between buttons using the parameterized constructor BorderLayout(int hgap, int gap)

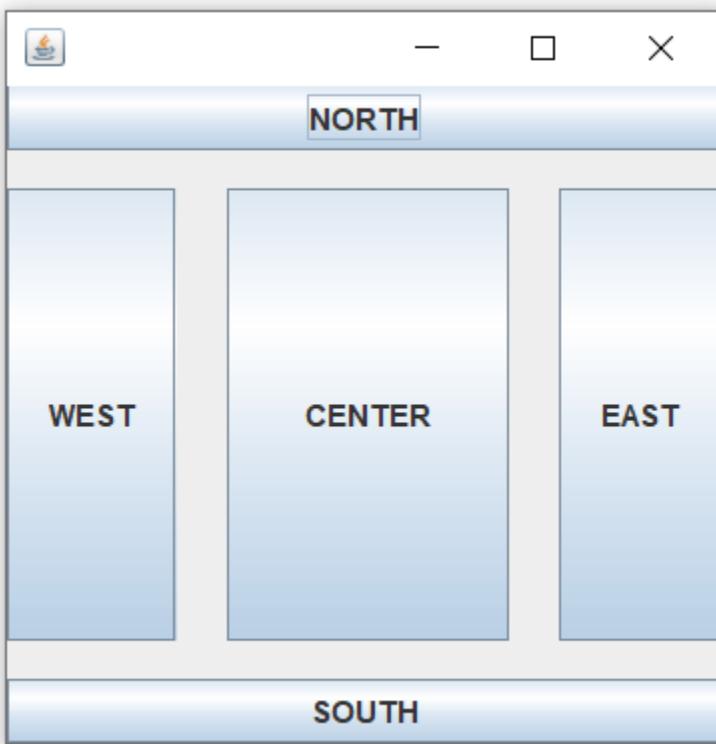
**FileName:** BorderLayoutExample.java

```

// import statement
import java.awt.*;

```

```
import javax.swing.*;
public class BorderLayoutExample
{
 JFrame jframe;
 // constructor
 BorderLayoutExample()
 {
 // creating a Frame
 jframe = new JFrame();
 // create buttons
 JButton btn1 = new JButton("NORTH");
 JButton btn2 = new JButton("SOUTH");
 JButton btn3 = new JButton("EAST");
 JButton btn4 = new JButton("WEST");
 JButton btn5 = new JButton("CENTER");
 // creating an object of the BorderLayout class using
 // the parameterized constructor where the horizontal gap is 20
 // and vertical gap is 15. The gap will be evident when buttons are placed
 // in the frame
 jframe.setLayout(new BorderLayout(20, 15));
 jframe.add(btn1, BorderLayout.NORTH);
 jframe.add(btn2, BorderLayout.SOUTH);
 jframe.add(btn3, BorderLayout.EAST);
 jframe.add(btn4, BorderLayout.WEST);
 jframe.add(btn5, BorderLayout.CENTER);
 jframe.setSize(300,300);
 jframe.setVisible(true);
 }
 // main method
 public static void main(String args[])
 {
 new BorderLayoutExample();
 }
}
```

**Output:***Java BorderLayout: Without Specifying Region*

The add() method of the JFrame class can work even when we do not specify the region. In such a case, only the latest component added is shown in the frame, and all the components added previously get discarded. The latest component covers the whole area. The following example shows the same.

**FileName:** BorderLayoutWithoutRegionExample.java

```
// import statements
import java.awt.*;
import javax.swing.*;

public class BorderLayoutWithoutRegionExample
{
 JFrame jframe;
```

```
// constructor
BorderLayoutWithoutRegionExample()
{
 jframe = new JFrame();

 JButton btn1 = new JButton("NORTH");
 JButton btn2 = new JButton("SOUTH");
 JButton btn3 = new JButton("EAST");
 JButton btn4 = new JButton("WEST");
 JButton btn5 = new JButton("CENTER");

 // horizontal gap is 7, and the vertical gap is 7
 // Since region is not specified, the gaps are of no use
 jframe.setLayout(new BorderLayout(7, 7));

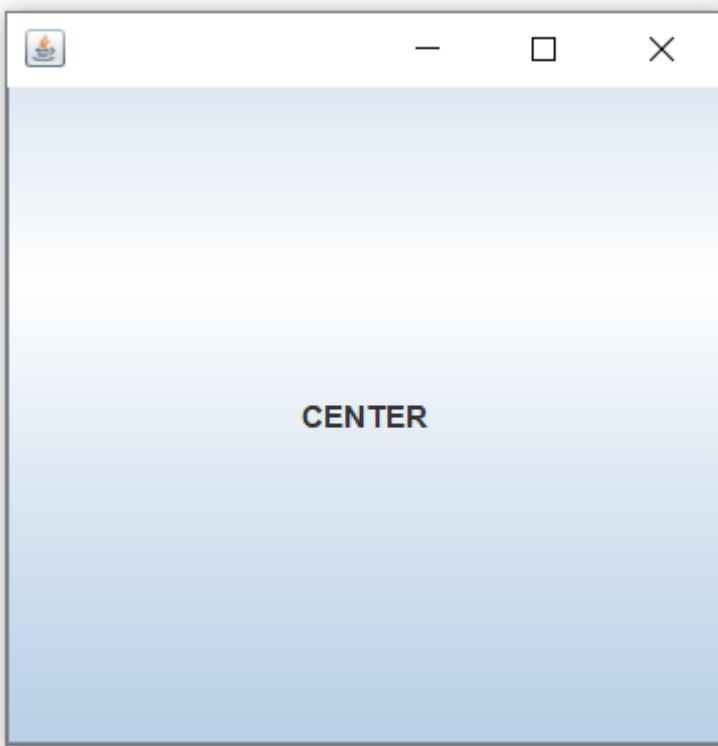
 // each button covers the whole area
 // however, the btn5 is the latest button
 // that is added to the frame; therefore, btn5
 // is shown
 jframe.add(btn1);
 jframe.add(btn2);
 jframe.add(btn3);
 jframe.add(btn4);
 jframe.add(btn5);

 jframe.setSize(300,300);
 jframe.setVisible(true);
}

// main method
public static void main(String args[])
{
 new BorderLayoutWithoutRegionExample();
}
```

}

**Output:**



**Java GridLayout**

The Java GridLayout class is used to arrange the components in a rectangular grid. One component is displayed in each rectangle.

*Constructors of GridLayout class*

1. **GridLayout()**: creates a grid layout with one column per component in a row.
2. **GridLayout(int rows, int columns)**: creates a grid layout with the given rows and columns but no gaps between the components.
3. **GridLayout(int rows, int columns, int hgap, int vgap)**: creates a grid layout with the given rows and columns along with given horizontal and vertical gaps.

*Example of GridLayout class: Using GridLayout() Constructor*

The GridLayout() constructor creates only one row. The following example shows the usage of the parameterless constructor.

**FileName:** GridLayoutExample.java

```
// import statements
import java.awt.*;
import javax.swing.*;

public class GridLayoutExample
{
 JFrame frameObj;

 // constructor
 GridLayoutExample()
 {
 frameObj = new JFrame();

 // creating 9 buttons
 JButton btn1 = new JButton("1");
 JButton btn2 = new JButton("2");
 JButton btn3 = new JButton("3");
 JButton btn4 = new JButton("4");
 JButton btn5 = new JButton("5");
 JButton btn6 = new JButton("6");
 JButton btn7 = new JButton("7");
 JButton btn8 = new JButton("8");
 JButton btn9 = new JButton("9");

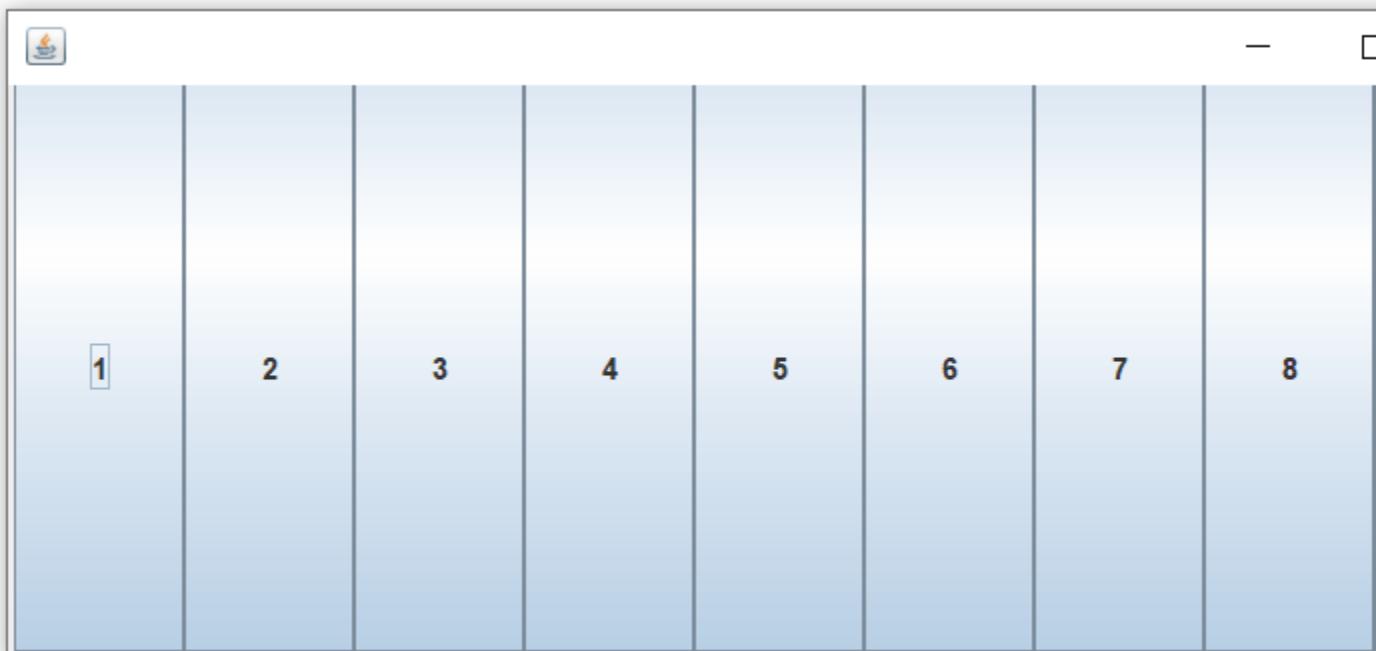
 // adding buttons to the frame
 // since, we are using the parameterless constructor, therefore;
 // the number of columns is equal to the number of buttons we
 // are adding to the frame. The row count remains one.
 frameObj.add(btn1); frameObj.add(btn2); frameObj.add(btn3);
 frameObj.add(btn4); frameObj.add(btn5); frameObj.add(btn6);
```

```
frameObj.add(btn7); frameObj.add(btn8); frameObj.add(btn9);

// setting the grid layout using the parameterless constructor
frameObj.setLayout(new GridLayout());

frameObj.setSize(300, 300);
frameObj.setVisible(true);
}

// main method
public static void main(String args[])
{
new GridLayoutExample();
}
```

**Output:**

*Example of GridLayout class: Using GridLayout(int rows, int columns) Constructor*

**FileName:** MyGridLayout.java

```
import java.awt.*;
import javax.swing.*;
public class MyGridLayout{
JFrame f;
MyGridLayout(){
 f=new JFrame();
 JButton b1=new JButton("1");
 JButton b2=new JButton("2");
 JButton b3=new JButton("3");
 JButton b4=new JButton("4");
 JButton b5=new JButton("5");
 JButton b6=new JButton("6");
 JButton b7=new JButton("7");
 JButton b8=new JButton("8");
 JButton b9=new JButton("9");
 // adding buttons to the frame
 f.add(b1); f.add(b2); f.add(b3);
 f.add(b4); f.add(b5); f.add(b6);
 f.add(b7); f.add(b8); f.add(b9);

 // setting grid layout of 3 rows and 3 columns
 f.setLayout(new GridLayout(3,3));
 f.setSize(300,300);
 f.setVisible(true);
}
public static void main(String[] args) {
 new MyGridLayout();
}
```

**Output:**



Example of GridLayout class: Using GridLayout(int rows, int columns, int hgap, int vgap)  
Constructor

The following example inserts horizontal and vertical gaps between buttons using the parameterized constructor GridLayout(int rows, int columns, int hgap, int vgap).

**FileName:** GridLayoutExample1.java

```
// import statements
import java.awt.*;
import javax.swing.*;

public class GridLayoutExample1
{
 JFrame frameObj;

 // constructor
 GridLayoutExample1()
 {
 frameObj = new JFrame();
```

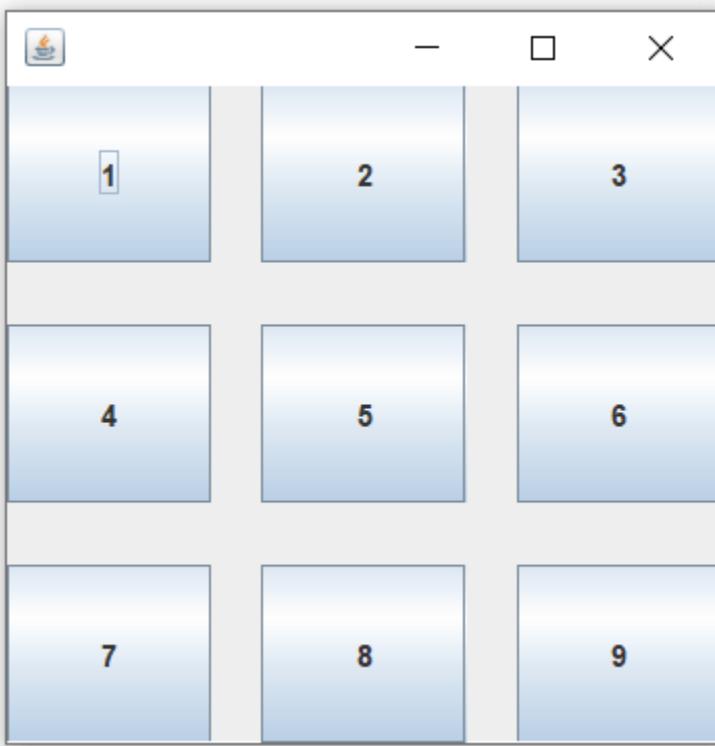
```
// creating 9 buttons
JButton btn1 = new JButton("1");
JButton btn2 = new JButton("2");
JButton btn3 = new JButton("3");
JButton btn4 = new JButton("4");
JButton btn5 = new JButton("5");
JButton btn6 = new JButton("6");
JButton btn7 = new JButton("7");
JButton btn8 = new JButton("8");
JButton btn9 = new JButton("9");

// adding buttons to the frame
// since, we are using the parameterless constructor, therefore;
// the number of columns is equal to the number of buttons we
// are adding to the frame. The row count remains one.
frameObj.add(btn1); frameObj.add(btn2); frameObj.add(btn3);
frameObj.add(btn4); frameObj.add(btn5); frameObj.add(btn6);
frameObj.add(btn7); frameObj.add(btn8); frameObj.add(btn9);

// setting the grid layout
// a 3 * 3 grid is created with the horizontal gap 20
// and vertical gap 25
frameObj.setLayout(new GridLayout(3, 3, 20, 25));
frameObj.setSize(300, 300);
frameObj.setVisible(true);
}

// main method
public static void main(String args[])
{
 new GridLayoutExample();
}
```

**Output:**



---

## java FlowLayout

The Java **FlowLayout** class is used to arrange the components in a line, one after another (in a flow). It is the default layout of the applet or panel.

### *Fields of FlowLayout class*

1. **public static final int LEFT**
2. **public static final int RIGHT**
3. **public static final int CENTER**
4. **public static final int LEADING**
5. **public static final int TRAILING**

### *Constructors of FlowLayout class*

1. **FlowLayout():** creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
2. **FlowLayout(int align):** creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.

3. **FlowLayout(int align, int hgap, int vgap):** creates a flow layout with the given alignment and the given horizontal and vertical gap.

*Example of FlowLayout class: Using FlowLayout() constructor*

**FileName:** FlowLayoutExample.java

```
// import statements
import java.awt.*;
import javax.swing.*;

public class FlowLayoutExample
{
 JFrame frameObj;

 // constructor
 FlowLayoutExample()
 {
 // creating a frame object
 frameObj = new JFrame();

 // creating the buttons
 JButton b1 = new JButton("1");
 JButton b2 = new JButton("2");
 JButton b3 = new JButton("3");
 JButton b4 = new JButton("4");
 JButton b5 = new JButton("5");
 JButton b6 = new JButton("6");
 JButton b7 = new JButton("7");
 JButton b8 = new JButton("8");
 JButton b9 = new JButton("9");
 JButton b10 = new JButton("10");
```

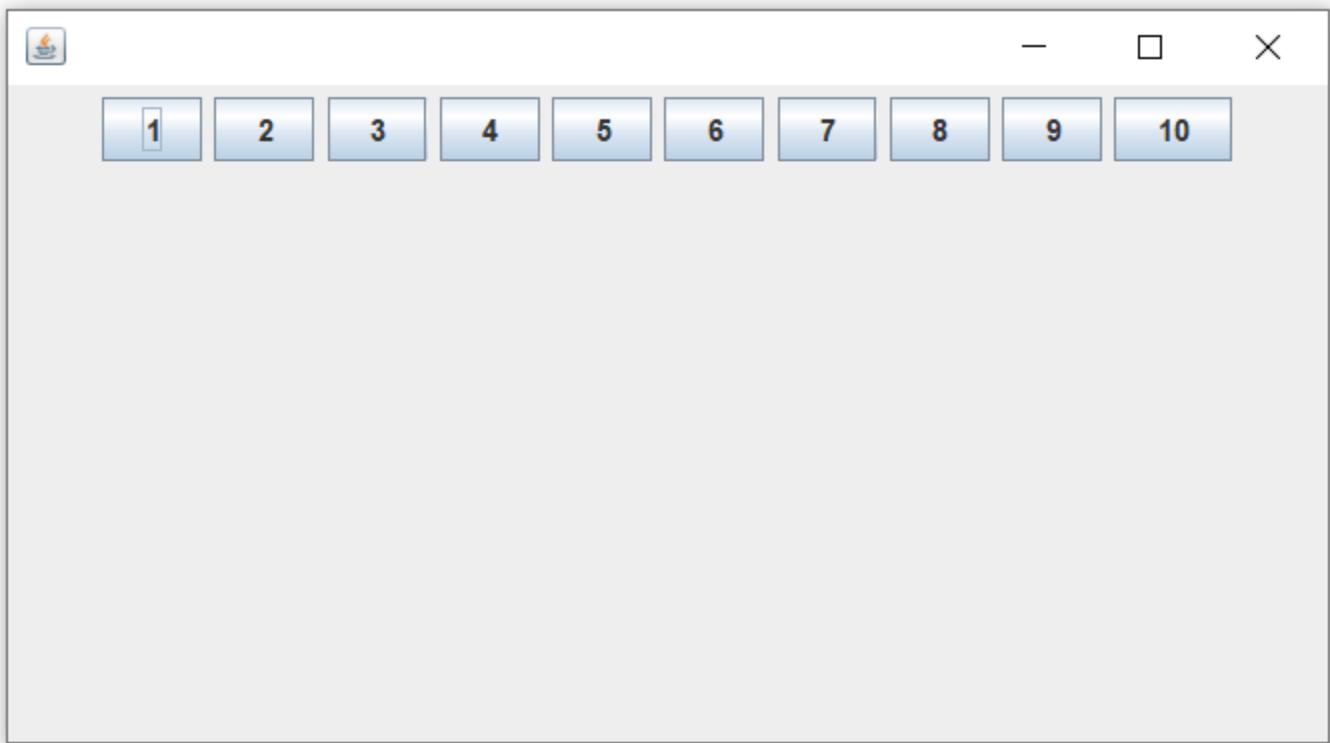
```
// adding the buttons to frame
frameObj.add(b1); frameObj.add(b2); frameObj.add(b3); frameObj.add(b4);
frameObj.add(b5); frameObj.add(b6); frameObj.add(b7); frameObj.add(b8);
frameObj.add(b9); frameObj.add(b10);

// parameter less constructor is used
// therefore, alignment is center
// horizontal as well as the vertical gap is 5 units.
frameObj.setLayout(new FlowLayout());

frameObj.setSize(300, 300);
frameObj.setVisible(true);
}

// main method
public static void main(String argvs[])
{
 new FlowLayoutExample();
}
}
```

**Output:**



*Example of FlowLayout class: Using FlowLayout(int align) constructor*

**FileName:** MyFlowLayout.java

```
import java.awt.*;
import javax.swing.*;

public class MyFlowLayout{
JFrame f;
MyFlowLayout(){
f=new JFrame();

JButton b1=new JButton("1");
JButton b2=new JButton("2");
JButton b3=new JButton("3");
JButton b4=new JButton("4");
JButton b5=new JButton("5");

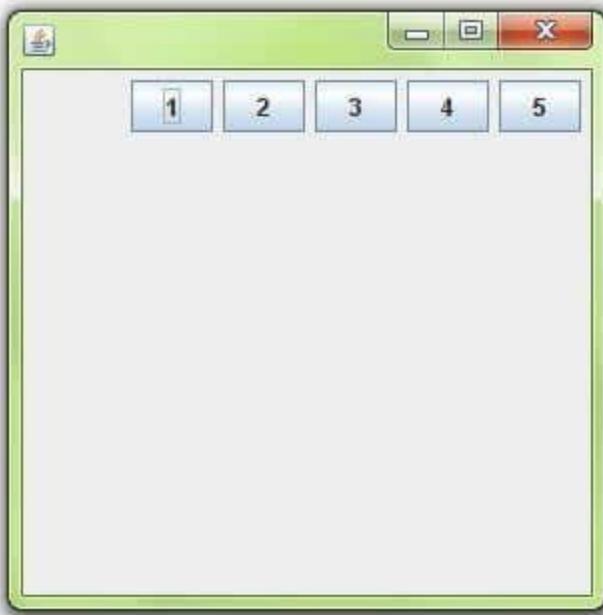
// adding buttons to the frame
```

```
f.add(b1); f.add(b2); f.add(b3); f.add(b4); f.add(b5);

// setting flow layout of right alignment
f.setLayout(new FlowLayout(FlowLayout.RIGHT));

f.setSize(300,300);
f.setVisible(true);
}

public static void main(String[] args) {
 new MyFlowLayout();
}
}
```

**Output:**

Example of FlowLayout class: Using FlowLayout(int align, int hgap, int vgap) constructor

**FileName:** FlowLayoutExample1.java

```
// import statement
import java.awt.*;
import javax.swing.*;
```

```
public class FlowLayoutExample1
{
 JFrame frameObj;

 // constructor
 FlowLayoutExample1()
 {
 // creating a frame object
 frameObj = new JFrame();

 // creating the buttons
 JButton b1 = new JButton("1");
 JButton b2 = new JButton("2");
 JButton b3 = new JButton("3");
 JButton b4 = new JButton("4");
 JButton b5 = new JButton("5");
 JButton b6 = new JButton("6");
 JButton b7 = new JButton("7");
 JButton b8 = new JButton("8");
 JButton b9 = new JButton("9");
 JButton b10 = new JButton("10");

 // adding the buttons to frame
 frameObj.add(b1); frameObj.add(b2); frameObj.add(b3); frameObj.add(b4);
 frameObj.add(b5); frameObj.add(b6); frameObj.add(b7); frameObj.add(b8);
 frameObj.add(b9); frameObj.add(b10);

 // parameterized constructor is used
 // where alignment is left
 // horizontal gap is 20 units and vertical gap is 25 units.
 frameObj.setLayout(new FlowLayout(FlowLayout.LEFT, 20, 25));
 }
}
```

```

frameObj.setSize(300, 300);
frameObj.setVisible(true);
}
// main method
public static void main(String args[])
{
 new FlowLayoutExample1();
}
}

```

**Output:**

Java BoxLayout

The **Java BoxLayout class** is used to arrange the components either vertically or horizontally. For this purpose, the BoxLayout class provides four constants. They are as follows:

*Note: The BoxLayout class is found in javax.swing package.*

*Fields of BoxLayout Class*

1. **public static final int X\_AXIS:** Alignment of the components are horizontal from left to right.
2. **public static final int Y\_AXIS:** Alignment of the components are vertical from top to bottom.
3. **public static final int LINE\_AXIS:** Alignment of the components is similar to the way words are aligned in a line, which is based on the ComponentOrientation property of the container. If the ComponentOrientation property of the container is horizontal, then the components are aligned horizontally; otherwise, the components are aligned vertically. For horizontal orientations, we have two cases: left to right, and right to left. If the value ComponentOrientation property of the container is from left to right, then the components are rendered from left to right, and for right to left, the rendering of components is also from right to left. In the case of vertical orientations, the components are always rendered from top to bottom.
4. **public static final int PAGE\_AXIS:** Alignment of the components is similar to the way text lines are put on a page, which is based on the ComponentOrientation property of the container. If the ComponentOrientation property of the container is horizontal, then

components are aligned vertically; otherwise, the components are aligned horizontally. For horizontal orientations, we have two cases: left to right, and right to left. If the value ComponentOrientation property of the container is also from left to right, then the components are rendered from left to right, and for right to left, the rendering of components is from right to left. In the case of vertical orientations, the components are always rendered from top to bottom.

*Constructor of BoxLayout class*

1. **BoxLayout(Container c, int axis):** creates a box layout that arranges the components with the given axis.

*Example of BoxLayout class with Y-AXIS:*

**FileName:** BoxLayoutExample1.java

```
import java.awt.*;
import javax.swing.*;

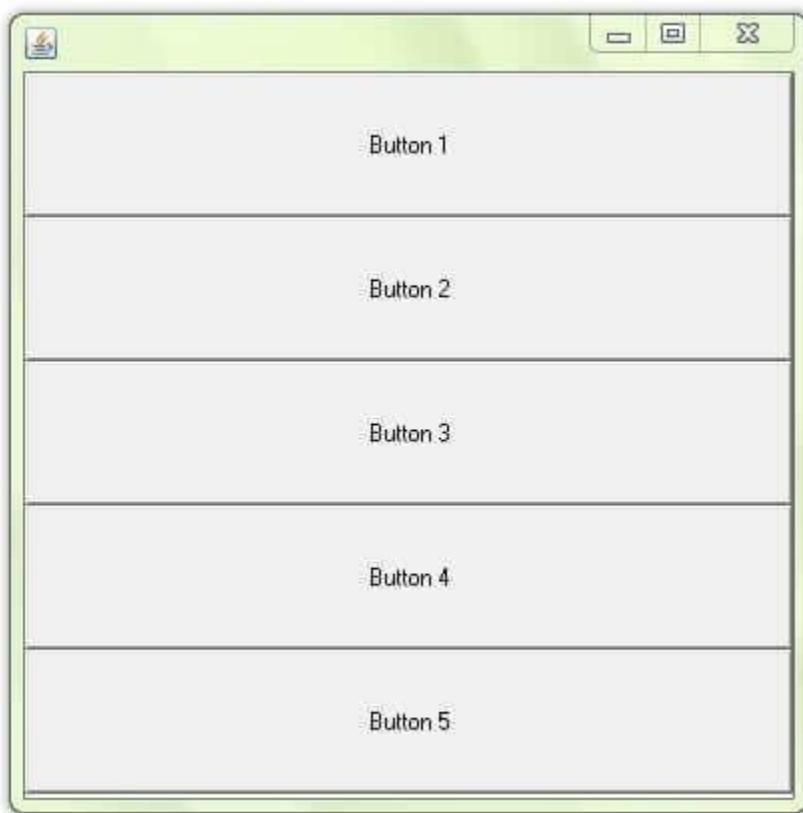
public class BoxLayoutExample1 extends Frame {
 Button buttons[];

 public BoxLayoutExample1 () {
 buttons = new Button [5];

 for (int i = 0;i<5;i++) {
 buttons[i] = new Button ("Button " + (i + 1));
 // adding the buttons so that it can be displayed
 add (buttons[i]);
 }
 // the buttons will be placed horizontally
 setLayout (new BoxLayout (this, BoxLayout.Y_AXIS));
 setSize(400,400);
 setVisible(true);
 }
 // main method
 public static void main(String args[]){

```

```
BoxLayoutExample1 b=new BoxLayoutExample1();
}
}
```

**Output:**

*Example of BoxLayout class with X-AXIS*

**FileName:** BoxLayoutExample2.java

49.2M  
1.1K

```
import java.awt.*;
import javax.swing.*;

public class BoxLayoutExample2 extends Frame {
 Button buttons[];

 public BoxLayoutExample2() {
```

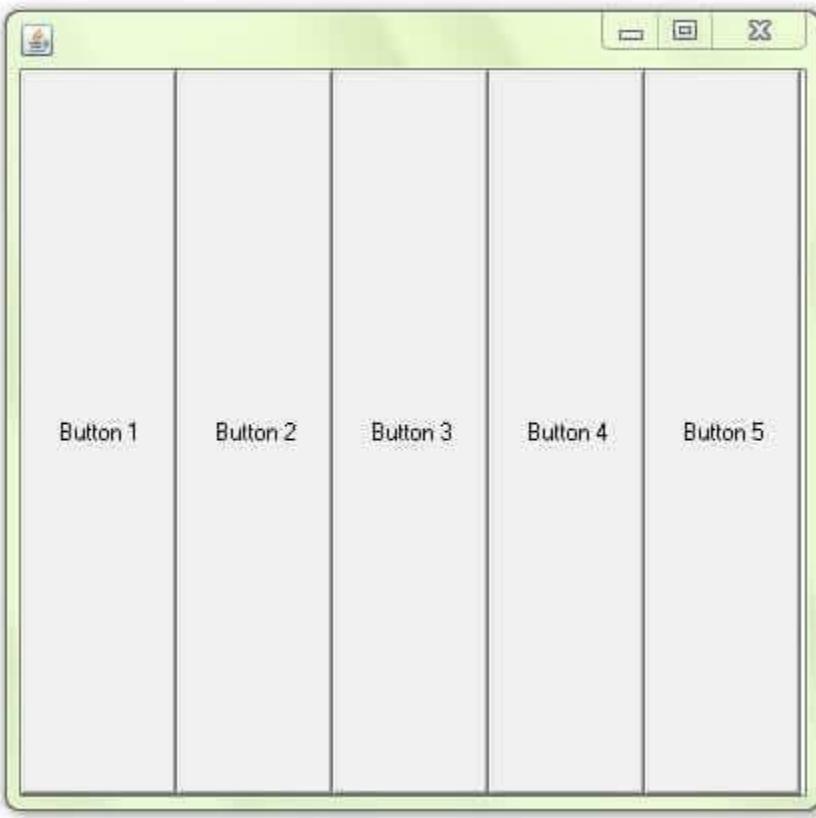
```
buttons = new Button [5];

for (int i = 0;i<5;i++) {
 buttons[i] = new Button ("Button " + (i + 1));
 // adding the buttons so that it can be displayed
 add (buttons[i]);
}

// the buttons in the output will be aligned vertically
setLayout (new BoxLayout(this, BoxLayout.X_AXIS));
setSize(400,400);
setVisible(true);
}

// main method
public static void main(String args[]){
 BoxLayoutExample2 b=new BoxLayoutExample2();
}
}
```

**Output:**



*Example of BoxLayout Class with LINE\_AXIS*

The following example shows the effect of setting the value of ComponentOrientation property of the container to RIGHT\_TO\_LEFT. If we do not set the value of ComponentOrientation property, then the components would be laid out from left to right. Comment line 11, and see it yourself.

**FileName:** BoxLayoutExample3.java

```
// import statements
import java.awt.*;
import javax.swing.*;

public class BoxLayoutExample3 extends Frame
{
 Button buttons[];

 // constructor of the class
 public BoxLayoutExample3()
```

```
{
 buttons = new Button[5];
 setComponentOrientation(ComponentOrientation.RIGHT_TO_LEFT); // line 11

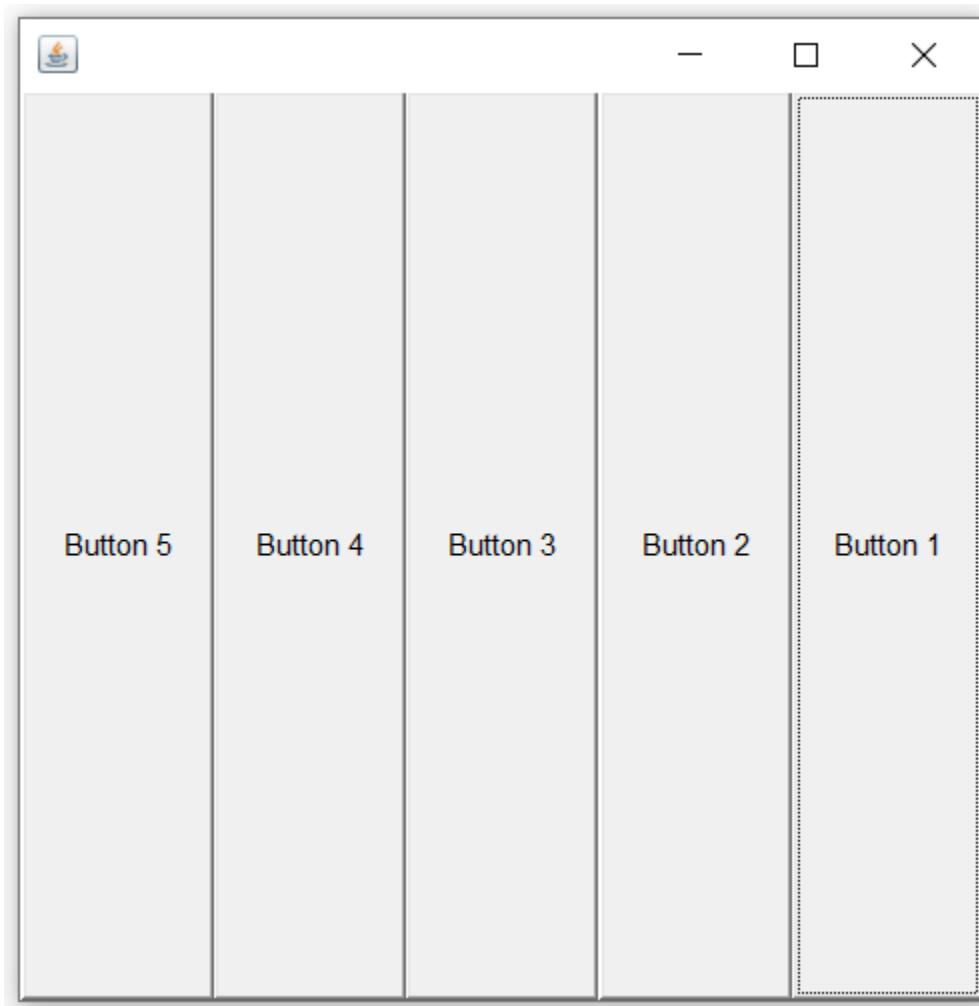
 for (int i = 0; i < 5; i++)
 {
 buttons[i] = new Button ("Button " + (i + 1));

 // adding the buttons so that it can be displayed
 add (buttons[i]);
 }

 // the ComponentOrientation is set to RIGHT_TO_LEFT. Therefore,
 // the added buttons will be rendered from right to left
 setLayout (new BoxLayout(this, BoxLayout.LINE_AXIS));
 setSize(400, 400);
 setVisible(true);
}

// main method
public static void main(String args[])
{
 // creating an object of the class BoxLayoutExample3
 BoxLayoutExample3 obj = new BoxLayoutExample3();
}
```

**Output:**



*Example of BoxLayout Class with PAGE\_AXIS*

The following example shows how to use PAGE\_AXIS.

**FileName:** BoxLayoutExample4.java

```
// import statements
import java.awt.*;
import javax.swing.*;

public class BoxLayoutExample4 extends Frame
{
 Button buttons[];
 // constructor of the class
 public BoxLayoutExample4()
```

```
{
JFrame f = new JFrame();
JPanel pnl = new JPanel();
buttons = new Button[5];
GridBagConstraints constrntObj = new GridBagConstraints();

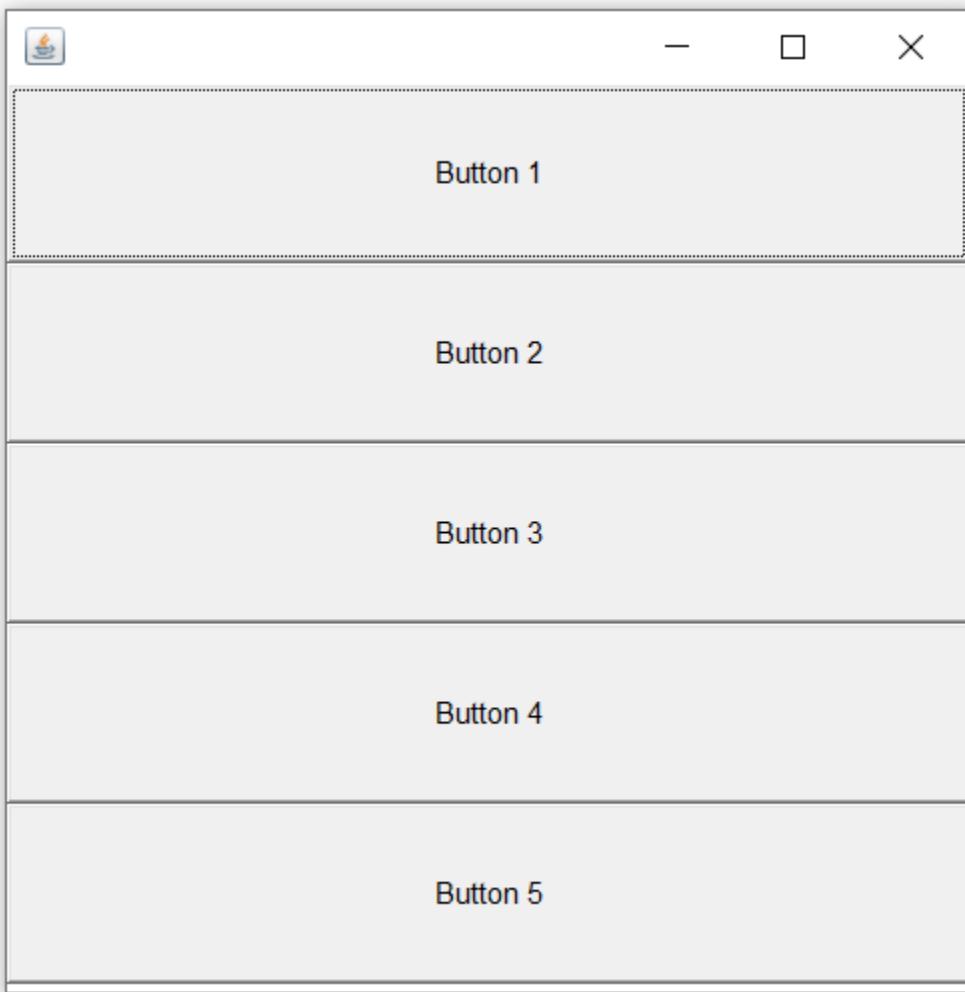
constrntObj.fill = GridBagConstraints.VERTICAL;
for (int i = 0; i < 5; i++)
{
 buttons[i] = new Button ("Button " + (i + 1));

 // adding the buttons so that it can be displayed
 add(buttons[i]);
}

// the components will be displayed just like the line is present on a page
setLayout (new BoxLayout(this, BoxLayout.PAGE_AXIS));
setSize(400, 400);
setVisible(true);
}

// main method
public static void main(String argvs[])
{
 // creating an object of the class BoxLayoutExample4
 BoxLayoutExample4 obj = new BoxLayoutExample4();
}
```

**Output:**



The above output shows that the buttons are aligned horizontally. Now, we will display the buttons vertically using the PAGE\_AXIS.

**FileName:** BoxLayoutExample5.java

```
// import statements
import java.awt.*;
import javax.swing.*;

public class BoxLayoutExample5 extends Frame
{
 Button buttons[];

 // constructor of the class
```

```
public BoxLayoutExample5()
{
JFrame f = new JFrame();
buttons = new Button[5];

// Creating a Box whose alignment is horizontal
Box horizontalBox = Box.createHorizontalBox();

// ContentPane returns a container
Container contentPane = f.getContentPane();

for (int i = 0; i < 5; i++)
{
 buttons[i] = new Button ("Button " + (i + 1));

 // adding the buttons to the box so that it can be displayed
 horizontalBox.add(buttons[i]);
}

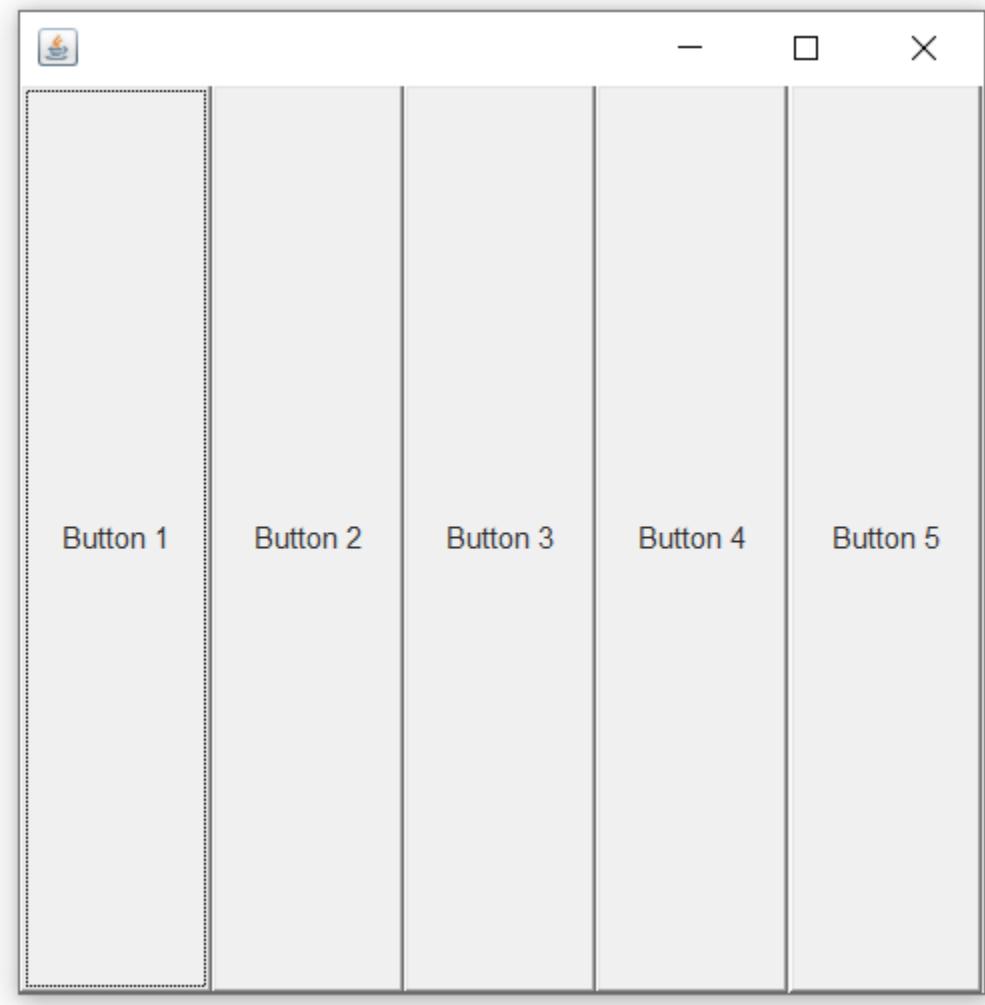
// adding the box and the borderlayout to the content pane
contentPane.add(horizontalBox, BorderLayout.NORTH);

// now, the rendered components are displayed vertically.
// it is because the box is aligned horizontally
f.setLayout (new BoxLayout(contentPane, BoxLayout.PAGE_AXIS));

f.setSize(400, 400);
f.setVisible(true);
}

// main method
public static void main(String args[])
{
// creating an object of the class BoxLayoutExample5
```

```
BoxLayoutExample5 obj = new BoxLayoutExample5();
}
}
```

**Output:**

---

**Java CardLayout**

The **Java CardLayout** class manages the components in such a manner that only one component is visible at a time. It treats each component as a card that is why it is known as CardLayout.

*Constructors of CardLayout Class*

1. **CardLayout()**: creates a card layout with zero horizontal and vertical gap.

2. **CardLayout(int hgap, int vgap):** creates a card layout with the given horizontal and vertical gap.

*Commonly Used Methods of CardLayout Class*

- **public void next(Container parent):** is used to flip to the next card of the given container.
- **public void previous(Container parent):** is used to flip to the previous card of the given container.
- **public void first(Container parent):** is used to flip to the first card of the given container.
- **public void last(Container parent):** is used to flip to the last card of the given container.
- **public void show(Container parent, String name):** is used to flip to the specified card with the given name.

*Example of CardLayout Class: Using Default Constructor*

The following program uses the next() method to move to the next card of the container.

**FileName:** CardLayoutExample1.java

```
// import statements
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class CardLayoutExample1 extends JFrame implements ActionListener
{
 CardLayout crd;

 // button variables to hold the references of buttons
 JButton btn1, btn2, btn3;
 Container cPane;

 // constructor of the class
```

```
CardLayoutExample1()
{
 cPane = getContentPane();

 //default constructor used
 // therefore, components will
 // cover the whole area
 crd = new CardLayout();

 cPane.setLayout(crd);

 // creating the buttons
 btn1 = new JButton("Apple");
 btn2 = new JButton("Boy");
 btn3 = new JButton("Cat");

 // adding listeners to it
 btn1.addActionListener(this);
 btn2.addActionListener(this);
 btn3.addActionListener(this);

 cPane.add("a", btn1); // first card is the button btn1
 cPane.add("b", btn2); // first card is the button btn2
 cPane.add("c", btn3); // first card is the button btn3

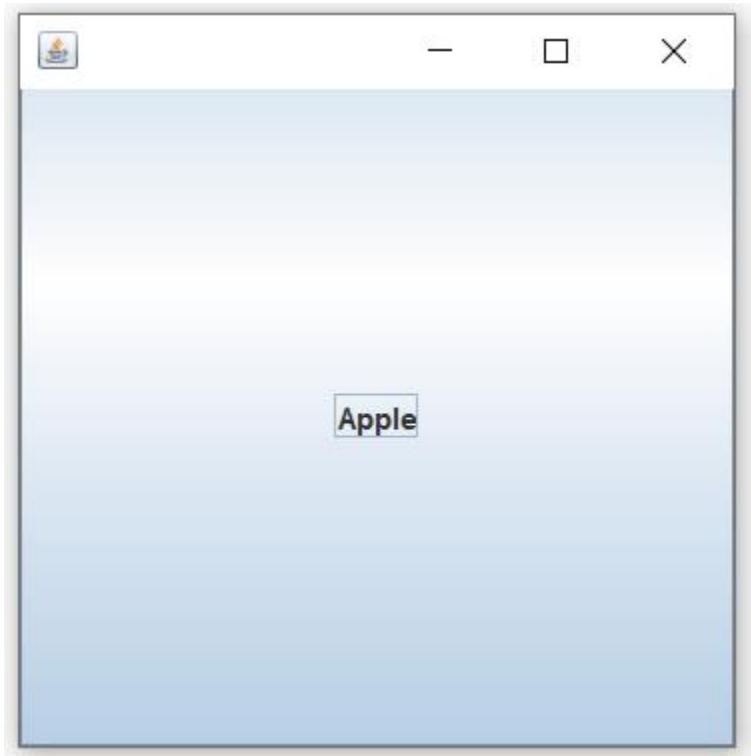
}

public void actionPerformed(ActionEvent e)
{
 // Upon clicking the button, the next card of the container is shown
 // after the last card, again, the first card of the container is shown upon clicking
 crd.next(cPane);
}
```

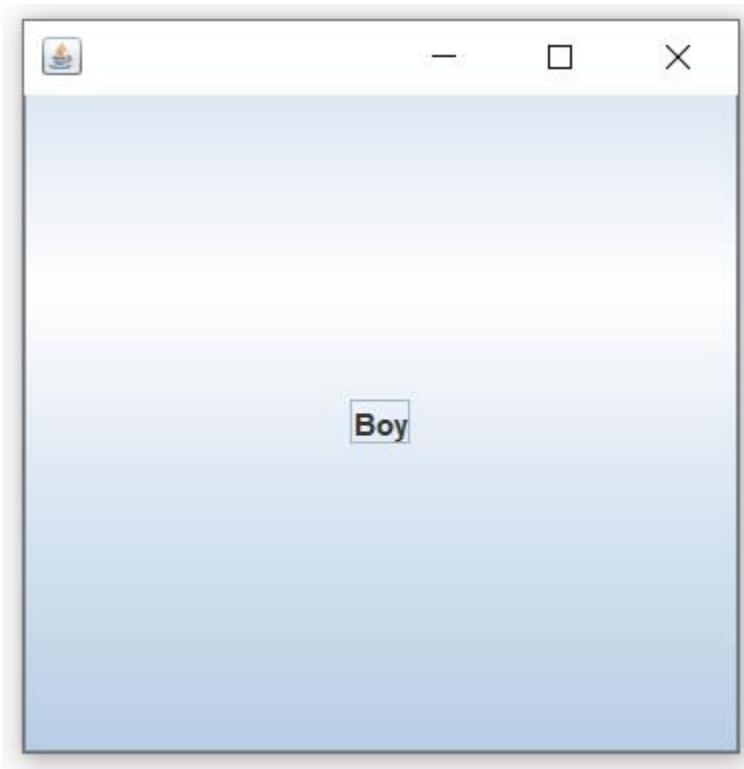
```
// main method
public static void main(String argvs[]){
 // creating an object of the class CardLayoutExample1
 CardLayoutExample1 crdl = new CardLayoutExample1();

 // size is 300 * 300
 crdl.setSize(300, 300);
 crdl.setVisible(true);
 crdl.setDefaultCloseOperation(EXIT_ON_CLOSE);
}
}
```

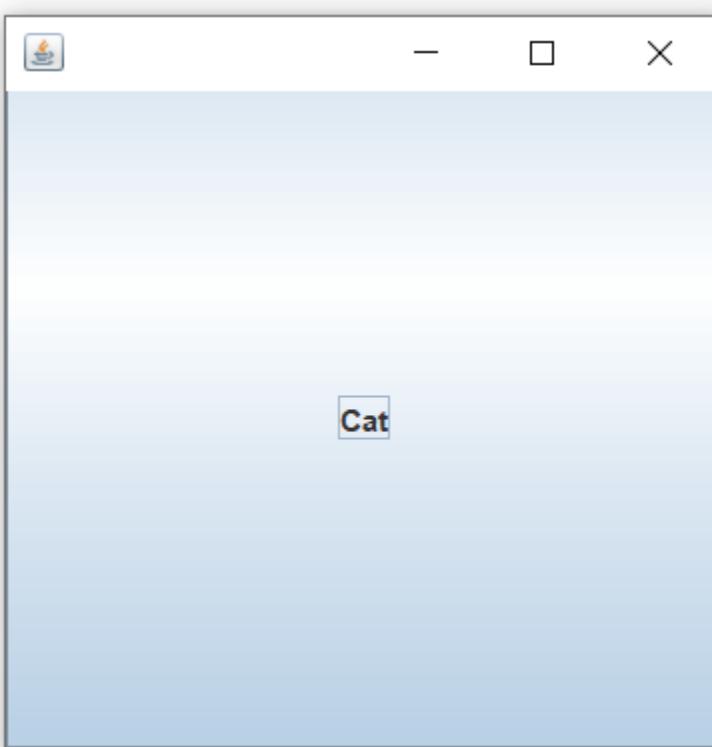
**Output:**



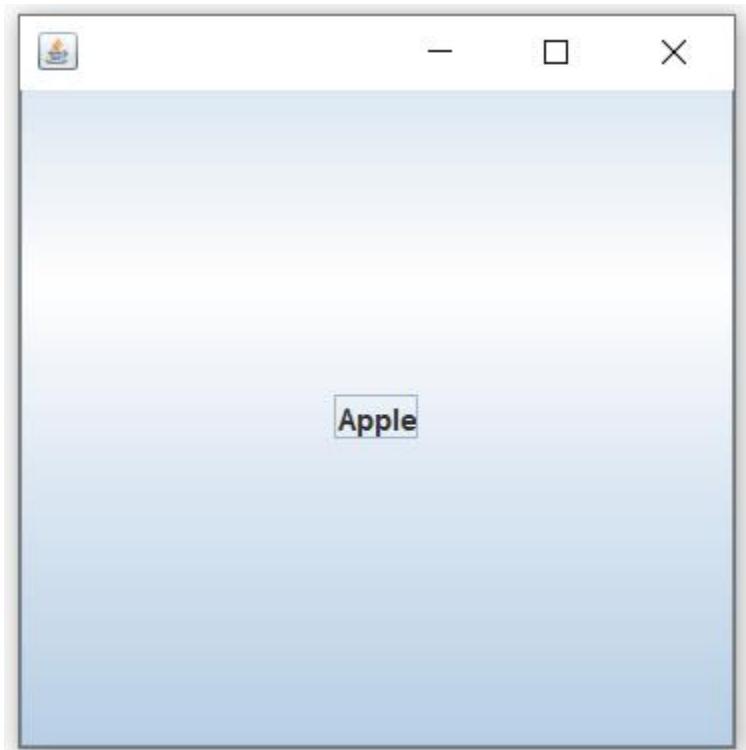
When the button named apple is clicked, we get



When the boy button is clicked, we get



Again, we reach the first card of the container if the cat button is clicked, and the cycle continues.



*Example of CardLayout Class: Using Parameterized Constructor*

**FileName:** CardLayoutExample2.java

```
import java.awt.*;
import java.awt.event.*;

import javax.swing.*;

public class CardLayoutExample2 extends JFrame implements ActionListener{
 CardLayout card;
 JButton b1,b2,b3;
 Container c;
 CardLayoutExample2(){
 c=getContentPane();
 card=new CardLayout(40,30);
 //create CardLayout object with 40 hor space and 30 ver space
```

```
c.setLayout(card);

b1=new JButton("Apple");
b2=new JButton("Boy");
b3=new JButton("Cat");
b1.addActionListener(this);
b2.addActionListener(this);
b3.addActionListener(this);

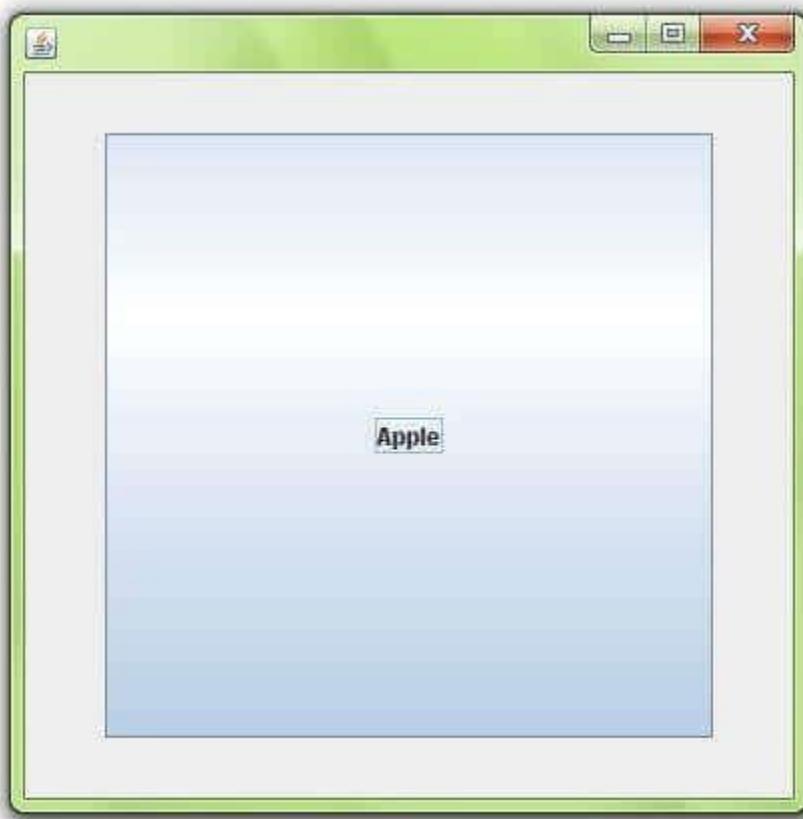
c.add("a",b1);c.add("b",b2);c.add("c",b3);

}

public void actionPerformed(ActionEvent e) {
card.next(c);
}

public static void main(String[] args) {
CardLayoutExample2 cl=new CardLayoutExample2();
cl.setSize(400,400);
cl.setVisible(true);
cl.setDefaultCloseOperation(EXIT_ON_CLOSE);
}
}
```

**Output:**



### *Usage of the Methods of the CardLayout Class*

The following example shows how one can use different methods of the CardLayout class.

**FileName:** CardLayoutExample3.java

```
// Import statements.
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;

public class CardLayoutExample3 extends JFrame
{

 // Initializing the value of
 // currCard to 1 .
private int currCard = 1;
```

```
// Declaring of objects
// of the CardLayout class.
private CardLayout cObjl;

// constructor of the class
public CardLayoutExample3()
{

 // Method to set the Title of the JFrame
 setTitle("Card Layout Methods");

 // Method to set the visibility of the JFrame
 setSize(310, 160);

 // Creating an Object of the "Jpanel" class
 JPanel cPanel = new JPanel();

 // Initializing of the object "cObjl"
 // of the CardLayout class.
 cObjl = new CardLayout();

 // setting the layout
 cPanel.setLayout(cObjl);

 // Initializing the object
 // "jPanel1" of the JPanel class.
 JPanel jPanel1 = new JPanel();

 // Initializing the object
 // "jPanel2" of the CardLayout class.
 JPanel jPanel2 = new JPanel();

 // Initializing the object
```

```
// "jPanel3" of the CardLayout class.
JPanel jPanel3 = new JPanel();

// Initializing the object
// "jPanel4" of the CardLayout class.
JPanel jPanel4 = new JPanel();

// Initializing the object
// "jl1" of the JLabel class.
JLabel jLabel1 = new JLabel("C1");

// Initializing the object
// "jLabel2" of the JLabel class.
JLabel jLabel2 = new JLabel("C2");

// Initializing the object
// "jLabel3" of the JLabel class.
JLabel jLabel3 = new JLabel("C3");

// Initializing the object
// "jLabel4" of the JLabel class.
JLabel jLabel4 = new JLabel("C4");

// Adding JLabel "jLabel1" to the JPanel "jPanel1".
jPanel1.add(jLabel1);

// Adding JLabel "jLabel2" to the JPanel "jPanel2".
jPanel2.add(jLabel2);

// Adding JLabel "jLabel3" to the JPanel "jPanel3".
jPanel3.add(jLabel3);

// Adding JLabel "jLabel4" to the JPanel "jPanel4".
jPanel4.add(jLabel4);
```

```
// Add the "jPanel1" on cPanel
cPanel.add(jPanel1, "1");

// Add the "jPanel2" on cPanel
cPanel.add(jPanel2, "2");

// Add the "jPanel3" on cPanel
cPanel.add(jPanel3, "3");

// Add the "jPanel4" on cPanel
cPanel.add(jPanel4, "4");

// Creating an Object of the "JPanel" class
JPanel btnPanel = new JPanel();

// Initializing the object
// "firstButton" of the JButton class.
JButton firstButton = new JButton("First");

// Initializing the object
// "nextButton" of the JButton class.
JButton nextButton = new JButton("->");

// Initializing the object
// "previousbtn" of JButton class.
JButton previousButton = new JButton("<-");

// Initializing the object
// "lastButton" of the JButton class.
JButton lastButton = new JButton("Last");

// Adding the JButton "firstbutton" on the JPanel.
btnPanel.add(firstButton);
```

```
// Adding the JButton "nextButton" on the JPanel.
btnPanel.add(nextButton);

// Adding the JButton "previousButton" on the JPanel.
btnPanel.add(previousButton);

// Adding the JButton "lastButton" on the JPanel.
btnPanel.add(lastButton);

// adding firstButton in the ActionListener
firstButton.addActionListener(new ActionListener()
{
 public void actionPerformed(ActionEvent ae)
 {

 // using the first cObjl CardLayout
 cObjl.first(cPanel);

 // value of currCard is 1
 currCard = 1;
 }
});

// add lastButton in ActionListener
lastButton.addActionListener(new ActionListener()
{
 public void actionPerformed(ActionEvent ae)
 {

 // using the last cObjl CardLayout
 cObjl.last(cPanel);

 // value of currCard is 4
```

```
currCard = 4;
}
});

// add nextButton in ActionListener
nextButton.addActionListener(new ActionListener()
{
 public void actionPerformed(ActionEvent ae)
{

 if (currCard < 4)
{

 // increase the value of currCard by 1
 currCard = currCard + 1;

 // show the value of currCard
 cObjl.show(cPanel, "" + (currCard));
 }
 }
});

// add previousButton in ActionListener
previousButton.addActionListener(new ActionListener()
{
 public void actionPerformed(ActionEvent ae)
{

 if (currCard > 1)
{

 // decrease the value
 // of currCard by 1
 currCard = currCard - 1;
 }
 }
});
```

```
// show the value of currCard
cObj1.show(cPanel, "" + (currCard));
}
}
});

// using to get the content pane
getContentPane().add(cPanel, BorderLayout.NORTH);

// using to get the content pane
getContentPane().add(btnPanel, BorderLayout.SOUTH);
}

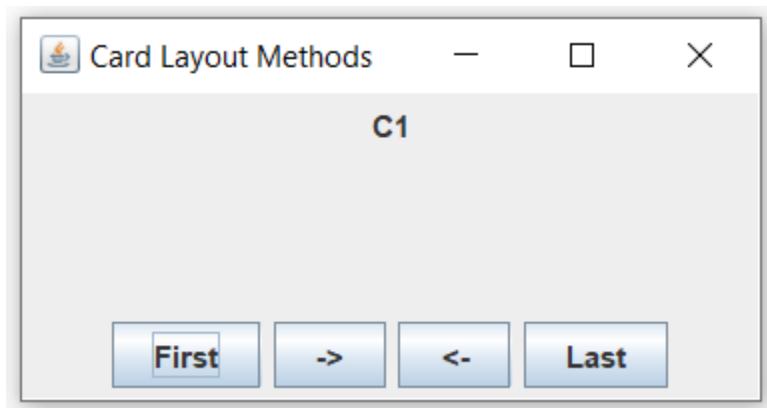
// main method
public static void main(String args[]){

// Creating an object of the CardLayoutExample3 class.
CardLayoutExample3 cll = new CardLayoutExample3();

// method to set the default operation of the JFrame.
cll.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

// aethod to set the visibility of the JFrame.
cll.setVisible(true);
}
}
```

### Output:



## **Servlet – life cycle of a servlet**

Java Servlets are programs that run on a Web or Application server and act as a middle layer between a request coming from a Web browser or other HTTP client and databases or applications on the HTTP server.

Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

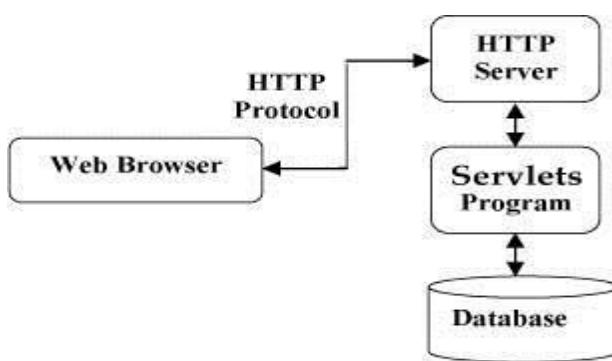
Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But Servlets offer several advantages in comparison with the CGI.

- Performance is significantly better.

- Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request.
- Servlets are platform-independent because they are written in Java.
- Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. So servlets are trusted.
- The full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms that you have seen already.

**Servlets Architecture:**

Following diagram shows the position of Servlets in a Web Application.



**Servlets Tasks:**

Servlets perform the following major tasks:

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

- Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.

Servlets Packages:

Java Servlets are Java classes run by a web server that has an interpreter that supports the Java Servlet specification.

Servlets can be created using the **javax.servlet** and **javax.servlet.http** packages, which are a standard part of the Java's enterprise edition, an expanded version of the Java class library that supports large-scale development projects.

These classes implement the Java Servlet and JSP specifications. At the time of writing this tutorial, the versions are Java Servlet 2.5 and JSP 2.1.

Java servlets have been created and compiled just like any other Java class. After you install the servlet packages and add them to your computer's Classpath, you can compile servlets with the JDK's Java compiler or any other current compiler.

## **Life cycle of servlet**

A servlet life cycle can be defined as the entire process from its creation till the destruction. 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.
- Now let us discuss the life cycle methods in details.

### **The init() method :**

The init method is designed to be called only once. It is called when the servlet is first created, and not called again for each user request. So, it is used for one-time initializations, just as with the init method of applets.

The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.

When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate. The init() method simply creates or loads some data that will be used throughout the life of the servlet.

The init method definition looks like this:

```
public void init() throws ServletException {
 // Initialization code...
}
```

### **The service() method :**

The service() method is the main method to perform the actual task. The servlet container (i.e. web server) calls the service() method to handle requests coming from the client( browsers) and to write the formatted response back to the client.

Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

Here is the signature of this method:

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

The service () method is called by the container and service method invokes doGet, doPost, doPut, doDelete, etc. methods as appropriate. So you have nothing to do with service() method but you override either doGet() or doPost() depending on what type of request you receive from the client.

The doGet() and doPost() are most frequently used methods with in each service request. Here is the signature of these two methods.

## The doGet() Method

A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

```
public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException{
 // Servlet code
}
```

## The doPost() Method

A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

```
public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException{
 // Servlet code
}
```

## The destroy() method :

The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.

After the destroy() method is called, the servlet object is marked for garbage collection. The destroy method definition looks like this:

```
public void destroy(){
 // Finalization code...
}
```

## Servlet Deployment:

By default, a servlet application is located at the path <Tomcat-installation-directory>/webapps/ROOT and the class file would reside in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/classes.

If you have a fully qualified class name of **com.myorg.MyServlet**, then this servlet class must be located in WEB-INF/classes/com/myorg/MyServlet.class.

For now, let us copy HelloWorld.class into <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/classes and create following entries in **web.xml** file located in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/

```
<servlet>
<servlet-name>HelloWorld</servlet-name>
<servlet-class>HelloWorld</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>HelloWorld</servlet-name>
<url-pattern>/HelloWorld</url-pattern>
</servlet-mapping>
```

Above entries to be created inside <web-app>...</web-app> tags available in web.xml file. There could be various entries in this table already available, but never mind.

You are almost done, now let us start tomcat server using <Tomcat-installation-directory>\bin\startup.bat (on windows) or <Tomcat-installation-directory>/bin/startup.sh (on Linux/Solaris etc.) and finally type **http://localhost:8080/HelloWorld** in browser's address box. If everything goes fine, you would get following result:

## **Servlet API**

Interfaces in javax.servlet package

Classes in javax.servlet package

Interfaces in javax.servlet.http package

Classes in javax.servlet.http package

The javax.servlet and javax.servlet.http packages represent interfaces and classes for servlet api.

The javax.servlet package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol.

The javax.servlet.http package contains interfaces and classes that are responsible for http requests only.

Let's see what are the interfaces of javax.servlet package.

Interfaces in javax.servlet package

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

Servlet

ServletRequest  
ServletResponse  
RequestDispatcher  
ServletConfig  
ServletContext  
SingleThreadModel  
Filter  
FilterConfig  
FilterChain  
ServletRequestListener  
ServletRequestAttributeListener  
ServletContextListener  
ServletContextAttributeListener

#### Classes in javax.servlet package

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

GenericServlet  
ServletInputStream  
ServletOutputStream  
ServletRequestWrapper  
ServletResponseWrapper  
ServletRequestEvent  
ServletContextEvent  
ServletRequestAttributeEvent  
ServletContextAttributeEvent  
ServletException  
UnavailableException

#### Interfaces in javax.servlet.http package

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

HttpServletRequest  
HttpServletResponse  
HttpSession  
HttpSessionListener  
HttpSessionAttributeListener  
HttpSessionBindingListener  
HttpSessionActivationListener  
HttpSessionContext (deprecated now)

#### Classes in javax.servlet.http package

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

HttpServlet  
Cookie  
HttpServletRequestWrapper  
HttpServletResponseWrapper  
HttpSessionEvent  
HttpSessionBindingEvent  
HttpUtils (deprecated now)

## Servlet Interface

1. [Servlet Interface](#)
2. [Methods of Servlet interface](#)

**Servlet interface provides** common behavio to all the servlets. Servlet interface defines methods that all servlets must implement.

Servlet interface needs to be implemented for creating any servlet (either directly or indirectly). It provides 3 life cycle methods that are used to initialize the servlet, to service the requests, and to destroy the servlet and 2 non-life cycle methods.

## Methods of Servlet interface

There are 5 methods in Servlet interface. The init, service and destroy are the life cycle methods of servlet. These are invoked by the web container.

Method	Description
<b>public void init(ServletConfig config)</b>	initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once.
<b>public void service(HttpServletRequest request, HttpServletResponse response)</b>	provides response for the incoming request. It is invoked at each request by the web container.
<b>public void destroy()</b>	is invoked only once and indicates that servlet is being destroyed.
<b>public ServletConfig getServletConfig()</b>	returns the object of ServletConfig.
<b>public String getServletInfo()</b>	returns information about servlet such as writer, copyright, version etc.

## GenericServlet class

1. [GenericServlet class](#)
2. [Methods of GenericServlet class](#)
3. [Example of GenericServlet class](#)

**GenericServlet** class implements **Servlet**, **ServletConfig** and **Serializable** interfaces. It provides the implementation of all the methods of these interfaces except the service method.

GenericServlet class can handle any type of request so it is protocol-independent.

You may create a generic servlet by inheriting the GenericServlet class and providing the implementation of the service method.

## Methods of GenericServlet class

There are many methods in GenericServlet class. They are as follows:

1. **public void init(ServletConfig config)** is used to initialize the servlet.
2. **public abstract void service(ServletRequest request, ServletResponse response)** provides service for the incoming request. It is invoked at each time when user requests for a servlet.
3. **public void destroy()** is invoked only once throughout the life cycle and indicates that servlet is being destroyed.
4. **public ServletConfig getServletConfig()** returns the object of ServletConfig.
5. **public String getServletInfo()** returns information about servlet such as writer, copyright, version etc.
6. **public void init()** it is a convenient method for the servlet programmers, now there is no need to call super.init(config)
7. **public ServletContext getServletContext()** returns the object of ServletContext.
8. **public String getInitParameter(String name)** returns the parameter value for the given parameter name.
9. **public Enumeration getInitParameterNames()** returns all the parameters defined in the web.xml file.
10. **public String getServletName()** returns the name of the servlet object.
11. **public void log(String msg)** writes the given message in the servlet log file.
12. **public void log(String msg, Throwable t)** writes the explanatory message in the servlet log file and a stack trace.

## **HttpServlet class**

1. [HttpServlet class](#)
2. [Methods of HttpServlet class](#)

The HttpServlet class extends the GenericServlet class and implements Serializable interface. It provides http specific methods such as doGet, doPost, doHead, doTrace etc.

### *Methods of HttpServlet class*

There are many methods in HttpServlet class. They are as follows:

1. **public void service(ServletRequest req, ServletResponse res)** dispatches the request to the protected service method by converting the request and response object into http type.

2. **protected void service(HttpServletRequest req, HttpServletResponse res)** receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.
3. **protected void doGet(HttpServletRequest req, HttpServletResponse res)** handles the GET request. It is invoked by the web container.
4. **protected void doPost(HttpServletRequest req, HttpServletResponse res)** handles the POST request. It is invoked by the web container.
5. **protected void doHead(HttpServletRequest req, HttpServletResponse res)** handles the HEAD request. It is invoked by the web container.
6. **protected void doOptions(HttpServletRequest req, HttpServletResponse res)** handles the OPTIONS request. It is invoked by the web container.
7. **protected void doPut(HttpServletRequest req, HttpServletResponse res)** handles the PUT request. It is invoked by the web container.
8. **protected void doTrace(HttpServletRequest req, HttpServletResponse res)** handles the TRACE request. It is invoked by the web container.
9. **protected void doDelete(HttpServletRequest req, HttpServletResponse res)** handles the DELETE request. It is invoked by the web container.
10. **protected long getLastModified(HttpServletRequest req)** returns the time when HttpServletRequest was last modified since midnight January 1, 1970 GMT

## Introduction to JSP

Jsp stands for java Server 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 %>.

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?

JavaServer Pages often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But JSP offer several advantages in comparison with the CGI.

- Performance is significantly better because JSP allows embedding Dynamic Elements in HTML Pages itself instead of having a separate CGI files.
- JSP are always compiled before it's processed by the server unlike CGI/Perl which requires the server to load an interpreter and the target script each time the page is requested.
- 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, JNDI, 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.

Finally, JSP is an integral part of J2EE, a complete platform for enterprise class applications. This means that JSP can play a part in the simplest applications to the most complex and demanding.C

## Advantages of JSP:

Following is the list of other advantages of using JSP over other technologies:

- Active Server Pages (ASP):The advantages of JSP are twofold. First, the dynamic part is written in Java, not Visual Basic or other MS specific language, so it is more powerful and easier to use. Second, it is portable to other operating systems and non-Microsoft Web servers.
- Pure Servlets:It is more convenient to write (and to modify!) regular HTML than to have plenty of println statements that generate the HTML.
- Server-Side Includes (SSI):SSI is really only intended for simple inclusions, not for "real" programs that use form data, make database connections, and the like.
- JavaScript:JavaScript can generate HTML dynamically on the client but can hardly interact with the web server to perform complex tasks like database access and image processing etc.

- Static HTML: Regular HTML, of course, cannot contain dynamic information

## JSP Declarations:

A declaration declares one or more variables or methods that you can use in Java code later in the JSP file. You must declare the variable or method before you use it in the JSP file. Following is the syntax of JSP Declarations: `<%!declaration;[declaration;]+...%>`

You can write XML equivalent of the above syntax as follows:

```
<jsp:declaration>
code fragment
</jsp:declaration>
```

Following is the simple example for JSP Declarations:

```
<%!int i =0;%>
<%!int a,b,c;%>
<%!Circle a =newCircle(2.0);%>
<\%Represents static <% literal.%\>
Represents static %>
```

literal.\'A single quote in an attribute that uses single quotes.\\"A double quote in an attribute that uses double quotes.

## JSP Directives:

A JSP directive affects the overall structure of the servlet class. It usually has the following form:

```
<%@directive attribute="value"%>
```

There are three types of directive tag:

Directive

Description

```
<%@ page ... %>
```

Defines page

dependent attributes, such as scripting language, error page, and buffering requirements.

<%@ include ... %>

Includes a file during the translation phase.

<%@ tag lib ... %>

Declares a tag library, containing custom actions, used in the page The <jsp:forward> Action

The forward action terminates the action of the current page and forwards the request to another resource such as a static page, another JSP page, or a Java Servlet. The simple syntax of this action is as follows:

<jsp:forward page="Relative URL"/>

Following is the list of required attributes associated with forward action:

Attribute

Description

page

Should consist of a relative URL of another resource such as a static page, another JSP page, or a Java Servlet

Example:

Let us reuse following two files (a) date.jsp and (b) main.jsp as follows:

Following is the content of date.jsp file:

<p>

Today's date:

<%=(new java.util.Date()).toLocaleString()%>

</p>

Here is the content of main.jsp file:

<html>

<head>

<title>

The include Action Example

</title>

</head>

<body>

<center>

<h2>

The include action Example

</h2>

<jsp:forward page="date.jsp"/>

</center>

</body>

</html>

Now let us keep all these files in root directory and try to access main.jsp. This would display result something like as below. Here it discarded content from main page and displayed content from forwarded page only.

Today's date: 12-Sep-2010 14:54:22

The <jsp:plugin> Action

The plugin action is used to insert Java components into a JSP page. It determines the type of browser and inserts the <object> or <embed> tags as needed. If the needed plugin is not present, it downloads the plugin and then executes the Java component. The Java component can be either an Applet or a JavaBean.

The plugin action has several attributes that correspond to common HTML tags used to format Java components.

The <param> element can also be used to send parameters to the Applet or Bean.

Following is the typical syntax of using plugin action:

<jsp:plugin type="applet" code base="dirname" code="MyApplet.class" width="60",height="80">

<jsp:param name="fontcolor" value="red"/>

<jsp:param name="background" value="black"/>

<jsp:fallback>

Unable to initialize Java Plugin

</jsp:fallback>

</jsp:plugin>

You can try this action using some applet if you are interested. A new element, the <fallback> element, can be

used to specify an error string to be sent to the user in case the component fails.

The <jsp:element> Action

The <jsp:attribute> Action

The <jsp:body> Action

The <jsp:element>, lt;jsp:attribute> and <jsp:body> actions are used to define XML elements dynamically. The word dynamically is important, because it means that the XML elements can be generated at request time rather than statically at compile time.

Following is a simple example to define XML elements dynamically:

```
<%@page language ="java" contentType="text/html"%>
<html xmlns="http://www.w3c.org/1999/xhtml" xmlns:jsp="http://java.sun.com/JSP/Page">
<head><title>
Genera
te XML Element
</title></head>
<body>
<jsp:element name="xmlElement">
<jsp:attribute name="xmlElementAttr">
Value for the attribute
</jsp:attribute>
<jsp:body>
Body for XML element
</jsp:body>
</jsp:element>
</body>
</html>
```

This would produce followin  
g HTML code at run time:

```
<html xmlns="http://www.w3c.org/1999/xhtml" xmlns:jsp="http://java.sun.com/JSP/Page">
<head><title>
Generate XML Element
</title></head>
<body>
<XmlElement xmlElementAttr= "Value for the attribute">
Body for XML element
</XmlElement>
</body>
```

## WEB PROGRAMMING

</html>

The <jsp:text> Action

The <jsp:text> action can be used to write template text in JSP pages and documents.

Following is the simple

syntax for this action:

```
<jsp:text>
```

Template data

```
</jsp:text>
```

## UNIT – V

### XML AND WEB SERVICES

Xml – Introduction-Form Navigation-XML Documents- XSL – XSLT- Web services-UDDI- WSDL-Java web services – Web resources.

#### Xml – Introduction

XML is a software- and hardware-independent tool for storing and transporting data.

What is 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

#### XML Simplifies Things

- XML simplifies data sharing
- XML simplifies data transport
- XML simplifies platform changes
- XML simplifies data availability

Many computer systems contain data in incompatible formats. Exchanging data between incompatible systems (or upgraded systems) is a time-consuming task for web developers. Large amounts of data must be converted, and incompatible data is often lost.

XML stores data in plain text format. This provides a software- and hardware-independent way of storing, transporting, and sharing data.

XML also makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.

With XML, data can be available to all kinds of "reading machines" like people, computers, voice machines, news feeds, etc.

## WEB PROGRAMMING

### XML Documents- XSL – XSLT-

With XSLT you can transform an XML document into HTML.

#### Displaying XML with XSLT

XSLT (eXtensible Stylesheet Language Transformations) is the recommended style sheet language for XML.

XSLT is far more sophisticated than CSS. With XSLT you can add/remove elements and attributes to or from the output file. You can also rearrange and sort elements, perform tests and make decisions about which elements to hide and display, and a lot more.

XSLT uses XPath to find information in an XML document.

#### XSLT Example

We will use the following XML document:

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

<food>
<name>Belgian Waffles</name>
<price>$5.95</price>
<description>Two of our famous Belgian Waffles with plenty of real maple
syrup</description>
<calories>650</calories>
</food>

<food>
<name>Strawberry Belgian Waffles</name>
<price>$7.95</price>
<description>Light Belgian waffles covered with strawberries and whipped
cream</description>
<calories>900</calories>
</food>

<food>
<name>Berry-Berry Belgian Waffles</name>
<price>$8.95</price>
<description>Light Belgian waffles covered with an assortment of fresh berries and whipped
cream</description>
<calories>900</calories>
</food>

<food>
```

## WEB PROGRAMMING

```
<name>French Toast</name>
<price>$4.50</price>
<description>Thick slices made from our homemade sourdough bread</description>
<calories>600</calories>
</food>

<food>
<name>Homestyle Breakfast</name>
<price>$6.95</price>
<description>Two eggs, bacon or sausage, toast, and our ever-popular hash browns</description>
<calories>950</calories>
</food>

</breakfast_menu>
```

Use XSLT to transform XML into HTML, before it is displayed in a browser:

### Web services

The Internet is the worldwide connectivity of hundreds of thousands of computers of various types that belong to multiple networks. On the World Wide Web, a web service is a standardized method for propagating messages between client and server applications. A web service is a software module that is intended to carry out a specific set of functions. Web services in cloud computing can be found and invoked over the network. The web service would be able to deliver functionality to the client that invoked the web service.

A web service is a set of open protocols and standards that allow data to be exchanged between different applications or systems. Web services can be used by software programs written in a variety of programming languages and running on a variety of platforms to exchange data via computer networks such as the Internet in a similar way to inter-process communication on a single computer.

Any software, application, or cloud technology that uses standardized web protocols (HTTP or HTTPS) to connect, interoperate, and exchange data messages – commonly XML (Extensible Markup Language) – across the internet is considered a web service. Web services have the advantage of allowing programs developed in different languages to connect with one another by exchanging data over a web service between clients and servers. A client invokes a web service by submitting an XML request, which the service responds with an XML response.

## WEB PROGRAMMING

### Functions of Web Services

- It's possible to access it via the internet or intranet networks.
- XML messaging protocol that is standardized.
- Operating system or programming language independent.
- Using the XML standard, it is self-describing.
- A simple location approach can be used to locate it.

#### *Components of Web Service*

XML and HTTP is the most fundamental web services platform. The following components are used by all typical web services:

### **SOAP (Simple Object Access Protocol)**

SOAP stands for “Simple Object Access Protocol.” It is a transport-independent messaging protocol. SOAP is built on sending XML data in the form of SOAP Messages. A document known as an XML document is attached to each message. Only the structure of the XML document, not the content, follows a pattern. The best thing about Web services and SOAP is that everything is sent through HTTP, the standard web protocol.

A root element known as the `<Envelope>` element is required in every SOAP document. In an XML document, the root element is the first element. The “envelope” is separated into two halves. The header comes first, followed by the body. The routing data, or information that directs the XML document to which client it should be sent to, is contained in the header. The real message will be in the body.

### **UDDI (Universal Description, Discovery, and Integration)**

UDDI is a standard for specifying, publishing and discovering a service provider’s online services. It provides a specification that aids in the hosting of data via web services. UDDI provides a repository where WSDL files can be hosted so that a client application can discover a WSDL file to learn about the various actions that a web service offers. As a result, the client application will have full access to the UDDI, which serves as a database for all WSDL files.

The UDDI registry will hold the required information for the online service, just like a telephone directory has the name, address, and phone number of a certain individual. So that a client application may figure out where it is.

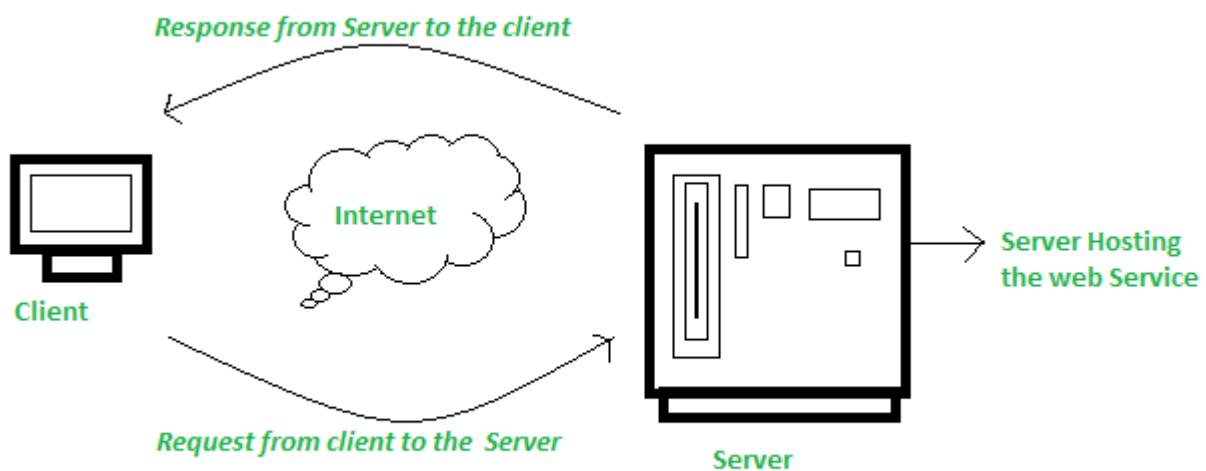
### **WSDL (Web Services Description Language)**

## WEB PROGRAMMING

If a web service can't be found, it can't be used. The client invoking the web service should be aware of the location of the web service. Second, the client application must understand what the web service does in order to invoke the correct web service. The WSDL, or Web services description language, is used to accomplish this. The WSDL file is another XML-based file that explains what the web service does to the client application. The client application will be able to understand where the web service is located and how to use it by using the WSDL document.

### **How Does Web Service Work?**

The diagram depicts a very simplified version of how a web service would function. The client would use requests to send a sequence of web service calls to a server that would host the actual web service.



Remote procedure calls are what are used to make these requests. Calls to methods hosted by the relevant web service are known as Remote Procedure Calls (RPC). Example: Flipkart offers a web service that displays prices for items offered on Flipkart.com. The front end or presentation layer can be written in .Net or Java, but the web service can be communicated using either programming language.

The data that is exchanged between the client and the server, which is XML, is the most important part of a web service design. XML (Extensible markup language) is a simple intermediate language that is understood by various programming languages. It is a counterpart to HTML. As a result, when programs communicate with one another, they do so using XML. This creates a common platform for applications written in different

## WEB PROGRAMMING

programming languages to communicate with one another. For transmitting XML data between applications, web services employ SOAP (Simple Object Access Protocol). The data is sent using standard HTTP. A SOAP message is data that is sent from the web service to the application. An XML document is all that is contained in a SOAP message. The client application that calls the web service can be created in any programming language because the content is written in XML.

### ***Features/Characteristics Of Web Service***

Web services have the following features:

**(a) XML Based:** The information representation and record transportation layers of a web service employ XML. There is no need for networking, operating system, or platform binding when using XML. At the middle level, web offering-based applications are highly interoperable.

**(b) Loosely Coupled:** A customer of an internet service provider isn't necessarily directly linked to that service provider. The user interface for a web service provider can change over time without impacting the user's ability to interact with the service provider. A strongly coupled system means that the patron's and server's decisions are inextricably linked, indicating that if one interface changes, the other should be updated as well. A loosely connected architecture makes software systems more manageable and allows for easier integration between different structures.

**(c) Capability to be Synchronous or Asynchronous:** Synchronicity refers to the client's connection to the function's execution. The client is blocked and the client has to wait for the service to complete its operation, before continuing in synchronous invocations. Asynchronous operations allow a client to invoke a task and then continue with other tasks. Asynchronous clients get their results later, but synchronous clients get their effect immediately when the service is completed. The ability to enable loosely linked systems requires asynchronous capabilities.

**(d) Coarse-Grained:** Object-oriented systems, such as Java, make their services available through individual methods. At the corporate level, a character technique is far too fine an operation to be useful. Building a Java application from the ground, necessitates the development of several fine-grained strategies, which are then combined into a rough-grained provider that is consumed by either a buyer or a service. Corporations should be coarse-grained, as should the interfaces they expose. Web services

## WEB PROGRAMMING

generation is an easy approach to define coarse-grained services that have access to enough commercial enterprise logic.

**(e) Supports Remote Procedural Call:** Consumers can use an XML-based protocol to call procedures, functions, and methods on remote objects utilizing web services. A web service must support the input and output framework exposed by remote systems. Enterprise-wide component development Over the last few years, JavaBeans (EJBs) and.NET Components have become more prevalent in architectural and enterprise deployments. A number of RPC techniques are used to allocate and access both technologies. A web function can support RPC by offering its own services, similar to those of a traditional role, or by translating incoming invocations into an EJB or.NET component invocation.

**(f) Supports Document Exchanges:** One of XML's most appealing features is its simple approach to communicating with data and complex entities. These records can be as simple as talking to a current address or as complex as talking to an entire book or a Request for Quotation. Web administrations facilitate the simple exchange of archives, which aids incorporate reconciliation.

The web benefit design can be seen in two ways: **(i)** The first step is to examine each web benefit on-screen character in detail. **(ii)** The second is to take a look at the rapidly growing web benefit convention stack.

### *Advantages Of Web Service*

Using web services has the following advantages:

**(a) Business Functions can be exposed over the Internet:** A web service is a controlled code component that delivers functionality to client applications or end-users. This capability can be accessed over the HTTP protocol, which means it can be accessed from anywhere on the internet. Because all apps are now accessible via the internet, Web services have become increasingly valuable. Because all apps are now accessible via the internet, Web services have become increasingly valuable. That is to say, the web service can be located anywhere on the internet and provide the required functionality.

**(b) Interoperability:** Web administrations allow diverse apps to communicate with one another and exchange information and services. Different apps can also make use of web services. A .NET application, for example, can communicate with Java web administrations and vice versa. To make the application stage and innovation self-contained, web administrations are used.

## WEB PROGRAMMING

- (c) Communication with Low Cost:** Because web services employ the SOAP over HTTP protocol, you can use your existing low-cost internet connection to implement them. Web services can be developed using additional dependable transport protocols, such as FTP, in addition to SOAP over HTTP.
- (d) A Standard Protocol that Everyone Understands:** Web services communicate via a defined industry protocol. In the web services protocol stack, all four layers (Service Transport, XML Messaging, Service Description, and Service Discovery) use well-defined protocols.
- (e) Reusability:** A single web service can be used simultaneously by several client applications.

## Beyond the Syllabus

- Introduction to PHP
- Handling File Uploads.
- Connecting to database (MySQL as reference)
- Introduction to JSP: The Anatomy of a JSP Page
- Connecting to database in JSP.

## ASSIGNMENT QUESTIONS

### Assignment-1

1. Differentiate client side and server side scripts?
2. Explain about Windows, document and browser Objects
3. Explain built in functions in JavaScript with program

### Assignment-2

1. Define an exception. Give example
2. What is Package? Give a detailed description for creation of packages in Java with an example
3. Write a Java Program to sort an array of ‘n’ numbers in ascending order.

## WEB PROGRAMMING

### Assignment-3

1. Describe JDBC with its architecture.
2. Describe the Callable Statement in JDBC?
3. Mention the purpose of using InetAddress class.

### Assignment-4

1. What are HttpServletRequest and HttpServletResponse?
2. Classify the different types of directive in JSP?
3. What are AWT controls?

### Assignment-5

1. What is the purpose of XSLT?
2. What is UDDI?
3. Illustrate the encoding of array in SOAP.

## WEB PROGRAMMING

### 4. Short Long Answer Question with Blooms Taxonomy Levels

#### Long Answer Questions-

SUBJECT : WEB PROGRAMMING

YEAR: III Year

UNIT I - SCRIPTING			
PART – A			
Q.No	Questions	BT Level	Competence
1.	List the two differences between HTML and XHTML with respect to elements.	BTL 1	Remembering
2.	Write the syntax to display the following statement “I am learning <b>Web Programming</b> ”	BTL 2	Understanding
3.	Create a HTML code to display an image.	BTL 6	Creating
4.	How will you create a password field in a HTML form?	BTL 3	Applying
5.	Differentiate client side and server side scripts?	BTL 2	Understanding
6.	Create two rows of horizontal frames using HTML frames.	BTL 6	Creating
7.	What is a JavaScript’s statement? Give example.	BTL 1	Remembering
8.	List out the objects used in JavaScript with its purpose.	BTL 1	Remembering
9.	What is the difference between undefined value and null value?	BTL 1	Remembering

## WEB PROGRAMMING

10.	What is the significance of, and reason for, wrapping the entire content of a Javascript source file in a function block?	BTL 3	Applying
11.	How a scripting language differs from HTML?	BTL 3	Applying
12.	List and explain any four HTML intrinsic event attributes.	BTL 1	Remembering

**WEB PROGRAMMING**

13.	Explain array creation in JavaScript with example.	BTL 4	Analyzing
14.	What are style sheets? List the ways of including style information in a HTML document.	BTL 2	Understanding
15.	Discuss the core syntax of CSS.	BTL 2	Understanding
16.	Analyze some advantages of using cascading style sheets (CSS).	BTL 4	Analyzing
17.	Explain with an example for inline style sheet.	BTL 5	Evaluating
18.	List out the limitations of CSS?	BTL 1	Remembering
19.	What is meant by canvas in HTML?	BTL 5	Evaluating
20.	Explain how external style sheet is useful in web page design?	BTL 4	Analyzing

**PART – B**

1.	(i) List and explain any four HTML elements in detail. (7) (ii) Classify the types of lists supported by HTML and describe them in detail.(6)	BTL1	Remembering
2.	Briefly discuss about (i) HTML frames. (6) (ii) Table tags. (7)	BTL 2	Understanding
3.	(i) Write a JavaScript program to count the number of unique alphabets present in a given string. (7) (ii) Discuss about the different types of Cascading style sheets. (6)	BTL 2	Understanding
4.	Create a website using HTML for a “Library management system”. Your website should have a home page which helps the user to navigate to various pages like student membership, books catalog, transactions and search pages. (13)	BTL 6	Creating
5.	(i) Describe how do you use JavaScript for form validation? Develop a complete application that would include functions to validate the user data.(8) (ii) Write short notes on JavaScript built-in objects.(5)	BTL1	Remembering
6.	Explain objects and arrays in JavaScript with suitable example.(13)	BTL 4	Analyzing
7.	(i) Describe the difference between document and window objects with suitable examples.(7) (ii) Develop a web page for student data form using HTML and validate the registration form.(6)	BTL 3	Applying

**WEB PROGRAMMING**

8.	(i) Write a JavaScript program to delete the roll no property from the following object. Also print the object before and after deleting the property. Sample object: var student = { name: "Santhosh Ravy", class: "VI", rollno: 29}; (7) (ii) Write a JavaScript program to search a date (MM/DD/YYYY) within a string.(6)	BTL 5	Evaluating
9.	(i) State and explain the types of statements in JavaScript. (7) (ii) Explain how functions can be written in JavaScript with an example.(6)	BTL 4	Analyzing
10.	Explain in detail about CSS3 and their types with suitable example program. (13)	BTL4	Analyzing
11.	(i) List and explain in detail the various selector strings. (7) (ii) Discuss the features of cascading style sheets. (6)	BTL1	Remembering
12.	Apply CSS to a web page with the following requirements (i) Add a background image of a submarine (4) (ii) Set a color to the span elements (different color for each class) (4) (iii) Set a line spacing between the lines (2) (iv) Set letter spacing between the letters in each span of type instruction(3)	BTL 3	Applying
13.	(i) Describe the CSS box model in detail. (7) (ii) List and explain in details about any four types of selector strings.(6)	BTL 1	Remembering
14.	(i) Express a CSS rule which adds background images and indentation.(7) (ii) Define external style sheet with an example.(6)	BTL2	Understanding

**PART – C**

1.	i) Outline the different types of cascading style sheets. (8) ii) Compare the features of client side scripting languages and server side scripting languages. (7)	BTL5	Evaluating
2.	Create a website for a online super market. Your website should have a home page which helps the user to navigate to various pages. Every web page in the website give a detailed description of different items present in the super market. Make the website user friendly by adding relevant images and other formatting options. The last web page should present a feedback form for the customer. (15)	BTL6	Creating
3.	(i) Differentiate client side and server side scripting with suitable examples.(8) (ii) Develop a web page for student data form using HTML and validate the registration form.(7)	BTL5	Evaluating

<b>WEB PROGRAMMING</b>			
4.	i) Write JavaScript to find sum of first „n“ even number and display the result. Read the value of n from user. (8) ii) Explain in detail about the CSS.(7)	BTL6	Creating

## **UNIT II - JAVA**

Introduction to object oriented programming-Features of Java – Data types, variables and arrays – Operators – Control statements – Classes and Methods – Inheritance. Packages and Interfaces –Exception Handling – Multithreaded Programming – Input/output – Files – Utility Classes – String Handling.

### **PART - A**

<b>Q.N</b>	<b>Questions</b>	<b>BT Level</b>	<b>Competence</b>
1.	Define abstract classes in Java.	BTL1	Remembering
2.	What is inheritance?	BTL1	Remembering
3.	What gives Java its „write once and run anywhere“ nature?	BTL1	Remembering
4.	Write the syntax for declaring a two dimensional array in Java.	BTL3	Applying
5.	State the purpose of encapsulation.	BTL2	Understanding
6.	Write a Java code to find the Fibonacci series of a given number.	BTL3	Applying
7.	If I don't provide any arguments on the command line, then the string array of main method will be empty or null? Justify.	BTL5	Evaluating
8.	Differentiate Class and Interface in Java	BTL2	Understanding
9.	What is the use of StringBuffer class in Java?	BTL1	Remembering
10.	Define an exception. Give example.	BTL1	Remembering
11.	List some important input and output stream classes.	BTL1	Remembering
12.	What are the two ways of creating a thread?	BTL2	Understanding
13.	What is an exception? Give example.	BTL2	Understanding
14.	What is a Java package and how it is used?	BTL3	Applying
15.	Compare between the File and Random Access File classes?	BTL4	Analyzing
16.	What is the importance of == and equals () method with respect to String object?	BTL4	Analyzing
17.	Mention the purpose of the keyword „final“	BTL4	Analyzing
18.	Why do we need run () and start () method both? Can we achieve it with only run method?	BTL5	Evaluating
19.	Write a Java code to check if the given string is palindrome or not.	BTL6	Creating
20.	Write a multithreaded program that joins two threads.	BTL6	Creating

### **PART – B**

1.	Outline on the various object oriented concepts with necessary illustrations.(13)	BTL2	Understanding
----	-----------------------------------------------------------------------------------	------	---------------

**WEB PROGRAMMING**

2.	(i) State the use of constructor and finalize () method in Java using a programming example. Show how garbage collection is achieved here. (7) (ii) Can Java directly support multiple inheritance. Illustrate your answer with an example Java program(6)	BTL5	Evaluating
3.	(i) What does it mean that a method or class is abstract? Can we make an instance of an abstract class? Explain it with example. (6) (ii) What is polymorphism in Java? Explain how polymorphism is supported in Java.(7)	BTL1	Remembering
4.	Elaborate the inheritance and their types in Java with example coding. (13)	BTL1	Remembering
5.	(i) Discuss constructor in java? Why constructor does not have return type in java? Explain it with proper example.(7) (ii) Why so we need static members and how to access them? Explain it with clear example(6)	BTL2	Understanding
6.	(i) Illustrate with the help of a program how object oriented programming overcomes the shortcomings of procedure oriented programming. (7) (ii) Explain inheritance? How will you call parameterized constructor and over ridded method from parent class in sub class?(6)	BTL4	Analyzing
7.	(i) What is Package? Give a detailed description for creation of packages in Java with an example (7) (ii) List the different contexts in which the final keyword is used in Java program.(6)	BTL1	Remembering
8.	(i) What is meant by stream? What are the types of streams and classes? Explain in detail.(7) (ii) List and discuss the role of various Buffer classes used in Java programming.(6)	BTL1	Remembering
9.	How to declare and initialize a string in java and also discuss the different string handling functions with suitable examples. (13)	BTL2	Understanding
10.	(i) Write a Java program that collects the input as a decimal number of integer type and converts it into a String of equivalent hexadecimal number.(7) (ii) Write a Java program that arranges the given set of strings in alphabetical order. Supply the strings through the command line.(6)	BTL3	Applying
11.	(i) Briefly explain interface? Write a Java program to illustrate the use of an interface. (7) (ii) Explain about packages. Give an example program which uses packages.(6)	BTL3	Applying
12.	With an example explain about multithreaded programming concept in Java. (13)	BTL4	Analyzing

**WEB PROGRAMMING**

13.	Explain how exception handling mechanism has been implemented in Java. Also explain various types of checked and unchecked exceptions that may arise in Java with suitable examples. (13)	BTL4	Analyzing
14.	Write a Java program that acts as a web server and transfer the HTML file requested by the browser client. (13)	BTL6	Creating

**PART – C**

1.	i) Write a Java Program to sort an array of „n“ numbers in ascending order.(7) ii) Write a multithreaded Java Program to generate even, Odd and prime numbers.(8)	BTL6	Creating
2.	i) Explain exceptions handling in Java with examples.(7) ii) Explain string handling in Java with examples.(8)	BTL3	Applying
3.	i) What is Package? Give detailed descriptions for creation of packages in Java with example.(9) ii) List the different applications of Utility Classes in Java.(6)	BTL6	Creating
4.	Explain how exceptions are handled in Java. Also explain various categories of exceptions in Java with suitable examples(15)	BTL5	Evaluating

**WEB PROGRAMMING****UNIT III - JDBC**

JDBC Overview – JDBC implementation – Connection class – Statements – Catching Database Results, handling database Queries. Networking– InetAddress class – URL class- TCP sockets – UDP sockets, Java Beans –RMI.

**PART - A**

<b>Q.No</b>	<b>Questions</b>	<b>BT Level</b>	<b>Competence</b>
1.	Describe JDBC with its architecture.	BTL1	Remembering
2.	What are the various database connectivity supported in Java?	BTL1	Remembering
3.	What are the main steps in java to make JDBC connectivity?	BTL1	Remembering
4.	What is the difference between Statement and Prepared Statement interface?	BTL1	Remembering
5.	How JDBC API does helps us in achieving loose coupling between Java Program and JDBC Drivers API?	BTL4	Analyzing
6.	How can you create JDBC statements? Illustrate	BTL3	Applying
7.	Describe the Callable Statement in JDBC?	BTL2	Understanding
8.	Create the connection class using ODBC connectivity?	BTL6	Creating
9.	Mention the purpose of using InetAddress class.	BTL1	Remembering
10.	Mention the purpose of using URL class.	BTL1	Remembering
11.	Can the InetAddress class functionality to detect the IP Addresses, be handled using URL class? If yes, Explain?	BTL4	Analyzing
12.	Create an InetAddress class to print the address and names of local machine.	BTL6	Creating
13.	Summarize the advantages of Java networking	BTL5	Evaluating
14.	Classify the various server socket constructors.	BTL3	Applying
15.	Define a socket. Classify sockets.	BTL4	Analyzing
16.	What is JavaBean? Write the properties of JavaBean.	BTL2	Understanding
17.	How to access the java bean class? Give example.	BTL3	Applying
18.	Give the need of Registry objects.	BTL2	Understanding
19.	Distinguish stub and skeleton in RMI.	BTL2	Understanding
20.	Summarize the steps involved in RMI with a neat sketch.	BTL5	Evaluating

**PART - B**

1.	(i) Describe how JDBC works. (6) (ii) Show the various JDBC driver types in detail. (7)	BTL 1	Remembering
----	--------------------------------------------------------------------------------------------	-------	-------------

**WEB PROGRAMMING**

2.	(i) What is batch processing and how to perform batch processing in JDBC? (7) (ii) Write a short note on catching database result. (6)	BTL1	Remembering
3.	Write a java program that queries for customer information from a database, the program must also facilitates insertion, deletion and updation of customer details into the database. (13)	BTL5	Evaluating
4.	(i) Explain the steps involved to create JDBC connectivity. List the advantages of JDBC. (7) (ii) Explain the various methods used in ResultSet interface.(6)	BTL2	Understanding
5.	(i) Discuss in detail about Batch Updates with an example. (7) (ii) Illustrate the concept of updatable resultsets.(6)	BTL2	Understanding
6.	(i) Illustrate the connection interface with its methods. (7) (ii) Illustrate in detail about the statement interface and its methods.(6)	BTL1	Remembering
7.	(i) List the steps involved in developing Java Bean (5) (ii)Describe in detail about properties presented in java bean and its uses .(8)	BTL1	Remembering
8.	(i) What is a Java Bean? Write down the properties of a Java Bean in detail. (7) (ii) Write down the steps involved in establishing client-server communication using UDP. (6)	BTL4	Analyzing
9.	(i) Discuss in detail about the URL class. (7) (ii) List out the classes and interfaces used in java.net package.(6)	BTL2	Understanding
10.	Write an RMI program in which the remote method computes the factorial of the number given by the client. (13)	BTL3	Applying
11.	Explain with an example the process of developing a client-server program using RMI in Java. (13)	BTL4	Analyzing
12.	Explain the following with suitable example. (i) How to implement Java Beans? (6) (ii) Differentiate TCP sockets and UDP sockets. (7)	BTL3	Applying
13.	Explain the architecture of RMI in detail with neat diagram. (13)	BTL4	Analyzing

## WEB PROGRAMMING

14.	<p>(i) Create a TCP server which accepts data from a TCP client and stores it in a file. The data should be combined and sent back to the client only after receiving twice from the client. (7)</p> <p>(ii) Write code to implement a telephone directory application where the administrator on entering his appropriate user id and password (requires validation of the form) should be able to add, delete or modify the records in the database use Javascript and JDBC (MySQL or ORACLE) (6)</p>	BTL6	Creating
-----	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------	----------

**WEB PROGRAMMING****PART - C**

1.	Explain with an example how a Java application can access a database using JDBC. (15)	BTL5	Evaluating
2.	Implement a remote phone book server that maintains a file of names and phone numbers and a client allows the user to scroll through the file using RMI concept. (15)	BTL5	Evaluating
3.	Write a Java Program those queries for student information from a database. The program must also facilities insertion, deletion, and updation of student details into the database.(15)	BTL6	Creating
4.	Explain RMI. Create an RMI program that generates „n“ prime numbers. (15)	BTL6	Creating

**UNIT IV - APPLETS**

Java applets- Life cycle of an applet – Adding images to an applet – Adding sound to an applet. Passing parameters to an applet. Event Handling. Introducing AWT: Working with Windows Graphics and Text. Using AWT Controls, Layout Managers and Menus. Servlet – life cycle of a servlet. The Servlet API, Handling HTTP Request and Response, using Cookies, Session Tracking. Introduction to JSP

**PART - A**

Q.No	Questions	BT Level	Competence
1.	Define a Java Applet.	BTL1	Remembering
2.	Write about the lifecycle of an applet.	BTL1	Remembering
3.	Discuss the two types of methods to run an applet.	BTL2	Understanding
4.	Formulate a code to pass parameter to an applet?	BTL6	Creating
5.	How to implement an applet into a web page using applet tag?	BTL4	Analyzing
6.	What are the advantages of event delegation model?	BTL2	Understanding
7.	What are AWT controls?	BTL1	Remembering
8.	Code a graphics method in Java to draw the string “Hello World” from the coordinates (100,200).	BTL5	Evaluating
9.	Where do you prefer AWT controller? Give an example.	BTL3	Applying
10.	List any four layout managers supported in Java.	BTL1	Remembering

**WEB PROGRAMMING**

11.	Illustrate the use of frames in layout management.	BTL3	Applying
12.	What are the life cycle methods of servlets?	BTL2	Understanding
13.	What are HttpServletRequest and HttpServletResponse?	BTL2	Understanding
14.	How sessions are handled in servlets?	BTL4	Analyzing
15.	List the use of cookies.	BTL1	Remembering
16.	When are cookies created? How long does a cookie last?	BTL5	Evaluating
17.	What is JSP? Write two main usages of it.	BTL1	Remembering
18.	Classify the different types of directive in JSP?	BTL3	Applying
19.	Compare JSP and servlet.	BTL4	Analyzing
20.	Formulate a JSP with simple java code to display a welcome message.	BTL6	Creating

**PART – B**

1.	(i) Describe in detail about the AWT Event classes. (7) (ii) Describe about the various Event Listeners(6)	BTL1	Remembering
2.	(i) Explain the lifecycle of an applet. How will you pass parameters to an applet? (6) (ii) Write a program using applet to print the area of various shapes rectangle, circle, and square using the concept of virtual function. (7)	BTL5	Evaluating
3.	(i) With an example explain how parameters are passed to an applet. (7) (ii) Explain how Graphics class can be used in an applet. (6)	BTL4	Analyzing
4.	Discuss in detail about AWT event hierarchy.(13)	BTL2	Understanding
5.	(i) What is the function of layout manager? Describe in detail about the different layout in Java GUI. (7) (ii) With an example, define working with text in an applet. (6)	BTL1	Remembering
6.	(i) Explain the various Layout Managers in Java with neat diagram.(7) (ii) Write the Advantage of JSP over Servlets.(6)	BTL4	Analyzing
7.	(i) Discuss about servlet life cycle with example. (5) (ii) Discuss database connectivity with Servlet to display student marks.(8)	BTL2	Understanding
8.	Enlist the methods that can be used to handle the HTTP request. (13)	BTL1	Remembering
9.	Discuss in detail about the HttpServlet Class and its interface. (13)	BTL2	Understanding

## WEB PROGRAMMING

10.	(i) What is HTTP? What are HTTP request and response? Give the limitations of HTTP. (7) (ii) Explain the lifecycle of servlet. (6)	BTL1	Remembering
-----	---------------------------------------------------------------------------------------------------------------------------------------	------	-------------

**WEB PROGRAMMING**

11.	Explain about session tracking mechanism of a servlet. (13)	BTL4	Analyzing
12.	(i) Develop a client server JSP program to find simple interest and display the result in the client. (8) (ii) Define the JSP tag libraries. (5)	BTL3	Applying
13.	(i) Develop a JSP to accept user's first name and then welcome the user by name. (7) (ii) Identify the implicit objects that are used in JSP.(6)	BTL3	Applying
14.	(i) In an Applet, create a frame with two text fields and three buttons (Cut, Copy & Paste). Data entered in the first text field should respond, according to the buttons clicked. (8) (ii) Write a JavaServlet program to implement Cookies using getCookies(), getName() and getValue() methods. (5)	BTL6	Creating

**PART – C**

1.	Explain with diagrammatic illustration the following: i) Life cycle of a servlet.(7) ii) Life cycle of an applet.(8)	BTL5	Evaluating
2.	i) How do add images and sound files to an applet? Explain with an example.(8) ii) Highlight the features of JSP.(7)	BTL4	Analyzing
3.	What is Applet? Explain how the parameters are passed to an Applet with suitable example. (15)	BTL5	Evaluating
4.	Using AWT create a frame which contains four text fields name, age, sex and qualification layout using the flow layout manager. Run the program and give the values of all text fields in the command line. Initially all the values of text field should be blank. On clicking the click button all the text fields should contain the command line inputs. (15)	BTL6	Creating

**WEB PROGRAMMING****UNIT V - XML AND WEB SERVICES**

Xml – Introduction-Form Navigation-XML Documents- XSL – XSLT- Web services-UDDI-WSDL-Java web services – Web resources.

**PART - A**

<b>Q.No</b>	<b>Questions</b>	<b>BT Level</b>	<b>Competence</b>
1.	What is a well formed XML document?	BTL1	Remembering
2.	Give the rules for well-formed documents in XML.	BTL1	Remembering
3.	State the advantages of XML.	BTL1	Remembering
4.	Differentiate XML Schema and DTD.	BTL4	Analyzing
5.	Explain the purpose of XML schema.	BTL5	Evaluating
6.	What is XSLT?	BTL1	Remembering
7.	What is the purpose of XSLT?	BTL4	Analyzing
8.	What are the elements of WSDL?	BTL1	Remembering
9.	What is UDDI?	BTL1	Remembering
10.	Give an example of a web service registry and its function.	BTL2	Understanding
11.	What are the various building blocks in an XML?	BTL2	Understanding
12.	Summarize the differences between HTML and XML.	BTL2	Understanding
13.	Illustrate the encoding of array in SOAP.	BTL3	Applying
14.	Show how UDDI is utilized in web service.	BTL3	Applying
15.	What is Java Web Service? Draw the architecture of Web Service.	BTL3	Applying
16.	What are the uses of WSDL and XLang in web service interaction?	BTL2	Understanding
17.	Analyze the need for SOAP in web services.	BTL4	Analyzing
18.	State the significance of a WSDL document.	BTL5	Evaluating
19.	Formulate the basic concepts behind JAX-RPC technology.	BTL6	Creating
20.	Create a XML program to display student details.	BTL6	Creating

**PART-B**

1.	(i) List and explain the XML syntax rules in detail. (7) (ii) Explain how a XML document can be displayed on a browser.(6)	BTL1	Remembering
2.	Give a brief note on XSL and XSLT. (13)	BTL1	Remembering
3.	(i) Describe in detail about XSL.(7) (ii) What languages are used to represent data in web? Explain any two of them.(6)	BTL1	Remembering
4.	(i) With a simple example illustrate the steps to create a web service. (6) (ii) Explain the SOAP elements in detail.(7)	BTL1	Remembering

## WEB PROGRAMMING

5.	Summarize the role of XML schema in building web services in detail.(13)	BTL2	Understanding
----	--------------------------------------------------------------------------	------	---------------

WEB PROGRAMMING			
6.	(i) Briefly discuss how data types are represented in XML schema.  (ii) Briefly discuss how SOAP encodes struct data and arrays(6)	BTL2	Understanding
7.	(i) Describe with an example about the various web service technologies. (6)  (ii) Discuss JAX-RPC concept with suitable example.(7)	BTL2	Understanding
8.	(i) Explain WSDL structure and its elements. (7)  (ii) Develop a Java Web Service that would do arithmetic operations.(6)	BTL3	Applying
9.	Briefly discuss about XML and DTD. Write a DTD for employee details including employee name (first name and last name), employee ID, Date of Birth (month, date and year) and address (city and state).(13)	BTL3	Applying
10.	(i) Explain with an example about the various Web Service technologies. (7)  (ii) Write an example for default XML namespace and create the XSLT with font, color, size and bgcolor. (6)	BTL4	Analyzing
11.	Explain the following with suitable example. (i) UDDI (6) (ii) WSDL (7)	BTL4	Analyzing
12.	(i) Analogous to the calculator service, implement a simple SOAP Web Service. (6)  (ii) Write short notes on Java Web Services and its benefits. (7)	BTL4	Analyzing
13.	Explain about creating and using of a Java web services. (13)	BTL5	Evaluating
14.	Design a railway reservation system using UDDI and WSDL for the following case study. Railway could register their services into an UDDI directory for checking the train rate and reservation. Travel agencies could then search the UDDI directory to find the railway reservation interface for ticket booking. (13)	BTL6	Creating
PART-C			
1.	i) Create a CD catalog (title, artist, country, price, year) using XML. (10)  ii) Explain in detail about XSL. (5)	BTL5	Evaluating
2.	Develop XML document that will hold player (like Cricket) collection with field for player-name, age, batting-average and highest-score. Write suitable document type definition and schema for the XML. (15)	BTL6	Creating
3.	Write the case study of discovery and analysis of web usage patterns and explain the current trends. (15)	BTL6	Creating
4.	i) Explain briefly about UDDI and SOAP. (9)  ii) Implement a simple SOAP Web service analogous to the Calculator service. (6)	BTL5	Evaluating

**OBJECTIVE TYPE UNIT WISE QUESTIONS**  
**UNIT-1**

Which attribute specifies a unique alphanumeric identifier to be associated with an element?

- a) class
- b) id
- c) article
- d) html

Answer: b

The \_\_\_\_\_ attribute specifies an inline style associated with an element, which determines the rendering of the affected element.

- a) dir
- b) style
- c) class
- d) article

Answer: b

Which attribute is used to provide an advisory text about an element or its contents?

- a) tooltip
- b) dir
- c) title
- d) head

Answer: c

The \_\_\_\_\_ attribute sets the text direction as related to the lang attribute.

- a) lang
- b) sub
- c) dir
- d) ds

Answer: c

HTML is what type of language ?

- A.Scripting Language
- B.Markup Language
  
- C.Programming Language
- D.Network Protocol

Answer : B

HTML uses

- A. User defined tags
- B. Pre-specified tags
  
- C. Fixed tags defined by the language
- D. Tags only for linking

Answer : C

Apart from <b> tag, what other tag makes text bold ?

- A. <fat>
- B. <strong>
  
- C. <black>
- D. <emp>

Answer : B

Why so JavaScript and Java have similar name?

**MRITS-IT**

- A. JavaScript is a stripped-down version of Java
- B. JavaScript's syntax is loosely based on Java's
- C. They both originated on the island of Java
- D. None of the above

Ans: B

2. When a user views a page containing a JavaScript program, which machine actually executes the script?

- A. The User's machine running a Web browser
- B. The Web server
- C. A central machine deep within Netscape's corporate offices
- D. None of the above

Ans: A

3. \_\_\_\_\_ JavaScript is also called client-side JavaScript.

- A. Microsoft
- B. Navigator
- C. LiveWire
- D. Native

Ans: B

4. \_\_\_\_\_ JavaScript is also called server-side JavaScript.

- A. Microsoft
- B. Navigator
- C. LiveWire
- D. Native

Ans: C

5. What are variables used for in JavaScript Programs?

- A. Storing numbers, dates, or other values
- B. Varying randomly
- C. Causing high-school algebra flashbacks
- D. None of the above

Ans: A

6. \_\_\_\_\_ JavaScript statements embedded in an HTML page can respond to user events such as mouse-clicks, form input, and page navigation.

- A. Client-side
- B. Server-side
- C. Local
- D. Native

Ans: A

7. What should appear at the very end of your JavaScript?

The <script LANGUAGE="JavaScript">tag

- A. The </script>
- B. The <script>
- C. The END statement
- D. None of the above

Ans: A

8. Which of the following can't be done with client-side JavaScript?

- A. Validating a form
- B. Sending a form's contents by email
- C. Storing the form's contents to a database file on the server

**MRITS-IT**

D. None of the above

Ans: C

9. Which of the following are capabilities of functions in JavaScript?

- A. Return a value
- B. Accept parameters and Return a value
- C. Accept parameters
- D. None of the above

Ans: C

10. Which of the following is not a valid JavaScript variable name?

- A. 2names
- B. \_first\_and\_last\_names
- C. FirstAndLast
- D. None of the above

Ans: A

11. \_\_\_\_\_ tag is an extension to HTML that can enclose any number of JavaScript statements.

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

Ans: A

12. How does JavaScript store dates in a date object?

- A. The number of milliseconds since January 1st, 1970
- B. The number of days since January 1st, 1900
- C. The number of seconds since Netscape's public stock offering.
- D. None of the above

Ans: A

13. Which of the following attribute can hold the JavaScript version?

- A. LANGUAGE
- B. SCRIPT
- C. VERSION
- D. None of the above

Ans: A

14. 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")

Ans: C

15. Which of the following way can be used to indicate the LANGUAGE attribute?

- A. <LANGUAGE="JavaScriptVersion">
- B. <SCRIPT LANGUAGE="JavaScriptVersion">
- C. <SCRIPT LANGUAGE="JavaScriptVersion"> JavaScript statements...</SCRIPT>
- D. <SCRIPT LANGUAGE="JavaScriptVersion"!> JavaScript statements...</SCRIPT>

Ans: C

16. Inside which HTML element do we put the JavaScript?

- A. <js>

**MRITS-IT**

B. <scripting>

C. <script>

D. <javascript>

Ans: C

17. 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

Ans: C

18. Which types of image maps can be used with JavaScript?

A. Server-side image maps

B. Client-side image maps

C. Server-side image maps and Client-side image maps

D. None of the above

Ans: B

19. Which of the following navigator object properties is the same in both Netscape and IE?

A. navigator.appCodeName

B. navigator.appName

C. navigator.appVersion

D. None of the above

Ans: A

20. 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"

Ans: C

21. What does the <noscript> tag do?

A. Enclose text to be displayed by non-JavaScript browsers.

B. Prevents scripts on the page from executing.

C. Describes certain low-budget movies.

D. None of the above

Ans: A

22. If para1 is the DOM object for a paragraph, what is the correct syntax to change the text within the paragraph?

A. "New Text"?

B. para1.value="New Text";

C. para1.firstChild.nodeValue= "New Text";

D. para1.nodeValue="New Text";

Ans: B

23. JavaScript entities start with \_\_\_\_\_ and end with \_\_\_\_\_.

A. Semicolon, colon

B. Semicolon, Ampersand

C. Ampersand, colon

D. Ampersand, semicolon

Ans: D

**MRITS-IT**

24. 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-oriented scripting language.

Ans: D

25. Choose the server-side JavaScript object?

- A. FileUpLoad
- B. Function
- C. File
- D. Date

Ans: C

## **UNIT-2**

Who invented Java Programming?

- a) Guido van Rossum
- b) James Gosling
- c) Dennis Ritchie
- d) Bjarne Stroustrup

Answer: b

Which statement is true about Java?

- a) Java is a sequence-dependent programming language
- b) Java is a code dependent programming language
- c) Java is a platform-dependent programming language
- d) Java is a platform independent programming language

Answer: d

Which component is used to compile, debug and execute the java programs?

- a) JRE
- b) JIT
- c) JDK
- d) JVM

Answer: c

Which one of the following is not a Java feature?

- a) Object-oriented
- b) Use of pointers
- c) Portable
- d) Dynamic and Extensible

Answer: b

Which of these cannot be used for a variable name in Java?

- a) identifier & keyword
- b) identifier
- c) keyword
- d) none of the mentioned

Answer: c

What is the extension of java code files?

- a) .js
- b) .txt
- c) .class
- d) .java

Answer: d

**MRITS-IT**

What will be the output of the following Java code?

```
1. class increment {
2. public static void main(String args[])
3. {
4. int g = 3;
5. System.out.print(++g * 8);
6. }
7. }
```

- a) 32
- b) 33
- c) 24
- d) 25

Answer: a

What will be the output of the following Java code?

```
1. class output {
2. public static void main(String args[])
3. {
4. double a, b,c;
5. a = 3.0/0;
6. b = 0/4.0;
7. c=0/0.0;
8.
9. System.out.println(a);
10. System.out.println(b);
11. System.out.println(c);
12. }
13. }
```

- a) NaN
- b) Infinity
- c) 0.0
- d) all of the mentioned

Answer: a

Which of the following package is used for text formatting in Java programming language?

- a) java.io
- b) java.awt.text
- c) java.awt
- d) java.text

Answer: d

Which of the following is not a segment of memory in java?

- a) Code Segment
- b) Register Segment
- c) Stack Segment
- d) Heap Segment

Answer: b

What is the extension of compiled java classes?

- a) .txt
- b) .js
- c) .class

**MRITS-IT**

d) .java

Answer: c

Which tag is used with JavaScript?

a.<canvas>

b.<table>

c.<article>

d.<footer>

Answer: a

Who is known as father of Java Programming Language?

A.James Gosling

B.M. P Java

C.Charles Babbage

D.Blaiz Pascal

Answer : A

In java control statements break, continue, return, try-catch-finally and assert belongs to?

A.Selection statements

B.Loop Statements

C.Transfer statements

D.Pause Statement

Answer : C

Which of the following option leads to the portability and security of Java?

a.Bytocode is executed by JVM

b.The applet makes the Java code secure and portable

c.Use of exception handling

d.Dynamic binding between objects

**Answer:** (a)

Which of the following is not a Java features?

a. Dynamic

b. Architecture Neutral

c. Use of pointers

d. Object-oriented

**Answer:** (c)

\_\_\_\_\_ is used to find and fix bugs in the Java programs.

a. JVM

b. JRE

c. JDK

d. JDB

**MRITS-IT**

**Answer:** (d)

Which of the following is a valid declaration of a char?

- a. char ch = '\utea';
- b. char ca = 'tea';
- c. char cr = '\u0223';
- d. char cc = '\itea';

**Answer:** (a)

What is the return type of the hashCode() method in the Object class?

- a. Object
- b. int
- c. long
- d. void

**Answer:** (b)

Which of the following is a valid long literal?

- a. ABH8097
- b. L990023
- c. 904423
- d. 0xnf029L

**Answer:** (d)

What does the expression float a = 35 / 0 return?

- a. 0
- b. Not a Number
- c. Infinity
- d. Run time exception

**Answer:** (c)

Which of the following for loop declaration is not valid?

- a. for ( int i = 99; i >= 0; i / 9 )
- b. for ( int i = 7; i <= 77; i += 7 )
- c. for ( int i = 20; i >= 2; - -i )

**MRITS-IT**

d. for ( int i = 2; i <= 20; i = 2\* i )

**Answer:** (a)

Which method of the Class.class is used to determine the name of a class represented by the class object as a String?

- a. getClass()
- b. intern()
- c. getName()
- d. toString()

**Answer:** (c)

In which process, a local variable has the same name as one of the instance variables?

- a. Serialization
- b. Variable Shadowing
- c. Abstraction
- d. Multi-threading

**Answer:** (b)

Which package contains the Random class?

- a. java.util package
- b. java.lang package
- c. java.awt package
- d. java.io package

**Answer:** (a)

An interface with no fields or methods is known as a \_\_\_\_\_.

- a. Runnable Interface
- b. Marker Interface
- c. Abstract Interface
- d. CharSequence Interface

Answer (b)

Which of these classes are the direct subclasses of the **Throwable** class?

- a. RuntimeException and Error class
- b. Exception and VirtualMachineError class
- c. Error and Exception class

**MRITS-IT**

- d. IOException and VirtualMachineError class

**Answer:** (c)

What do you mean by *chained exceptions* in Java?

- a. Exceptions occurred by the VirtualMachineError
- b. An exception caused by other exceptions
- c. Exceptions occur in chains with discarding the debugging information
- d. None of the above

**Answer:** (b)

In which memory a String is stored, when we create a string using **new** operator?

- a. Stack
- b. String memory
- c. Heap memory
- d. Random storage space

**Answer:** (c)

Which keyword is used for accessing the features of a package?

- a. package
- b. import
- c. extends
- d. export

**Answer:** (b)

In java, jar stands for\_\_\_\_\_.

- a. Java Archive Runner
- b. Java Application Resource
- c. Java Application Runner
- d. None of the above

**Answer:** (d)

## UNIT-3

What are the major components of the JDBC?

- a. DriverManager, Driver, Connection, Statement, and ResultSet
- b. DriverManager, Driver, Connection, and Statement
- c. DriverManager, Statement, and ResultSet
- d. DriverManager, Connection, Statement, and ResultSet

**Answer:** a

Select the packages in which JDBC classes are defined?

- a. jdbc and javax.jdbc
- b. rdb and javax.rdb
- c. jdbc and java.jdbc.sql
- d. sql and javax.sql

**Answer:** d

Thin driver is also known as?

- a. Type 3 Driver
- b. Type-2 Driver
- c. Type-4 Driver
- d. Type-1 Driver

**Answer:** c

What is the correct sequence to create a database connection?

- i. Import JDBC packages.
  - ii. Open a connection to the database.
  - iii. Load and register the JDBC driver.
  - iv. Execute the statement object and return a query resultset.
  - v. Create a statement object to perform a query.
  - vi. Close the resultset and statement objects.
  - vii. Process the resultset.
  - viii. Close the connection.
- a. i, ii, iii, v, iv, vii, viii, vi

**MRITS-IT**

- b. i, iii, ii, v, iv, vii, vi, viii
- c. ii, i, iii, iv, viii, vii, v, vi
- d. i, iii, ii, iv, v, vi, vii, viii

**Answer:** b

Which of the following method is used to perform DML statements in JDBC?

- a. executeResult()
- b. executeQuery()
- c. executeUpdate()
- d. execute()

**Answer:** c

How many transaction isolation levels provide the JDBC through the Connection interface?

- a. 3
- b. 4
- c. 7
- d. 2

**Answer:** b

Which of the following method is static and synchronized in JDBC API?

- a. getConnection()
- b. prepareCall()
- c. executeUpdate()
- d. executeQuery()

**Answer:** A

Which methods are required to load a database driver in JDBC?

- a. getConnection()
- b. registerDriver()
- c. forName()
- d. Both b and c

**Answer:** d

Parameterized queries can be executed by?

**MRITS-IT**

- a. ParameterizedStatement
- b. PreparedStatement
- c. CallableStatement and Parameterized Statement
- d. All kinds of Statements

**Answer:** b

Which of the following is not a valid statement in JDBC?

- a. Statement
- b. PreparedStatement
- c. QueryStatement
- d. CallableStatement

**Answer:** c

Identify the isolation level that prevents the dirty in the JDBC Connection class?

- a. TRANSACTION\_READABLE\_READ
- b. TRANSACTION\_READ\_COMMITTED
- c. TRANSACTION\_READ\_UNCOMMITTED
- d. TRANSACTION\_NONE

**Answer:** b

What does setAutoCommit(false) do?

- a. It will not commit transactions automatically after each query.
- b. It explicitly commits the transaction.
- c. It never commits the transactions.
- d. It does not commit transaction automatically after each query.

\

**Answer:** b

Which JDBC driver can be used in servlet and applet both?

- a. Type 3
- b. Type 4
- c. Type 3 and Type 2
- d. Type 3 and Type 4

**MRITS-IT**

**Answer: d**

JDBC-ODBC driver is also known as?

- a. Type 4
- b. Type 3
- c. Type 1
- d. Type 2

**Answer: c**

Which of the following is not a type of ResultSet object?

- a. TYPE\_FORWARD\_ONLY
- b. CONCUR\_WRITE\_ONLY
- c. TYPE\_SCROLL\_INSENSITIVE
- d. TYPE\_SCROLL\_SENSITIVE

**Answer: b**

Which of the following services use TCP?

DHCP

SMTP

HTTP

TFTP

FTP

A. 1 and 2

B. 2, 3 and 5

C. 1, 2 and 4

D. 1, 3 and 4

**Answer:** Option **B**

Which of these package contains classes and interfaces for networking?

A. java.io

B. java.util

C. java.net

D. javax.swing

Discussion

C. java.net

2. In the following URL, identify the protocol identifier? <https://gtu.ac.in:8080/course.php>

**MRITS-IT**

- A. http
- B. gtu.ac.in
- C. //gtu.ac.in:80/course.php
- D. 8080

**Discussion**

A. http

3. Which of the following protocol follows connection less service?

- A. TCP
- B. TCP/IP
- C. UDP
- D. HTTP

**Discussion**

C. UDP

4. Which of the following statement is NOT true?

- A. TCP is a reliable but slow.
- B. UDP is not reliable but fast.
- C. File Transfer Protocol (FTP) is a standard Internet protocol for transmitting files between computers on the Internet over TCP/IP connections.
- D. In HTTP, all communication between two computers are encrypted

**Discussion**

D. In HTTP, all communication between two computers are encrypted

5. Which of the following statement is TRUE?

- A. With stream sockets there is no need to establish any connection and data flows between the processes are as continuous streams.
- B. Stream sockets are said to provide a connection-less service and UDP protocol is used
- C. Datagram sockets are said to provide a connection-oriented service and TCP protocol is used
- D. With datagram sockets there is no need to establish any connection and data flows between the processes are as packets.

**Discussion**

D. With datagram sockets there is no need to establish any connection and data flows between the processes are as packets.

6. Which of the following method call is valid to obtain the server's hostname by invoking an applet?

- A. getCodeBase().host()
- B. getCodeBase().getHost()
- C. getCodeBase().hostName()
- D. getCodeBase().getHostName()

**Discussion**

B. getCodeBase().getHost()

**MRITS-IT**

7. The server listens for a connection request from a client using which of the following statement?

- A. Socket s = new Socket(ServerName, port);
- B. Socket s = serverSocket.accept()
- C. Socket s = serverSocket.getSocket()
- D. Socket s = new Socket(ServerName);

Discussion

B. `Socket s = serverSocket.accept()`

8. The client requests a connection to a server using which of the following statement?

- A. Socket s = new Socket(ServerName, port);
- B. Socket s = serverSocket.accept();
- C. Socket s = serverSocket.getSocket();
- D. Socket s = new Socket(ServerName);

Discussion

A. `Socket s = new Socket(ServerName, port);`

9. To connect to a server running on the same machine with the client, which of the following cannot be used for the hostname?

- A. "localhost"
- B. "127.0.0.1"
- C. InetAddress.getLocalHost(),
- D. "127.0.0.0"

Discussion

D. "127.0.0.0"

10. In the socket programming, for an IP address, which can be used to find the host name and IP address of a client/ server?

- A. The ServerSocket class
- B. The Socket class
- C. The InetAddress class
- D. The Connection interface

Discussion

C. The InetAddress class

11. To create an InputStream on a socket, say s, which of the following statement is necessary?

- A. `InputStream in = new InputStream(s);`
- B. `InputStream in = s.getInputStream();`
- C. `InputStream in = s.obtainInputStream();`
- D. `InputStream in = s.getStream();`

Discussion

B. `InputStream in = s.getInputStream();`

**MRITS-IT**

12. Which of the following protocols is/are for splitting and sending packets to an address across a network?

- A. TCP/IP
- B. FTP
- C. SMTP
- D. UDP

**Discussion**

A. TCP/IP

13. Which of these is a protocol for breaking and sending packets to an address across a network?

- A. TCP/IP
- B. DNS
- C. Socket
- D. Proxy Server

**Discussion**

A. TCP/IP

14. Which of these class is used to encapsulate IP address and DNS?

- A. DatagramPacket
- B. URL
- C. InetAddress
- D. ContentHandler

**Discussion**

C. InetAddress

15. Which of the following type of JDBC driver, is also called Type 2 JDBC driver?

- A. JDBC-ODBC Bridge plus ODBC driver
- B. Native-API, partly Java driver
- C. JDBC-Net, pure Java driver
- D. Native-protocol, pure Java driver

**Discussion**

B. Native-API, partly Java driver

16. Which of the following type of JDBC driver, is also called Type 1 JDBC driver?

- A. JDBC-ODBC Bridge plus ODBC driver
- B. Native-API, partly Java driver
- C. JDBC-Net, pure Java driver
- D. Native-protocol, pure Java driver

**Discussion**

A. JDBC-ODBC Bridge plus ODBC driver

17. Which of the following holds data retrieved from a database after you execute an SQL

**MRITS-IT**

query using Statement objects?

- A. ResultSet
- B. JDBC driver
- C. Connection
- D. Statement

Discussion

A. ResultSet

18. Which of the following is not a valid type of ResultSet?

- A. ResultSet.TYPE\_FORWARD\_ONLY
- B. ResultSet.TYPE\_SCROLL\_INSENSITIVE
- C. ResultSet.TYPE\_SCROLL\_SENSITIVE
- D. ResultSet.TYPE\_BACKWARD\_ONLY

Discussion

D. ResultSet.TYPE\_BACKWARD\_ONLY

19. Which of the following type of JDBC driver, uses database native protocol?

- A. JDBC-ODBC Bridge plus ODBC driver
- B. Native-API, partly Java driver
- C. JDBC-Net, pure Java driver
- D. Native-protocol, pure Java driver

Discussion

D. Native-protocol, pure Java driver

20. What is JDBC?

- A. JDBC is a java based protocol.
- B. JDBC is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.
- C. JDBC is a specification to tell how to connect to a database.
- D. Joint Driver for Basic Connection

Discussion

B. JDBC is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

21. Which of the following manages a list of database drivers in JDBC?

- A. DriverManager
- B. JDBC driver
- C. Connection
- D. Statement

Discussion

A. DriverManager

22. Which of the following type of JDBC driver should be used if your Java application is  
**MRITS-IT**

accessing multiple types of databases at the same time?

- A. Type 1
- B. Type 2
- C. Type 3
- D. Type 4

Discussion

- C. Type 3

23. Which of the following is correct about JDBC?

- A. JDBC architecture decouples an abstraction from its implementation.
- B. JDBC follows a bridge design pattern.
- C. Both of the above.
- D. None of the above.

Discussion

- C. Both of the above.

24. Which of the following step establishes a connection with a database?

- A. Import packages containing the JDBC classes needed for database programming.
- B. Register the JDBC driver, so that you can open a communications channel with the database.
- C. Open a connection using the DriverManager.getConnection () method.
- D. Execute a query using an object of type Statement.

Discussion

- C. Open a connection using the DriverManager.getConnection () method.

25. Which of the following is true about JDBC?

- A. The JDBC API is an API to access different relational databases.
- B. You use it to access relational databases without embedding a dependency on a specific database type in your code.
- C. JDBC stands for Java DataBase Connectivity.
- D. All of the above.

Discussion

- D. All of the above.

## **UNIT-4**

Which of these functions is called to display the output of an applet?

- a) display()
- b) paint()
- c) displayApplet()
- d) PrintApplet()

View Answer

Answer: b

**MRITS-IT**

2. Which of these methods can be used to output a string in an applet?

- a) display()
- b) print()
- c) drawString()
- d) transient()

[View Answer](#)

Answer: c

3. Which of these methods is a part of Abstract Window Toolkit (AWT) ?

- a) display()
- b) paint()
- c) drawString()
- d) transient()

[View Answer](#)

Answer: b

Which of these modifiers can be used for a variable so that it can be accessed from any thread or parts of a program?

- a) transient
- b) volatile
- c) global
- d) No modifier is needed

[View Answer](#)

Answer: b

5. Which of these operators can be used to get run time information about an object?

- a) getInfo
- b) Info
- c) instanceof
- d) getinfoof

[View Answer](#)

Answer: c.

6. What is the Message is displayed in the applet made by the following Java program?

```
1. import java.awt.*;
2. import java.applet.*;
3. public class myapplet extends Applet
4. {
5. public void paint(Graphics g)
6. {
7. g.drawString("A Simple Applet", 20, 20);
8. }
9. }
```

- a) A Simple Applet
- b) A Simple Applet 20 20
- c) Compilation Error
- d) Runtime Error

[View Answer](#)

Answer: a.

Output:

A Simple Applet

(Output comes in a new java application)

**MRITS-IT**

7. What is the length of the application box made by the following Java program?

```
1. import java.awt.*;
2. import java.applet.*;
3. public class myapplet extends Applet
4. {
5. public void paint(Graphics g)
6. {
7. g.drawString("A Simple Applet", 20, 20);
8. }
9. }
```

- a) 20
- b) 50
- c) 100
- d) System dependent

[View Answer](#)

Answer: a

8. What is the length of the application box made the following Java program?

```
1. import java.awt.*;
2. import java.applet.*;
3. public class myapplet extends Applet
4. {
5. Graphic g;
6. g.drawString("A Simple Applet", 20, 20);
7. }
```

- a) 20
- b) Default value
- c) Compilation Error
- d) Runtime Error

[View Answer](#)

Answer: c

9. What will be the output of the following Java program?

```
1. import java.io.*;
2. class Chararrayinput
3. {
4. public static void main(String[] args)
5. {
6. String obj = "abcdefgh";
7. int length = obj.length();
8. char c[] = new char[length];
9. obj.getChars(0, length, c, 0);
10. CharArrayReader input1 = new CharArrayReader(c);
11. CharArrayReader input2 = new CharArrayReader(c, 1, 4);
12. int i;
13. int j;
14. try
15. {
16. while((i = input1.read()) == (j = input2.read()))
```

**MRITS-IT**

```
17. {
18. System.out.print((char)i);
19. }
20. }
21. catch (IOException e)
22. {
23. e.printStackTrace();
24. }
25. }
26. }
```

- a) abc
- b) abcd
- c) abcde
- d) none of the mentioned

[View Answer](#)

Answer: d

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

*Answer : B*

Q 2 - if isThreadSafe attribute of page directive is set as true, then generated servlet implements SingleThreadModel interface.

A - True

B - False

*Answer : B*

Q 3 - Which of the following do not supports JSP directly?

A - Weblogic Server

B - WebSphere Server

C - Tomcat Server

D - Apache HTTP Server

*Answer : D*

Q 4 - Which of the following is true about language attribute?

A - The language attribute indicates the programming language used in scripting the servlet.

B - The language attribute indicates the programming language used in scripting the html page.

C - The language attribute indicates the programming language used in scripting the JSP page.

D - None of the above.

*Answer : C*

Q 5 - What is true about filters?

**MRITS-IT**

A - JSP Filters are used to intercept requests from a client before they access a resource at back end.

B - JSP Filters are used to manipulate responses from server before they are sent back to the client.

C - Both of the above.

D - None of the above.

*Answer : C*

Q 6 - Which of the following is true about import Attribute?

A - The import attribute serves the same function as, and behaves like, the Java import statement.

B - The value for the import option is the name of the package you want to import.

C - Both of the above.

D - None of the above.

*Answer : C*

Q 7 - Which of the following is true about isELIgnored Attribute?

A - The isELIgnored option gives you the ability to disable the evaluation of Expression Language (EL) expressions.

B - The default value of the isELIgnored attribute is true.

C - Both of the above.

D - None of the above.

*Answer : C*

Q 8 - Which of the following is true about <jsp:forward> action?

A - The forward action terminates the action of the current page.

B - The forward action forwards the request to another resource such as a static page, another JSP page, or a Java Servlet.

C - Both of the above.

D - None of the above.

*Answer : C*

Q 9 - Which of the following is true about locale?

A - This is a particular cultural or geographical region.

B - Locale is usually referred to as a language symbol followed by a country symbol which are separated by an underscore.

C - Both of the above.

D - None of the above.

*Answer : C*

Q 10 - Which of the following is true about <c:choose> tag?

A - The <c:choose> works like a Java switch statement in that it lets you choose between a number of alternatives.

B - The <c:choose> tag has <c:when> tags.

C - The <c:choose> tag has <otherwise> as default clause.

**MRITS-IT**

D - All of the above.

*Answer : D*

## **UNIT-5**

1. What Does XML Stand For?

EXtra Modern Link

EXtensible Markup Language

Example Markup Language

X-Markup Language

VDO.AI

[View Answer](#)

EXtensible Markup Language

2. Which Statement Is True?

All XML Elements Must Have A Closing Tag

All XML Documents Must Have A DTD

All XML Elements Must Be Lower Case

All The Statements Are True

[View Answer](#)

All XML Elements Must Have A Closing Tag

3. Which Of The Following Programs Support XML Or XML Applications?

Netscape 4.7

Internet Explorer 5.5

RealPlayer

Both A & B

[View Answer](#)

Both A & B

4. Which Of The Following Strings Are A Correct XML Name?

#myElement

My Element

\_myElement

None Of The Above

[View Answer](#)

\_myElement

5. Which Allows Hyperlinks To Point To Specific Parts (fragments) Of XML Documents?

XPath

Xpointer

XSLT

XLink

[Download Free : Xml MCQ PDF](#)

[View Answer](#)

Xpointer

6. Which Is Not A XML Function?

Transport Information

Style Information

Store Information

Structure Information

[View Answer](#)

Style Information

7. Which Is Not A W3C-recommended Specification?

SAX

DOM

Both A & B

**MRITS-IT**

None Of The Above

[View Answer](#)

SAX

8. Which Internet Language Is Used For Describing Available Web Services In XML?

RSS

RDF

WSDL

OWL

[View Answer](#)

WSDL

9. In XML Document Comments Are Given As?

<!-- --!>

<!-- ?

</-- -->

<?-- ?

[View Answer](#)

<!-- ?

10. XML Is Designed To Store Data And \_\_\_\_\_.

Design

Verify

Both A & B

Transport

[Read Best: Xml interview Questions](#)

[View Answer](#)

Transport

11. The Correct Tag Is

[View Answer](#)

12. In Schema Data Type Can Be Specified Using

Type

DataType

Dt:type

Data:type

[View Answer](#)

Dt:type

13. Which Is/are The Example(s) Of XML-based Multimedia Language?

SMIL

MULTIMEDIA DESCRIPTION LANGUAGE

HTML+TIME

Both A & B

[View Answer](#)

HTML+TIME

14. Which Vocabulary Provides Supports To Build Links Into XML?

XLink

XPointer

XSL

XHTML

[View Answer](#)

Xlink

15. Which Of The Following Is Not An Online XML Validator?

Tidy

**MRITS-IT**

Expat

XML.com's

W3C Validation Service

Download Free: Xml interview Questions PDF

[View Answer](#)

Expat

16. Which Language Is Case Sensitive?

XML

HTML

Both A & B

None Of The Above

[View Answer](#)

XML

17. The XSL formatting object which formats the data in a table -

table-body

table

table-content

title

[View Answer](#)

table

18. XSL has “ block container” for formating the document to -

Create a display block to format the paragraphs

Create a display block to format the titles

Create a block level reference area

Create a display block to format the titles

[View Answer](#)

Create a block level reference area

19. Which of the following is a valid XSLT iteration command -

for-all

for-each

for

in-turn

[View Answer](#)

for-each

20. What is a NCName?

A Non-Colonized Name

A Non-Common Name

A Non-Conforming Name

None of the above

[View Answer](#)

A Non-Colonized Name

21. Is it easier to process XML than HTML?

No

Yes

[View Answer](#)

Yes

22. Well formed XML document means .....

it contain an element

it contains a root element

it contains one or more elements

must contain one or more elements and root element must contain all other elements

[View Answer](#)

must contain one or more elements and root element must contain all other elements

23. XML uses the features of .....

**MRITS-IT**

VML

SGML

HTML

XHTML

[View Answer](#)

SGML

24. What does DTD stand for?

Do the Dance

Direct Type Definition

Document Type Definition

Dynamic Type Definition

[View Answer](#)

Document Type Definition

25. The XML DOM object is .....

Entity

Comment Data

Entity Reference

Comment Reference

[View Answer](#)

Entity Reference

26. An xml database supports the storage and management of \_\_\_\_\_ xml data.

structured

semistructured

multistructured

None of the above

[View Answer](#)

semistructured